Interaktivna računalna grafika kroz primjere u OpenGL-u

Marko Čupić Željka Mihajlović

1. travnja 2021.

Sadržaj

Izvorni kodovi programa

Predgovor

Ovaj dokument predstavlja radnu verziju knjige iz računalne grafike: *Interaktivna računalna grafika kroz primjere u OpenGL-u.* Molimo sve pogreške, komentare, nejasnoće te sugestije dojaviti na
Marko.Cupic@fer.hr ili Zeljka.Mihajlovic@fer.hr.

Verzija dokumenta: 0.1.2019-05-17.

Poglavlje 0

Uvod

0.1 Kome je namijenjena ova knjiga

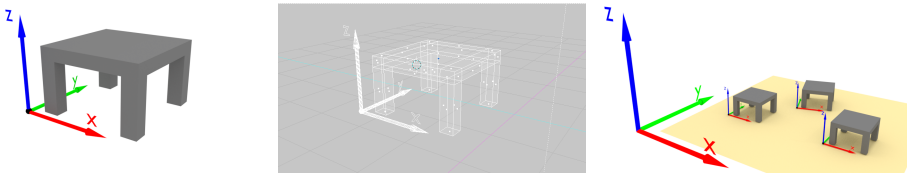
Knjiga je namijenjena svim studentima koji se žele upoznati s osnovama izrade prikaza 2D i 3D scena na računalu. Iako je dio knjige posvećen ponavljanju odnosno uvođenju relevantnog matematičkog aparata, očekuje se od čitatelja da dobro vladaju elementarnim znanjem linearne algebre (što su vektori i kako se nad njima provode različite operacije, što su matrice, kako se zbrajaju, množe, transponiraju, invertiraju i slično) te analitičke geometrije (pojam pravca i ravnine, načini traženja sjecišta i slično).

Očekuje se i razumno poznavanje nekog od konkretnih programskih jezika. U knjizi je dan niz primjera u programskom jeziku C ali i naputak kako složiti osnovne grafičke programe u drugim programskim jezicima.

0.2 Prikaz 3D-objekata na računalu

Kako bi opisali bilo kakav objekt odnosno trodimenzionalno tijelo na računalu, potrebno je napraviti njegovu matematičku reprezentaciju. Najčešće se matematička reprezentacija objekta temelji na opisu plašta, odnosno vanjskih ploha objekta koje ga omeđuju. Pri tome se za opisivanje ploha najčešće koriste poligoni, posebice oni najjednostavniji: trokuti. Tako, primjerice, želimo li prikazati stolić, potrebno je odrediti poligone koji čine plašt stolića. Da bismo specificirali svaki od poligona, potrebno je odrediti pripadne točke koje čine vrhove poligona. Zajedno, vrhovi i poligoni predstavljaju matematičku reprezentaciju objekta. Na slici 1a prikazan je primjer modela stolića napravljen korištenjem poligona i vrhova koji čine plašt stolića. Koordinatni sustav u kojem su zadani vrhovi i poligoni objekta naziva se *lokalni koordinatni sustav objekta.* Na slici 1a taj je koordinatni sustav prikazan razapetim strelicama crvene, zelene i plave

boje. Koordinate vrhova poligona objekta često se zadaju u normaliziranom rasponu od 0 do 1. Prikaz objekata poligonima naziva se *žična forma*; za stolić iz primjera, ovaj prikaz možemo vidjeti na slici 1b.



(a) lokalni koordinatni sustav (b) žična forma (c) globalni koordinatni sustav

Slika 1: Prikaz stolića u lokalnom koordinatnom sustavu, žična forma objekata u sceni te globalni koordinatni sustav s tri stolića.

U sceni vrlo često trebamo više istih objekata koji su na različitim pozicijama i mogu biti različito orijentirani. Primjerice, zamislimo da scena prikazuje prostoriju u kojoj se nalazi desetak stolića te na svakom stoliću piće. Koordinatni sustav scene u kojem je svaki od stolića smješten na različitom položaju naziva se *globalni koordinatni sustav* (slika 1c). Na slici 1c prikazan je globalni koordinatni sustav u kojem su prikazana tri stolića, pri čemu je uz svaki stolić prikazan i njegov lokalni koordinatni sustav. Globalni koordinatni sustav još se naziva i sustavom scene ili kraće scenom. Razlikovanje lokalnog i globalnog koordinatnog sustava omogućava nam da svaki različit objekt modeliramo samo jednom (primjerice, u nekom od alata za 3D modeliranje napravimo njegov opis u njegovom lokalnom koordinatnom sustavu, odnosno definiramo sve njegove poligone) i potom taj objekt dodamo u scenu koliko god puta je to potrebno, pri čemu svakom primjerku objekta moramo pridružiti transformaciju kojom određujemo gdje je u globalnom koordinatnom sustavu smješteno ishodište njegovog lokalnog koordinatnog sustava (takozvana transformacija *translacije*), treba li i kako promijeniti dimenzije objekta skaliranjem koordinatnih osi lokalnog koordinatnog sustava (takozvana transformacija *skaliranja*) te treba li i kako pri smještaju objekta osi lokalnog koordinatnog sustava zarotirati (takozvana transformacija *rotacije*). Uz navedene transformacije, moguće je napraviti i druge poput smika, zrcaljenja itd. Ovo će pak omogućiti da prilikom definiranja scene koja sadrži niz stolića, u memoriji računala možemo imati samo jednu definiciju stolića (samo na jednom mjestu u memoriji pamtimo poligone stolića), a svaki će primjerak stolića imati referencu na taj opis te pridruženu transformaciju. S obzirom da kompleksniji objekti mogu biti definirani uporabom tisuća poligona, ovime se ujedno postižu i značajnije uštede u potrošnji memorije.

Za zapisivanje transformacija poput pomaka, rotacije i drugih, na računalu se uobičajeno koristi matrični oblik o čemu će biti riječi u daljnjim poglavljima

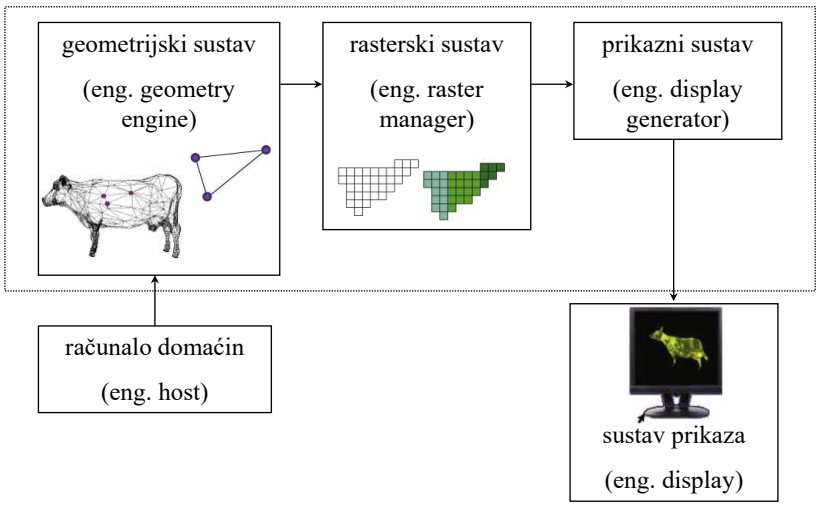
knjige. Programski, napraviti translaciju ili rotaciju znači da je potrebno odrediti matricu s kojom treba pomnožiti sve vrhove stolića da bi dobili njihove koordinate u sustavu scene. Matrica kojom se vrhovi objekta iz lokalnog koordinatnog sustava prebacuju u globalni koordinatni sustav naziva se *matrica transformacije modela*.

Da bismo u sceni mogli vidjeti objekte, potreban je i barem jedan izvor svjetla. Potom, uzimajući u obzir karakteristike materijala koje su pridružene poligonima te uzimajući u obzir broj, smještaj i karakteristike korištenih izvora, potrebno je izračunati kako treba obojiti pojedine poligone objekta, a da pri tome dobijemo dojam da su objekti (realistično) osvijetljeni postavljenim izvorima svjetlosti. Ovaj postupak naziva se *osvjetljavanjem*.

Konačno, kako bi prikazali scenu na zaslonu računala, potrebno je izraditi dvodimenzionalan prikaz scene. Zamislite samo što se događa kada fotoaparatom slikate: dobivate fotografiju odnosno dvodimenzionalan prikaz dijela trodimenzionalnog prostora. Govorimo li o zaslonu računala, dodatno u obzir treba uzeti i činjenicu da je zaslon izgrađen od *diskretnog niza slikovnih elemenata*, odnosno da je prikaz slike *rasterski*. Za zadanu scenu, to znači da za svaki objekt koji smo opisali vrhovima i poligonima u trodimenzionalnom prostoru scene, a koji su u ovom trenutku već smješteni u globalni koordinatni sustav, još trebamo odrediti njihovu projekciju u dvodimenzionalnoj ravnini projekcije. Kada stvarnom kamerom snimamo okolni prostor, radimo projekciju 3D-svijeta u 2D-prostor projekcije. Analogno želimo napraviti i na računalu: zanima nas projekcija svih objekata scene u ravninu projekcije, odnosno u našem slučaju projekcija svakog od stolića.

Za prikaz objekta u ravnini projekcije potrebno je napraviti dva koraka. Prvi korak je određivanje koordinata stolića u koordinatnom sustavu kamere. Naime, da bismo odredili kako izgleda prikaz scene kroz kameru, najprije moramo znati gdje je u prostoru kamera smještena i u kojem smjeru je usmjerena. Znamo li to, možemo se pitati koje su koordinate svih objekata koji čine scenu mjerene u trodimenzionalnom koordinatnom sustavu čije je ishodište sama kamera. Postupak kojim se to utvrđuje naziva se *transformacija pogleda*. Drugi korak je *projekcija* odnosno određivanje koordinata u 2D ravnini projekcije temeljem 3D koordinata objekta koje su sada u koordinatnom sustavu kamere. Položaj gdje se nalazi kamera nazivamo *očište*, a točka u koju je usmjeren pogled nazivamo *gledište*. Cijeli opisani postupak dobivanja slike našeg objekta naziva se *izrada prikaza* (engl. *rendering*). Prolazak poligona i vrhova kroz prethodno opisane korake opisujemo grafičkim protočnim sustavom.

Grafički protočni sustav (engl. *graphics pipeline*, *rendering pipeline*) je konceptualni model koji opisuje osnovne korake odnosno faze izvođenja u postupku ostvarivanja prikaza. Grafički protočni sustav je logička razina koju možemo preslikati na fizičku razinu na različite načine ovisno o tome kako će ista biti realizirana. Da li će se pojedini koraci obavljati na centralnom procesoru (CPU),



Slika 2: Grafički protočni sustav

na grafičkoj kartici (GPU) na sjenčaru vrhova (engl. *vertex shader*) ili na sjenčaru fragmenata (engl. *fragment shader*) ili pak na nekom udaljenom računalu, u ovom trenutku nećemo razmatrati.

Grafički protočni sustav možemo podijeliti na tri cjeline: prva je geometrijski sustav, druga je rasterski sustav, a treća je sustav prikaza (slika 2). Pokretanjem grafičke aplikacije na računalu domaćinu (engl. *host*), dijelovi vezani za grafiku šalju se u geometrijski sustav. U geometrijskom sustavu izdvajaju se vrhovi nad kojima radimo transformaciju koordinata i izračunavamo osvijetljenje objekta (engl. *Transform and Lighting*, T&L). To znači da množimo vrhove objekta matricom transformacije modela, određujemo transformaciju pogleda te transformaciju projekcije. Također provodimo izračune potrebne za osvjetljavanje objekta. Najčešće ovu funkcionalnost obavlja sjenčar vrhova. Sjenčar vrhova je zapravo mali program koji se izvodi na GPU i izvodi se za svaki vrh. Nakon prolaska kroz geometrijski protočni sustav, vrhovi našeg objekta koji su izvorno zadani u 3D prostoru u formatu s pomičnim zarezom (engl. *floating point*) bit će transformirani u 2D prostor projekcije. Koordinate će i dalje biti s pomičnim zarezom.

Osnovna funkcionalnost rasterskog sustava je *rasterizacija* poligona i linija. Kako bismo poligon odnosno liniju mogli prikazati na napravi za prikaz, potrebno je odrediti sve slikovne elemente (njihove diskretne koordinate) koji pripadaju slici tog poligona odnosno linije te pripadnu boju za svaki slikovni element. Pri tome treba biti svjestan da u svakom postupku diskretizacije, odnosno prevođenju iz kontinuiranog oblika u diskretni oblik, dolazi do neželjenog učinka (engl. *aliasing*) koji se kod prikaza objekata primarno manifestira kao nazubljenje rubova

objekta. Što je veća rezolucija slike, ovaj učinak dolazi manje do izražaja, no nije ga moguće u potpunosti ukloniti. U postupku rasterizacije možemo raditi i različite operacije nad elementima teksture koju pridružujemo poligonima pa se ove operacije rade tipično u sjenčaru fragmenata. Nakon rasterizacije moguće je raditi operacije s rasterskim prikazom (engl. *raster operation pipeline*, ROP), a uobičajene su operacije koje koriste spremnik maske, prozirnost stapanje slike i slično. U konačnici, sliku je potrebno zapisati u *slikovnu prikaznu memorija* (engl. *frame buffer*) iz koje se radi prikaz na sustav prikaza (engl. *display*).

Na kraju, da bi sliku prikazali na sustavu prikaza, potrebno je još proći kroz prikazni sustav. *Prikazni sustav* vezan je za napravu na kojoj ćemo prikazivati sliku. Za ostvarivanje konačnog prikaza potrebno je osigurati podršku za različite standarde koje prihvaća prikazna jedinica (VGA, HDMI, DVI) ili podršku za više sustava prikaza, ako su nam na raspolaganju.

0.3 Organizacija knjige

Za kraj uvodnog poglavlja osvrnimo se još i na organizaciju ostatka knjige.

U poglavlju 1 dan je kratak osvrt na izradu grafičke aplikacije uporabom biblioteka sukladnih specifikaciji OpenGL inačice 2.0. Ta se inačica umjesto naprednijih koje se izravno temelje na uporabi grafičkih sjenčara koristi kako bi se čitatelja na što jednostavniji način uvelo u osnovne koncepte OpenGL-a.

U poglavlju 2 prolazi se kroz matematičke osnove koje čine temelj računalne grafike: dvodimenzionalan prostor i trodimenzionalan prostor, specificiranje pravca i ravnine, izračun probodišta pravca i raznih objekata, baricentrične koordinate, homogeni prostor i homogene koordinate i slično.

Poglavlje 3 daje elementarni prikaz interpolacija. Razmatramo linearnu interpolaciju, interpolaciju polinomima, bilinearnu i trilinearnu interpolaciju te interpolaciju vektora.

Poglavlje 4 razmatra problem rasterizacije. Kroz to poglavlje analiziraju se načini prikazivanja linija i poligona na rasterskim prikaznim jedinicama. Razmatraju se postupci za popunjavanje poligona, utvrđivanje orijentacije poligona te utvrđivanje odnosa točke i poligona.

U poglavlju 5 razmatra se matematičko modeliranje geometrijskih transformacija u 2D i 3D-prostoru te njihov matrični prikaz. Razmatraju se transformacije translacije, rotacije, skaliranja i smika te transformacije kojima se prelazi iz jednog koordinatnog sustava u drugi.

Poglavlje 6 donosi pregled transformacija pogleda te projekcija. Razmatra se uporaba view-up vektora, paralelne projekcije te perspektivne projekcije. Dan je i osvrt na provedbu ovih koraka kroz OpenGL.

U poglavlju 7 razmatra se problematika zadavanja, i prikaza krivulja. Daje se njihova klasifikacija, parametarski prikaz, parametarski prikaz uporabom po-

linoma te prikaz pomoću razlomljenih funkcija. Kao važna porodica krivulja detaljnije se razmatra Bézierova krivulja. Dan je i osvrt na uporabu Bézierovih krivulja kroz prikaz TrueType fontova te za specificiranje površina.

Poglavlje 8 razmatra problematiku učinkovitog prikaza scena koje se sastoje od mnoštva kompleksnih objekata. Razmatraju se postupci koji omogućavaju brzu detekciju objekata ili njihovih dijelova koje nije potrebno iscrtavati, postupci za organizaciju podataka o sceni koji omogućavaju brzu provedbu različitih ispitivanja te postupci koji se mogu koristiti za ispravno generiranje prikaza trodimenzionalne scene.

Poglavlje 9 analizira postupke osvjetljavanja i sjenčanja. Razmatra se Phongov model osvjetljavanja te tri postupka sjenčanja: konstantno, Gouraudovo i Phongovo.

Poglavlje 10 fokusira se na dobivanje realističnijih prikaza scene kroz globalne modele osvjetljavanja. Dan je prikaz algoritma bacanja zrake te algoritma praćenja zrake.

Poglavlje 11 razmatra specificiranje boja na računalu. Razmatraju se različiti modeli poput RGB, XYZ, CMY/CMYK i HSL.

Poglavlje 12 uvodi pojam tekstura. Pojašnjava se što su teksture i kako se koriste pri prikazu objekata.

Poglavlje 13 završava knjigu s pričom o jednom vrlo lijepom području računalne grafike: generiranju fraktalnih prikaza. Razmatra se pojam fraktala te nekoliko vrsta fraktala. Razmatraju se svojstva fraktala i daju primjeri fraktala koji imaju beskonačan opseg a konačnu površinu. Razmatra se pojam fraktalne dimenzije.

Na kraju knjige nalaze se i 3 dodatka. U dodatku A dana je dopuna diskusije u transformacijama pogleda i uporabi view-up vektora. U dodatku B dodatno se diskutira o krivuljama te traženju presjecišta zrake i krivulja. Dodatak C sadrži naputak za izradu najjednostavnijih grafičkih aplikacija u nekoliko programskih jezika te operacijskih sustava.

Poglavlje 1

Osnove OpenGL-a

1.1 Što je OpenGL

Kada govorimo o računalnoj grafici na modernim računalnima, o vizualizaciji ili jednostavno o igranju igara, dva pojma koja se odmah pojavljuju su *OpenGL* i *DirectX*. Oba pojma odnose se na specifikacije (norme) koje danas omogućavaju rad s grafičkim karticama, a u svrhu crtanja 2D i 3D scena. Kako je od te dvije norme *OpenGL* široko prihvaćena i višeplatformska specifikacija, u ovoj knjizi primjeri će biti ilustrirani upravo kroz *OpenGL*, čije je ime kratica od ***Open G**raphics **L**ibrary*.

Za *OpenGL* možemo reći da je višeplatformska specifikacija s podrškom za niz programskih jezika, a čiji je cilj omogućiti pisanje aplikacija koje rade s 2D i 3D grafikom. Uz *OpenGL* možemo vezati još atributa, poput sklopovski neovisna specifikacija, specifikacija neovisna o operacijskom sustavu te specifikacija koja nije vezana uz jednog pojedinačnog proizvođača. *OpenGL* specifikacija definira niz primitiva koji se koriste za izgradnju složenih scena (poput točke, linije i poligona). Također, *OpenGL* nudi mogućnost primjene različitih transformacija nad objektima u sceni, nudi različite vrste projekcija, nudi odbacivanje dijelova objekata koji su promatraču nevidljivi, primjenu tekstura i još niz drugih mogućnosti. Sa stanovišta programera, *OpenGL* je jedan veliki stroj stanja. To znači da, primjerice, jednom kada definiramo boju kojom se crta, svi objekti koji se pošalju na crtanje koristit će upravo navedenu boju – tako dugo dok je ne promijenimo. Isto vrijedi i za sve ostale dijelove *OpenGL*-a.

Međutim, *OpenGL* specifikacija ne miješa se u rad s prozorima, što je danas jedan od temeljnih zadataka operacijskih sustava s grafičkim korisničkim sučeljem. Stoga se uz *OpenGL* tipično koriste još dvije pomoćne biblioteke: *GLU* i *GLUT*.

Biblioteka *GLU* (što je kratica od *Open**GL** **U**tility Library*) obogaćuje skup naredbi koje pruža *OpenGL* i uvodi kompleksnije primitive koji je moguće ko-

ristiti u opisu scena; jedan od primjera su NURBS krivulje i površine (engl. *Non-Uniform Rational Basis Spline*) te sfere, cilindri i stošci. Ova biblioteka uobičajeno dolazi sa svim instalacijama *OpenGL*-a, i nije ju potrebno zasebno instalirati.

Biblioteka *GLUT* (što je kratica od *Open**GL** **U**tility **T**oolkit*) dodatno pojednostavljuje izradu aplikacija koje koriste *OpenGL* uvođenjem platformski neovisne podrške za stvaranje i rad s prozorima, za hvatanje i obradu niza događaja (poput događaja vezanih uz miša i tipkovnicu), za izradu izborničkih struktura (engl. *menus*), i sl. Biblioteka donosi i definicije još nekih složenih objekata koje programeru stavlja na raspolaganje, poput torusa i čajnika. Ova biblioteka nije standardni dio *OpenGL* instalacija, i bit će je potrebno doinstalirati.

1.2 Prvi program

U ovom poglavlju opisat ćemo što je potrebno napraviti kako bismo preveli i pokrenuli naš prvi program koji će koristiti OpenGL. Program ćemo napisati u programskom jeziku C++, na operacijskom sustavu Windows. Ovo međutim neće biti nikakvo ograničenje, jer na gotovo identičan način program možemo prevesti i na operacijskom sustavu Linux. Detaljniji opis kako napraviti pripremu dostupan je u dodatku C. Stoga se preporuča svima koji rade u primjerice u razvojnim okolinama *Microsoft Visual Studio* i drugima, da najprije prouče dodatak C. U nastavku ove sekcije pogledat ćemo to za jedan konkretan primjer: program pisan u jeziku C++ koji želimo prevesti i pokrenuti na operacijskom sustavu *Microsoft Windows*.

1.2.1 Priprema prevodioca

Za potrebe ove knjige pretpostavit ćemo da ste instalirali gcc prevodioc (primjerice, *MinGW*). U konkretnom primjeru, prevodioc je instaliran u `D:\usr\MinGW`. Ovaj prevodioc može se skinuti kao ZIP arhiva, i to je u ovom slučaju i napravljeno. Direktorij `D:\usr\MinGW\bin` dodan je u varijablu okruženja `PATH`. Ako to niste napravili, tada je prije poziva prevodioca potrebno zadati naredbu:

```
SET "PATH=%PATH%;D:\usr\MinGW\bin"
```

Staza koja se pri tome koristi mora odgovarati direktoriju u koji ste raspakirali MinGW.

1.2.2 Priprema pomoćne biblioteke

S obzirom da je OpenGL biblioteka temeljena na prilično niskoj razini, razvijene su pomoćne biblioteke koje olakšavaju njegovu uporabu. Najprije je napravljena

biblioteka *glut*, a nakon nje i biblioteka *freeglut* koja nudi jednostavnu zamjenu za ovu prvu. Razlog razvoja biblioteke *freeglut* leži u problemima oko licence te činjenici da se biblioteka *glut* više ne razvija, što u današnje doba predstavlja problem. Stoga ćemo u okviru ove knjige sve primjere raditi koristeći biblioteku *freeglut* (uz napomenu da će primjeri raditi bez većih problema i s bibliotekom *glut*).

Biblioteku *freeglut* možete skinuti s adrese[1]. Na njoj ćete naći link na stranicu *Martin Payne's Windows binaries* gdje je potrebno skinuti najnoviju inačicu ZIP arhive *freeglut MSVC Package*. U toj arhivi se nalaze dva direktorija: `include` i `lib`, te biblioteka `freeglut.dll`.

Ako ipak želite koristiti biblioteku *glut*, do nje možete doći na stranici *Nate Robins Main OpenGL Chronicles Allies*[2]. Tamo potražite `glut-3.7.6-bin.zip` (ili noviju ZIP-arhivu) koja sadrži sve potrebne datoteke.

Glut je prilično korisna biblioteka (*OpenGL Utility Toolkit*) koja olakšava interaktivni rad i općenito pisanje OpenGL aplikacija jer sam OpenGL ne daje podršku za rad s prozorima. Načinjene aplikacije time su neovisne o platformi na kojoj se izvode a istovremeno koriste operacije kao što su zadavanje koordinata mišem u prozoru, pomicanje objekata interakivno mišem i slično. Time će načinjena aplikacija biti izvediva na različitim operacijskim sustavima.

1.2.3 Priprema strukture direktorija

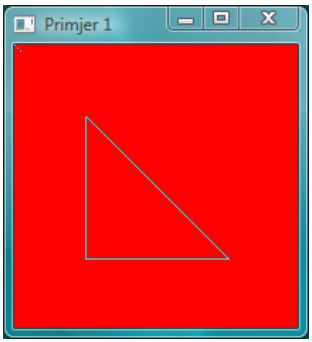
Nakon što ste nabavili ZIP arhivu biblioteke *freeglut*, spremni smo za izradu našeg prvog programa. Na disku ćemo napraviti direktorij `primjer1`. U taj direktorij ćemo iz ZIP arhive iskopirati direktorije `include` i `lib` i biblioteku `freeglut.dll`. Ako ste ovo dobro napravili, trebali biste dobiti strukturu direktorija kako je prikazano u nastavku.

```
primjer1
  include
    GL
      freeglut.h
      freeglut_ext.h
      freeglut_std.h
      glut.h
  lib
    freeglut.lib
  freeglut.dll
```

Ako ste se odlučili na korištenje biblioteke *glut*, pripadna ZIP arhiva nema direktorije `include` i `lib`, već sve datoteke sadrži na jednom mjestu. U tom

[1] `http://freeglut.sourceforge.net/`

[2] `http://www.xmission.com/~nate/glut.html`



Slika 1.1: Prvi OpenGL program

slučaju naprije sami u direktoriju `primjer1` napravite poddirektorije `include\GL` i `lib`. U direktorij `include\GL` iskopirajte `glut.h` a u direktorij `lib` iskopirajte `glut32.lib`. Biblioteku `glut32.dll` iskopirajte direktno u direktorij `primjer1`. Ako ste ovo dobro napravili, trebali biste dobiti strukturu direktorija kako je prikazano u nastavku.

```
primjer1
  include
    GL
      glut.h
  lib
    glut32.lib
  glut32.dll
```

1.2.4 Program

Nakon što smo pripremili strukturu direktorija za projekt, dodajmo još i naš prvi program. Program ćemo nazvati `prvi.cpp`, i smjestit ćemo ga izravno u direktorij `primjer1`. Izvorni kod prikazan je na ispisu 1.1.

Da bismo preveli program, iskoristit ćemo prevodioc `gcc`.

```
gcc -Iinclude -Llib -o prvi.exe prvi.cpp -lfreeglut -lopengl32
```

Argumentom `-Iinclude` direktorij `include` dodajemo u popis direktorija u kojima `gcc` traži zaglavne datoteke. Ovo je potrebno jer smo tamo smjestili zaglavnu datoteku `GL/glut.h` koju koristimo u programu. Argumentom `-Llib` direktorij `lib` dodajemo u popis direktorija u kojima `gcc` traži biblioteke. Ovo je potrebno jer prilikom prevođenja koristimo datoteku `freeglut.lib` koja je tamo smještena. Biblioteka `opengl32.lib` nalazi se među datotekama koje su došle

zajedno s *MinGW*-om. Argument `-o prvi.exe` nalaže prevodiocu da izvršnu datoteku nazove `prvi.exe`. Konačno, argumenti koji započinju s `-l` uključuju pojedine biblioteke.

Uočimo da nam za prevođenje programa nisu potrebne prevedene biblioteke (konkretno, `freeglut.dll`[3]) već samo njihovi opisi (**.lib* datoteke). Međutim, da bismo program uspješno pokrenuli, prevedene biblioteke moraju biti ili u direktoriju iz kojeg pokrećemo program, ili u direktoriju koji na razini operacijskog sustava čuva sve dijeljene biblioteke. U našem konkretnom slučaju, biblioteku `freeglut.dll` smjestili smo u trenutni direktorij, pa će biti automatski korištena.

Nakon što smo program uspješno preveli, njegovim pokretanjem dobit ćemo prozor s tri točkice i jednim trokutom, kao što je prikazano na slici 1.1.

Ispis 1.1: Jednostavan primjer

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <windows.h>
4  #include <GL/glut.h>
5
6  void reshape(int width, int height);
7  void display();
8  void renderScene();
9
10 int main(int argc, char **argv) {
11     glutInit(&argc, argv);
12     glutInitDisplayMode(GLUT_DOUBLE);
13     glutInitWindowSize(200, 200);
14     glutInitWindowPosition(0, 0);
15     glutCreateWindow("Primjer 1");
16     glutDisplayFunc(display);
17     glutReshapeFunc(reshape);
18     glutMainLoop();
19 }
20
21 void display() {
22     glClearColor(1.0f, 0.0f, 0.0f, 1.0f);
23     glClear(GL_COLOR_BUFFER_BIT);
24     glLoadIdentity();
25     // crtanje scene:
26     renderScene();
27     glutSwapBuffers();
28 }
29
30 void reshape(int width, int height) {
31     glDisable(GL_DEPTH_TEST);
```

[3]Datoteka *DLL*, engl. *Dynamic Linked Library*, je poseban format datoteke koja se koristi na operacijskom sustavu Windows i koja omogućava pohranu prevedenog koda koji se može koristiti dinamički od strane više korisničkih programa.

```
32      glViewport(0, 0, (GLsizei)width, (GLsizei)height);
33      glMatrixMode(GL_PROJECTION);
34      glLoadIdentity();
35      glOrtho(0, width-1, height-1, 0, 0, 1);
36      glMatrixMode(GL_MODELVIEW);
37  }
38
39  void renderScene() {
40      glPointSize(1.0f);
41      glColor3f(0.0f, 1.0f, 1.0f);
42      glBegin(GL_POINTS);
43      glVertex2i(0, 0);
44      glVertex2i(2, 2);
45      glVertex2i(4, 4);
46      glEnd();
47      glBegin(GL_LINE_STRIP);
48      glVertex2i(50, 50);
49      glVertex2i(150, 150);
50      glVertex2i(50, 150);
51      glVertex2i(50, 50);
52      glEnd();
53  }
```

1.3 Anatomija GLUT aplikacije

Sada kada smo uspjeli prevesti i pokrenuti prvi OpenGL program, vrijeme je da se upoznamo s anatomijom same aplikacije, i pogledamo od čega se sve sastoji program prikazan na ispisu 1.1.

Krenimo redom. Linije 1-4 definiraju sve zaglavne datoteke koje su nam potrebne. Radimo li na operacijskom sustavu Linux, liniju 3 možemo preskočiti. Slijede linije 6-8 koje definiraju prototipove funkcija koje su kasnije korištene u kodu, i uskoro ćemo ih objasniti. Linije 10-19 čine metodu main() - mjesto od kuda započinje izvođenje samog programa. Linije 21-28 čine pomoćnu metodu display, linije 30-37 čine pomoćnu metodu reshape, a linije 39-53 čine pomoćnu metodu renderScene.

1.3.1 Metoda main

Metoda main predstavlja početnu točku programa. U toj metodi potrebno je inicijalizirati biblioteku *GLUT*, podesiti potrebne parametre, i zatim kontrolu izvođenja predati samoj biblioteci. Inicijalizacija biblioteke *GLUT* započinje u retku 11 pozivom: glutInit(&argc, argv);, gdje se metodi predaju dva argumenta: adresa varijable koja sadrži broj argumenata iz komandne linije te polje tih argumenata.

U retku 12 pozivom metode glutInitDisplayMode slijedi podešavanje načina iscrtavanja scene. Ako se kao argument preda vrijednost GLUT_SINGLE, sve metode za iscrtavanje scene će crtati direktno u grafičkom spremniku koji se istovremeno i prikazuje. U tom slučaju nakon što završimo s crtanjem scene, potrebno je pozvati metodu glFlush();. Ovaj način prikaza nikako nije prikladan kada se OpenGL koristi za prikazivanje animacija, jer će rezultirati pojavom titranja (engl. *flicker*). Razlog pojave titranja jest taj što se kod animacije stalno ponavlja petlja *obriši scenu – nacrtaj scenu*. Kako se sadržaj grafičkog spremnika u određenim trenutcima čita u svrhu prikaza na ekranu, često se dogodi da se pročita i prikaže do pola obrisana scena ili nedovršena scena.

Rješenje navedenog problema jest uporaba dva grafička spremnika: jednog čiji se sadržaj čita u svrhu prikaza na zaslonu te drugog u koji se crta sljedeća scena. Jednom kada je nova scena nacrtana, pozivom metode glutSwapBuffers(); mijenja se uloga spremnika: slika se na ekranu počinje prikazivati iz spremnika u kojem smo završili s crtanjem nove scene, a stari spremnik sada postaje spremnik u kojem započinjemo s crtanjem sljedeće scene. Kako se na ovaj način za potrebe prikaza na zaslonu uvijek čita spremnik koji sadrži gotovu sliku scene, neće se javljati titranje. Ovakav način rada podešavamo uporabom parametra GLUT_DOUBLE.

Metodi glutInitDisplayMode osim specificiranja jednostrukog ili dvostrukog spremnika možemo predati i zastavicu koja određuje kako se radi s bojom (koristi li se RGB ili paleta boja), koristi li se z-spremnik i sl. Primjerice, ako želimo dvostruki spremnik te paletu boja, naredbu ćemo pozvati na sljedeći način:

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_INDEX);
```

Ako se ne izjasnimo o načinu prikaza boja, GLUT_RGBA smatra se zadanim načinom, uz koji će se, za svaku točku, komponente crvene, zelene i plave boje pamtiti nezavisno, što će omogućiti uporabu izuzetno velikog broja različitih boja.

U retku 13 pozivom metode glutInitWindowSize(200, 200); podešavamo dimenzije prozora u kojem ćemo iscrtavati scenu. Prvi argument određuje širinu prikaza, a drugi visinu. Oba argumenta zadaju se kao broj slikovnih elemenata (engl. *pixel*).

U retku 14 pozivom metode glutInitWindowPosition(0, 0); određujemo poziciju gornjeg lijevog ugla prozora koji će prikazivati našu scenu. Prvi argument je vrijednost koordinate x a drugi vrijednost koordinate y, u slikovnim elementima.

U retku 15 pozivom metode glutCreateWindow("Primjer 1"); zahtjeva se stvaranje prozora u kojem će se iscrtavati scena. Kao argument se predaje naziv prozora – taj će tekst biti ispisan u naslovnoj traci prozora. Metoda glutCreateWindow kao rezultat vraća podatak tipa **int** – identifikator prozora koji možemo zapamtiti, i kasnije koristiti za uništavanje prozora pozivom metode glutDestroyWindow(windowID);.

Da bismo objasnili sljedeće tri naredbe, moramo se upoznati s načinom na koji GLUT korisniku dojavljuje što se sve događa s prikazom.

1.3.2 Događaji

Grafičke aplikacije najčešće su interaktivne; prate pomake miša korisnika, prate pritiske tipaka i shodno tome, mijenjaju prikazanu scenu. Pomak miša, pritisak te otpuštanje neke od tipaka miša, te pritisak i otpuštanje neke od tipaka na tipkovnici primjeri su onoga što jednostavno zovemo – *događaji* (engl. *events*). Međutim, osim navedenih, GLUT će korisniku dojavljivati još neke događaje. Primjerice, nakon što stvorimo novi prozor, potrebno je u njemu nacrtati scenu. Ako se u nekom trenutku iznad tog prozora otvori i potom zatvori drugi prozor, scenu će ponovno trebati nacrtati. Ako korisnik odluči promijeniti dimenzije prozora, scenu će ponovno trebati nacrtati, a možda i dodatno prilagoditi. Stoga je programski model koji GLUT nudi korisniku model temeljen na događajima: kada se god dogodi nešto što zahtjeva intervenciju korisnika, GLUT će to dojaviti generiranjem određenog događaja.

Da bi korisnik (programer) mogao reagirati na događaj, potrebno je obaviti registraciju – korisnik mora najaviti GLUT-u da želi da se u slučaju pojave događaja `X` izvede korisnikova metoda `obradiX`. Dakako, ovisno o vrsti događaja, metode će imati različit broj i vrstu argumenata. Nakon što je i ovaj korak obavljen, kontrolu izvođenja potrebno je predati biblioteki GLUT, koja će ući u beskonačnu petlju osluškivanja, generiranja događaja i pozivanja registriranih metoda koje će potom obraditi te događaje. Sve ovo događa se pozivom metode glutMainLoop();, što je prikazano u retku 18. Pseudokod ove metode mogli bismo prikazati na sljedeći način.

```
void glutMainLoop() {
    while(1) {
        // cekaj na promjenu
        // utvrdi o cemu se radi
        // pozovi registriranu metodu
    }
}
```

Iz ove metode više se nikada ne izlazi, tako da je prije njenog poziva potrebno završiti kompletnu inicijalizaciju programa, i registrirati sve željene metode. Upoznajmo se sada redom i s događajima koji nam stoje na raspolaganju.

Potreba za iscrtavanjem scene

Svaki puta kada se prikazana scena na neki način uništi (prekrivanjem nekim drugim prozorom) ili pak promjenom dimenzija prozora, ili kada je scenu potrebno prikazati po prvi puta, GLUT generira događaj *display*. Da bi se korisnik registrirao za taj događaj, treba napisati metodu čiji je prototip prikazan u nastavku.

```
void nazivMetode();
```

Primjer ovakve metode prikazan je u nastavku.

```
void display() {
  // obavi crtanje scene
}
```

Naziv same metode uopće nije bitan. Naime, jednom kada smo metodu napisali, prilikom inicijalizacije biblioteke GLUT potrebno je tu metodu registrirati kao metodu koja se poziva u svrhu iscrtavanja scene. To se radi pozivom metode: glutDisplayFunc(metoda); kojoj kao argument predajemo pokazivač na samu metodu. U ovoj knjizi koristit ćemo konvenciju koja će takvu metodu uvijek zvati `display`.

U primjeru prikazanom na ispisu 1.1, prototip metode display naveden je u retku 7, registracija te metode obavljena je u retku 16, a sama metoda definirana je u retcima 21 do 28.

Osim opisanog događaja *display* u primjeru 1.1. mogući su i razni drugi događaji, te su neki od njih opisani u nastavku.

Tipka je pritisnuta

Svaki puta kada korisnik pritisne neku od *običnih* tipki (slovo, znamenku i sl.), GLUT generira događaj *keyboard*. Funkcija koju korisnik može registrirati za obradu ovog događaja treba imati prototip kako slijedi.

```
void nazivMetode(unsigned char key, int x, int y);
```

Takvu funkciju registriramo pozivom metode glutKeyboardFunc(metoda); Primjer takve funkcije prikazan je u nastavku. Prilikom poziva te funkcije, GLUT će nam odmah dostaviti i x i y koordinate na kojima se je u trenutku pritiska tipke nalazio pokazivač miša.

```
void keyPressed(unsigned char key, int x, int y) {
  // obradi; primjerice, zapamti u nekom polju da je
  // tipka key pritisnuta.
}
```

Tipka je otpuštena

Svaki puta kada korisnik otpusti neku od *običnih* tipki (slovo, znamenku i sl.), GLUT generira događaj *keyboardUp*. Funkcija koju korisnik može registrirati za obradu ovog događaja treba imati prototip kako slijedi.

```
void nazivMetode(unsigned char key, int x, int y);
```

Takvu funkciju registriramo pozivom metode glutKeyboardUpFunc(metoda); Primjer takve funkcije prikazan je u nastavku. Prilikom poziva te funkcije, GLUT će nam odmah dostaviti x i y koordinate na kojima se u trenutku otpuštanja tipke nalazio pokazivač miša.

```
void keyReleased(unsigned char key, int x, int y) {
  // obradi; primjerice, zapamti u nekom polju da tipka
  // key nije pritisnuta.
}
```

Posebna tipka je pritisnuta

Svaki puta kada korisnik pritisne neku od *posebnih* tipki (kursorska tipka gore, funkcijska tipka i sl.), GLUT generira događaj *special*. Funkcija koju korisnik može registrirati za obradu ovog događaja treba imati prototip kako slijedi.

```
void nazivMetode(int key, int x, int y);
```

Takvu funkciju registriramo pozivom metode glutSpecialFunc(metoda); Primjer takve funkcije prikazan je u nastavku. Prilikom poziva te funkcije, GLUT će nam odmah dostaviti i x i y koordinate na kojima se u trenutku pritiska tipke nalazio pokazivač miša. Prvi argument funkcije sadrži kod pritisnute tipke. Valjani kodovi definirani su konstantama poput GLUT_KEY_F1, GLUT_KEY_F2 za funkcijske tipke, GLUT_KEY_LEFT, GLUT_KEY_RIGHT za kursorske tipke i slično.

```
void keySpecial(int key, int x, int y) {
  // obradi; primjerice, zapamti u nekom polju da je
  // posebna tipka key pritisnuta.
}
```

Posebna tipka je otpuštena

Svaki puta kada korisnik otpusti neku od *posebnih* tipki (kursorska tipka gore, funkcijska tipka i sl.), GLUT generira događaj *specialUp*. Funkcija koju korisnik može registrirati za obradu ovog događaja treba imati prototip kako slijedi.

```
void nazivMetode(int key, int x, int y);
```

Takvu funkciju registriramo pozivom metode glutSpecialUpFunc(metoda); Primjer takve funkcije prikazan je u nastavku. Prilikom poziva te funkcije, GLUT će nam odmah dostaviti x i y koordinate na kojima se u trenutku otpuštanja tipke nalazio pokazivač miša.

```
void keySpecialUp(int key, int x, int y) {
  // obradi; primjerice, zapamti u nekom polju da
  // posebna tipka key nije pritisnuta.
}
```

Promijenjena je veličina prozora

Svaki puta kada se promijeni veličina prozora te pri prvom otvaranju prozora, GLUT generira događaj *reshape*. Funkcija koju korisnik može registrirati za obradu ovog događaja treba imati prototip kako slijedi.

```
void nazivMetode(int width, int height);
```

Takvu funkciju registriramo pozivom metode glutReshapeFunc(metoda); Primjer takve funkcije prikazan je u nastavku.

```
void reshape(int width, int height) {
  // obradi; promijeni potrebne parametre scene
}
```

U primjeru prikazanom na ispisu 1.1, prototip metode reshape naveden je u retku 6, registracija te metode obavljena je u retku 17, a sama metoda definirana je u retcima 30 do 37.

Tipka miša je pritisnuta ili otpuštena

Svaki puta kada korisnik pritisne ili otpusti neku od tipki miša, GLUT generira događaj *mouse*. Funkcija koju korisnik može registrirati za obradu ovog događaja treba imati prototip kako slijedi.

```
void nazivMetode(int button, int state, int x, int y);
```

Takvu funkciju registriramo pozivom metode glutMouseFunc(metoda); Primjer takve funkcije prikazan je u nastavku. Prilikom poziva te funkcije, GLUT će nam dostaviti indikator tipke (argument button) koja je u pitanju (bit će jedna od vrijednosti GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON), što se dogodilo (argument state, vrijednost će biti GLUT_UP ako je tipka otpuštena, GLUT_DOWN ako je tipka pritisnuta) te x i y koordinate na kojima se je u trenutku pritiska/otpuštanja tipke nalazio pokazivač miša.

```
void mousePressedOrReleased(int button, int state, int x, int y) {
  // obradi; primjerice, zapamti na kojoj je lokaciji bio mis
  // kada se ovo dogodilo.
}
```

Korisnik je pomaknuo pokazivač miša

Svaki puta kada korisnik pomakne pokazivač miša, GLUT generira događaj *motion* (ako je barem jedna od tipki miša u tom trenutku također pritisnuta) odnosno *passive-motion* (ako niti jedna tipka miša nije pritisnuta). Funkcije koju korisnik može registrirati za obradu ovih događaja trebaju imati prototip kako slijedi.

```
void nazivMetode(int x, int y);
```

Metodama glutMotionFunc(metoda); odnosno glutPassiveMotionFunc(metoda); registriramo takve funkcije. Primjer takve funkcije prikazan je u nastavku. Prilikom poziva te funkcije, GLUT će nam dostaviti x i y koordinate na kojima se u trenutku pomicanja nalazio pokazivač miša.

```
void mouseMoved(int x, int y) {
  // za registraciju preko glutPassiveMotionFunc
  // obradi; primjerice, nacrtaj segment linije do ove pozicije.
}
void mouseDragged(int x, int y) {
  // za registraciju preko glutMotionFunc
  // obradi; primjerice, pomakni ikonicu na novu poziciju.
}
```

Pozivanje funkcije za registraciju bilo kojeg od događaja s argumentom NULL odregistrirat će prethodno registriranu funkciju.

1.3.3 Iscrtavanje scene

Funkciju display() u ispisu 1.1 registrirali smo kod GLUT-a kao funkciju koju treba pozvati svaki puta kada se pojavi potreba za iscrtavanjem scene u prozoru. Implementacija te funkcije prikazana je u retcima 21 do 28. Metoda počinje pozivom kojim se postavlja vrijednost boje koja će biti korištena za brisanje površine platna na kojem radimo iscrtavanje (takozvana pozadinska boja). Radi se o pozivu u retku 22: glClearColor(1.0f, 0.0f, 0.0f, 1.0f);. Prva tri argumenta su redom RGB vrijednosti crvene boje, zelene boje i plave boje. Vrijednost 0 kod pojedine komponente znači da ta komponenta nije uključena u stvaranje konačne boje dok vrijednost 1 znači da je uključena maksimalnim intenzitetom. Četvrta komponenta predstavlja α vrijednost boje, odnosno vrijednost koja definira prozirnost te boje. Vrijednost $\alpha = 0$ označava da je boja potpuno prozirna (engl. *transparent*) dok vrijednost $\alpha = 1$ označava da je boja potpuno neprozirna (engl. *opaque*). Prema postavljenim argumentima u našem primjeru, boja pozadine bit će crvena.

U retku 23 slijedi poziv metode glClear(GL_COLOR_BUFFER_BIT); kojom se obavlja brisanje, tj. popunjavanje čitave površine prethodno definiranom bojom za brisanje.

Redak 24 sadrži poziv kojim se poništavaju sve prethodno definirane transformacije nad aktivnom transformacijskom matricom, a to je u konkretnom primjeru nad matricom MODEL_VIEW, odnosno matricom koja regulira operacije nad kamerom kojom se "snima" scena. To se obavlja tako što se kao matrica koja djeluje nad modelom postavlja matrica identiteta, odnosno jedinična matrica: glLoadIdentity();. Ovo na prvi pogled djeluje možda malo zbunjujuće, no lagano je za objasniti ako prihvatimo da OpenGL sve transformacije nad scenom radi kroz nekoliko matrica. Također, važno je zapamtiti da se OpenGL ponaša kao jedan veliki automat - svaka naredba koju zadamo mijenja stanje tog automata te ostaje djelovati sve dok to nekom drugom naredbom ne poništimo. Inicijalno, OpenGL se nalazi u stanju u kojem sve operacije djeluju nad matricom modela. Stoga, budući da još ništa nismo mijenjali, a to ćemo kasnije raditi naredbom glMatrixMode(koja_matrica);, naredba glLoadIdentity(); djelovat će upravo nad matri-

com modela i resetirat će je na jediničnu matricu. U metodi reshape() pogledat ćemo primjer u kojem se najprije prebacujemo u stanje u kojem manipuliramo projekcijskom matricom, nakon čega se vraćamo u način koji manipulira matricom modela.

Napomenimo još i da OpenGL uvijek radi s 3D koordinatama točaka. Zapravo, interno radi s 3D-koordinatama kojima je pridružena još i homogena koordinata; više o ovome bit će riječi u kasnijim poglavljima. Za sada je dovoljno razmišljati o tim točkama kao 3D točkama. U ovim početnim primjerima ograničit ćemo se na 2D prikaz, na način da OpenGL-u naložimo uporabu projekcije svih točaka na ravninu x-y. Dodatno, pozivat ćemo funkcije koje će točke ili zadavati kao 2D objekte (pa će time z-koordinata implicitno biti postavljena na 0), ili ćemo u slučaju uporabe funkcije koje primaju 3D koordinate treću koordinatu eksplicitno postaviti na 0.

Jednom kada smo sve pripremili za crtanje same scene (čitav spremnik popunili smo pozadinskom bojom, poništili smo sve eventualno zaostale transformacije od prethodnog crtanja), u retku 26 pozivamo našu metodu kojoj je cilj u aktivnom grafičkom spremniku nacrtati samu scenu. Sjetimo se da smo prilikom inicijalizacije OpenGL-a podesili uporabu dvostrukog spremnika - sve što smo do sada radili, radili smo nad spremnikom koji se trenutno ne prikazuje, i služi za pripremu nove slike. Metoda renderScene(); scenu će nacrtati upravo u tom spremniku.

Nakon što je slika nacrtana, u retku 27 pozivom metode glutSwapBuffers(); tražimo od OpenGL-a da na zaslon počne iscrtavati sliku koju smo upravo pripremili, te da spremnik koji je do tada čuvao staru sliku počne koristiti za iscrtavanje nove.

1.3.4 Promjena veličine prozora

Funkciju reshape() u ispisu 1.1 (retci 30-37) prilikom inicijalizacije GLUT-a registrirali smo kao funkciju koju GLUT treba pozvati svaki puta kada se promjeni veličina prozora. Argumenti te funkcije su nova širina (width) te nova visina (height) prozora. Kako u ovom primjeru ne želimo raditi s 3D scenom, prvi korak je isključivanje uporabe *z-spremnika* – strukture koja se koristi kako bi u 3D sceni otkrila između više točaka koje leže na pravcu gledanja, koju točku zapravo vidimo (odnosno koja točka skriva ostale). To se obavlja pozivom metode glDisable(GL_DEPTH_TEST); u retku 31. Nakon toga, u retku 32 pozivom funkcije glViewport(0, 0, (GLsizei)width, (GLsizei)height); definiramo koji dio prozora prikazuje samu scenu. Argumenti su redom x, y, width i height, što znači da se scena iscrtava počev od slikovnog elementa na koordinatama (0,0) i proteže width slikovnih elemenata u širinu i height slikovnih elemenata u visinu – dakle, preko čitavog prozora.

U retku 33 pozivom metode glMatrixMode(GL_PROJECTION); govorimo OpenGL-u da će se rad s matricama koji slijedi odnositi na podešavanje matrice projekcije.

Više o projekciji bit će objašnjeno u kasnijim poglavljima ove knjige. Slijedi redak 34 s naredbom glLoadIdentity(); koja kao projekcijsku matricu učitava matricu identiteta – matricu koja ne obavlja ništa, čime efektivno poništava sve prethodno definirane transformacije vezane uz projekciju točaka. U retku 35 potom definiramo novu projekcijsku matricu, koja obavlja ortogonalnu projekciju; naredba je glOrtho(0, width−1, height−1, 0, 0, 1);. Semantika ove naredbe bit će detaljno objašnjena nakon što se upoznamo s pojmom projekcije i vrstama koje se koriste. Za sada, bit će dovoljno reći da ovako definirana projekcija prikazuje sve slikovne elemente koji po x-u idu od 0 do width−1, te koji po y-u idu od 0 do height−1 a ishodište im je u gornjem lijevom kutu. Želimo li ishodište u donjem lijvom kutu, to možemo načiniti jednostavnom promjenom glOrtho(0, width−1, 0, height−1, 0, 1);. Svaka 3D točka oblika (x,y,z) kod koje su x i y koordinate unutar propisanog raspona naprosto se preslika u 2D prostor u točku (x,y).

Nakon što smo podesili parametre koji definiraju što se točno prikazuje, OpenGL prebacujemo u stanje u kojem daljnje zadavanje transformacija ponovno djeluje nad matricom modela, što radimo pozivom glMatrixMode(GL_MODELVIEW); u retku 36. Važno je ovaj korak ne preskočiti. Naime, sjetimo se da je OpenGL stroj stanja. Kako se funkcija reshape tipično zove neposredno prije metode display (kada dođe do promjene veličine, ili prvi puta kada stvorimo novi prozor), preskočimo li ovaj korak, poziv metode glLoadIdentity(); u retku 24 ponovno bi resetirao projekcijsku matricu, a ne transformacije nad kamerom – kako smo to prethodno objasnili.

1.3.5 Crtanje točaka i linija

I konačno, metoda renderScene() (retci 39-53) radi konkretno crtanje. Prisjetimo se, ta je metoda pozvana iz metode display nakon što je grafički spremnik bio popunjen s pozadinskom bojom. Pa krenimo redom. U retku 40 pozivom metode glPointSize(1.0f); podešava se veličina *točke* koju će OpenGL koristiti prilikom crtanja. Primjerice, ako veličinu točke postavimo na 1, naredba koja točku crta na mjestu (10,10) doista će i upaliti samo jedan slikovni element. Međutim, stavimo li primjerice kao veličinu točke 10, naredba koja točku crta na mjestu (10,10) nacrtat će popunjen krug čije je središte na zadanom mjestu.

U retku 41 naredbom glColor3f(0.0f, 1.0f, 1.0f); definiramo boju koja će se koristiti za crtanje točaka. Argumenti su vrijednosti R, G i B komponenti boje. Nakon ovoga slijede još dva bloka naredbi koji započinju s glBegin(vrstaPrimitiva); i završavaju naredbom glEnd().

Prvi primjer koji se proteže od retka 42 do retka 46 koristi primitiv GL_POINTS. Ovaj primitiv crta slijed točaka koje mu se zadaju – jednu po jednu točno kako su zadane. U ovom primjeru, palimo slikovne elemente smještene na koordinatama

(0,0), (2,2) i (4,4). Svaka točka zadaje se naredbom glVertex2i(x, y);. U toj naredbi broj 2 označava da se radi o 2D koordinati, a slovo *i* označava da su koordinate cijeli brojevi.

Sljedeći primjer koji se proteže od retka 47 do retka 52 demonstrira uporabu primitiva GL_LINE_STRIP. Ovaj primitiv niz točaka koje navedemo spaja linijama, i to onim redosljedom kojim su zadane točke. Tako ako nakon poziva funkcije glBegin(GL_LINE_STRIP); zadamo četiri točke: t_1, t_2, t_3 i t_4, naredba će povući linije t_1-t_2, t_2-t_3 i t_3-t_4. S obzirom da se u primjeru crta trokut, zadani su redom prvi vrh trokuta, drugi vrh trokuta, treći vrh trokuta i konačno još jednom je ponovljen prvi vrh trokuta (čime je povučena linija od trećeg vrha natrag do prvog vrha i tako je zatvoren trokut).

1.4 OpenGL primitivi za crtanje

U prethodnom poglavlju spomenuli smo već da se crtanje u OpenGL-u radi uporabom bloka naredbi koji započinje naredbom glBegin(vrstaPrimitiva); i završava naredbom glEnd(). Između ta dva graničnika, zadaje se jedna ili više točaka, koje se potom obrađuju, ovisno o odabranom primitivu vrstaPrimitiva. U nastavku ćemo pogledati što nam sve stoji na raspolaganju.

Tablica 1.1: OpenGL primitivi za crtanje

Primitiv	**Opis**
GL_POINTS	Svaka zadana točka crta se zasebno. Zadamo li n točaka, toliko će ih biti i nacrtano.
GL_LINES	Svake dvije točke tumače se kao jedan segment linije, i tako se crtaju. Zadamo li n točaka, nacrtat će se $n/2$ linija: t_1-t_2, t_3-t_4, ..., t_{n-3}-t_{n-2}, t_{n-1}-t_n. Ako se zada neparan broj točaka, posljednja točka bit će zanemarena.
GL_LINE_STRIP	Zadane točke tumače se kao vrhovi poligonalne linije koju treba nacrtati. Zadamo li n točaka, nacrtat će se $n-1$ linija: t_1-t_2, t_2-t_3, ..., t_{n-2}-t_{n-1}, t_{n-1}-t_n.
GL_LINE_LOOP	Zadane točke tumače se kao vrhovi zatvorene poligonalne linije koju treba nacrtati. Zadamo li n točaka, nacrtat će se n linija: t_1-t_2, t_2-t_3, ..., t_{n-2}-t_{n-1}, t_{n-1}-t_n, t_n-t_1.

Nastavlja se na sljedećoj stranici

Tablica 1.1 – nastavak s prethodne stranice

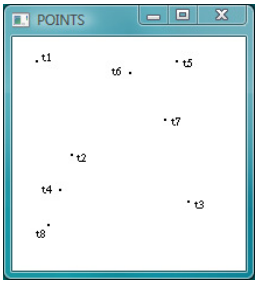
Primitiv	**Opis**
GL_TRIANGLES	Zadane točke grupiraju se u grupe po tri, i tumače kao vrhovi trokuta koje treba nacrtati. Zadamo li n točaka, nacrtat će se $n/3$ trokuta: trokut t_1-t_2-t_3, trokut t_4-t_5-t_6, itd. Trokuti se popunjavaju aktivnom bojom.
GL_TRIANGLE_STRIP	Ovaj primitiv također služi za crtanje trokuta, ali uz pretpostavku da susjedni trokuti dijele jednu zajedničku stranicu, pa se time štedi na broju točaka koje treba poslati grafičkoj kartici. Zadane točke tumače se kao vrhovi trokuta. Zadamo li n točaka, nacrtat će se $n-2$ trokuta. Za neparni vrh k crta se trokut t_k-t_{k+1}-t_{k+2}, dok se za parni vrh k crta trokut t_{k+1}-t_k-t_{k+2}. Trokuti se popunjavaju aktivnom bojom.
GL_TRIANGLE_FAN	Ovaj primitiv također služi za crtanje trokuta, ali uz pretpostavku da svi trokuti dijele jedan zajednički vrh, pa se time štedi na broju točaka koje treba poslati grafičkoj kartici. Zadane točke tumače se kao vrhovi trokuta. Zadamo li n točaka, nacrtat će se $n-2$ trokuta. Redom se crtaju trokuti t_1-t_2-t_3, t_1-t_3-t_4, t_1-t_4-t_5, itd. Trokuti se popunjavaju aktivnom bojom.
GL_QUADS	Zadane točke grupiraju se u grupe po četiri, i tumače kao vrhovi četverokuta koje treba nacrtati. Zadamo li n točaka, nacrtat će se $n/4$ četverokuta: četverokut t_1-t_2-t_3-t_4, četverokut t_5-t_6-t_7-t_8, itd. Četverokuti se popunjavaju aktivnom bojom.
GL_QUAD_STRIP	Zadane točke tumače kao vrhovi povezanih četverokuta koje treba nacrtati. Zadamo li n točaka, nacrtat će se $n/2-1$ četverokuta: četverokut t_1-t_2-t_4-t_3, četverokut t_3-t_4-t_6-t_5, itd. Četverokuti se popunjavaju aktivnom bojom.

Nastavlja se na sljedećoj stranici

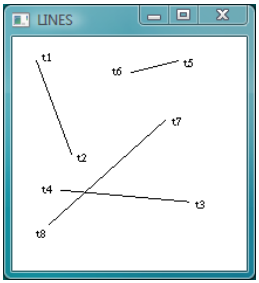
Tablica 1.1 – nastavak s prethodne stranice

Primitiv	**Opis**
GL_POLYGON	Crta se jedan jednostavan konveksan poligon određen zadanim točkama. Ovisno o stanju postavljenom naredbom glPolygonMode, nacrtat će se ili vrhovi poligona ili obrub poligona ili će se površina poligona obojati aktivnom bojom.

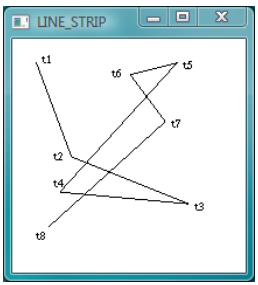
Primjeri rada ovih primitiva prikazani su na slikama 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10 i 1.11.



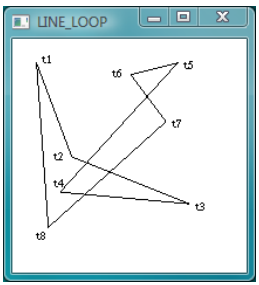
Slika 1.2: primitiv GL_POINT



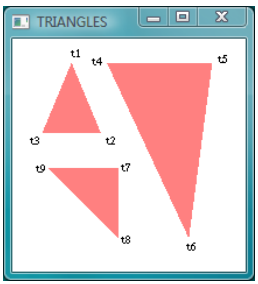
Slika 1.3: primitiv GL_LINES



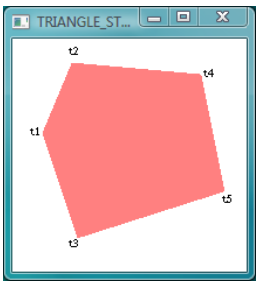
Slika 1.4: primitiv GL_LINE_STRIP



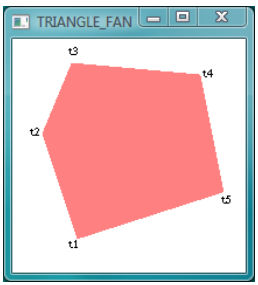
Slika 1.5: primitiv GL_LINE_LOOP



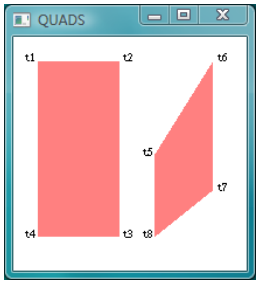
Slika 1.6: primitiv GL_TRIANGLES



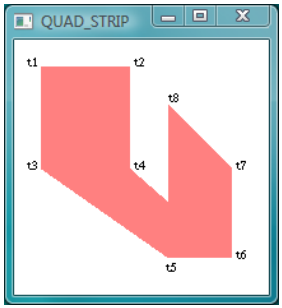
Slika 1.7: primitiv GL_TRIANGLE_STRIP



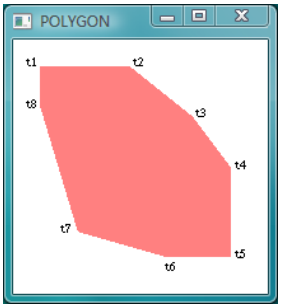
Slika 1.8: primitiv GL_TRIANGLE_FAN



Slika 1.9: primitiv GL_QUADS



Slika 1.10: primitiv GL_QUAD_STRIP



Slika 1.11: primitiv GL_POLYGON

1.5 Animacija i OpenGL

U prvom dijelu ovog poglavlja već smo spomenuli da *OpenGL* ne nudi direktno podršku za rad s prozorima; u tu svrhu koristili smo biblioteku *GLUT*. Unutar iste biblioteke nalazi se i osnovna podrška za izradu animacija. Osnovna ideja animacije jest prikazati promjenu kroz vrijeme. A što se može mijenjati? Tipično, govorimo o:

- *svojstvima objekata u sceni* – pri čemu se može mijenjati oblik, tekstura, položaj i orijentacija u prostoru, te čak i sam broj objekata,
- *položaju promatrača* – što nam omogućava da kameru pomičemo kroz prostor (ali ne mijenjamo točku u koju je kamera usmjerena) te
- *točki pogleda* – što pretpostavlja da je položaj kamere fiksiran, ali se mijenja točka u koju kamera gleda.

Dakako, u okviru animacije mogući su i složeniji slučajevi. Primjerice, kamera se može pomicati kroz scenu pri čemu se istovremeno može mijenjati i točka u koju je kamera usmjerena. Ovdje ćemo se prvenstveno fokusirati na općenitu podršku animaciji koja dozvoljava sve navedene slučajeve. Međutim, kako još nismo govorili o 3D scenama, primjeri će ilustrirati animaciju koja prikazuje promjenu svojstava objekata.

1.5.1 Animacija temeljena na metodi idle

Prvi primjer prikazan je u ispisu 1.2.

Ispis 1.2: Primjer animacije

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <GL/glut.h>

void reshape(int width, int height);
void display();
void renderScene();
void idle();
void drawSquare();
int kut = 0;

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(600, 300);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Primjer animacije");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(idle);
    glutMainLoop();
}

void idle() {
    kut++;
    if(kut>=360) kut = 0;
    glutPostRedisplay();
}

void display() {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    renderScene();
    glutSwapBuffers();
}

void reshape(int width, int height) {
    glDisable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, width-1, 0, height-1, 0, 1);
    glViewport(0, 0, (GLsizei)width, (GLsizei)height);
    glMatrixMode(GL_MODELVIEW);
}

void drawSquare() {
    glBegin(GL_QUADS);
    glVertex2f(  0.0f,   0.0f);
```

```
    glVertex2f(100.0f,   0.0f);
    glVertex2f(100.0f, 100.0f);
    glVertex2f(  0.0f, 100.0f);
    glEnd();
}

void renderScene() {
    glPointSize(1);
    glColor3f(1.0f, 0.0f, 0.3f);

    glPushMatrix();
    glTranslatef( 150.0f,  160.0f, 0.0f);
    glScalef(1.5f, 1.5f, 1.0f);
    glRotatef((float)kut, 0.0f, 0.0f, 1.0f);
    glTranslatef(-50.0f, -50.0f, 0.0f);
    drawSquare();
    glPopMatrix();

    glPushMatrix();
    glTranslatef( 400.0f,  160.0f, 0.0f);
    glScalef(1.5f, 1.5f, 1.0f);
    glRotatef(-(float)kut, 0.0f, 0.0f, 1.0f);
    glTranslatef(-50.0f, -50.0f, 0.0f);
    drawSquare();
    glPopMatrix();
}
```

Ovaj program prikazuje dva kvadrata koja se rotiraju; prvi u smjeru kazaljke na satu, drugi u suprotnom smjeru. U prikazanom primjeru iskorištena je *GLUT*-ova funkcija glutIdleFunc(...) koja nam omogućava registraciju funkcije koju treba pozvati svaki puta kada *GLUT* više nema nikakvog posla (obrađeni su svi događaji). Registracija je obavljena u metodi main(...) gdje se traži pozivanje naše funkcije **void** idle() {...}. U tijelu funkcije idle povećavamo globalnu varijablu kut (za 1 stupanj) te pozivom metode glutPostRedisplay(); dojavljujemo *GLUT*-u da bi sadržaj prozora trebalo ponovno nacrtati. Uočimo da poziv glutPostRedisplay(); ne pokreće odmah iscrtavanje prozora – interno, metoda samo postavlja zastavicu da bi prozor ponovno trebalo nacrtati. Višestruki pozivi te metode također neće uzrokovati višestruka iscrtavanja. Fizičko iscrtavanje dogodit će se tek nakon što se kontrola izvođenja vrati metodi glutMainLoop(), koja će ustanoviti da postoji potreba za ponovnim iscrtavanjem prozora, pa će pozvati u tu svrhu registriranu funkciju (u našem primjeru to će biti metoda display() koju smo prethodno registrirali u metodi main pozivom funkcije glutDisplayFunc).

Posljedica ovakvog pristupa animaciji jest da će se animacija odvijati maksimalnom brzinom. Naime, čim se završi obrada svih događaja, *GLUT* će pozvati našu funkciju idle() koja će tražiti ponovno iscrtavanje prozora. Povratkom kontrole metodi glutMainLoop() pozvat će se funkcija za crtanje, i kako nakon toga više neće biti nikakvog posla, pozvat će se naša registrirana metoda idle(), koja

će opet zatražiti novo crtanje. Brzina animacije bit će, dakle, određena brzinom centralnog procesora računala te brzinom grafičke kartice. Stoga će ista aplikacija na različito brzim računalima raditi različitom brzinom. Mehanizam kontrole brzine izvođenja bit će objašnjen u nastavku.

Pogledajmo još kako je izvedeno samo crtanje. Najprije, definirana je metoda drawSquare() koja crta jedan kvadrat, i to uvijek na istoj poziciji: lijevi donji ugao je ishodište – točka (0,0) – a desni gornji ugao je točka (100,100). Kvadrat crtamo uporabom primitiva GL_QUADS. Metoda renderScene() zadužena je za crtanje scene, nakon što je metoda display podesila početne postavke. Prve dvije naredbe metode renderScene() podešavaju veličinu točke te boju kojom će se popunjavati likovi. Potom slijede dva bloka naredbi omeđena pozivima glPushMatrix(); i glPopMatrix(); koji crtaju rotirane kvadate.

Objasnimo najprije kako se postiže efekti rotacije, da bi nam bilo jasno čemu taj par naredbi. Metoda drawSquare() uvijek crta kvadrat stranice 100 čiji je jedan ugao fiksiran u ishodištu. S druge pak strane, *OpenGL* ima na raspolaganju funkcije koje koje obavljaju rotaciju, ali uvijek oko ishodišta. Razmislimo li malo, doći ćemo do zaključka da nam ovo baš i ne odgovara. Naime, mi kvadrat želimo rotirati oko njegovog središta, a ne oko vrha koji je fiksiran u ishodište. Stoga *OpenGL*-u trebamo reći da napravi nekoliko koraka, kako slijedi.

1. Kvadrat želimo pomaknuti za -50 po osi x te za -50 po osi y. Ovime će se kvadrat protezati od (-50,-50) pa do (50,50), a njegov centar (sjecište dijagonala) bit će smješten u ishodište.

2. Sada možemo tako pomaknuti kvadrat zarotirati za željeni broj stupnjeva oko osi z (ta je os okomita na ravninu x-y u kojoj se radi crtanje.

3. Nastali kvadrat želimo uvećati za 50% (tj. duljinu stranica pomnožiti faktorom 1.5).

4. Konačno, prikladno zarotirani kvadrat (čije je središte načinjenom rotacijom i dalje ostalo u ishodištu) pomaknut ćemo do mjesta gdje ga stvarno želimo nacrtati.

Za sve ove operacije *OpenGL* nam nudi odgovarajuće funkcije. Pomak (translaciju) ćemo obaviti uporabom funkcije glTranslatef a rotaciju uporabom funkcije glRotatef. Međutim, ove funkcije ne primaju kao argument točku i ne vraćaju transformiranu točku. Prisjetimo se – *OpenGL* je stroj stanja, koji za sve transformacije koristi matrični račun. Pozivanjem navedenih metoda malo po malo modificira se matrica koja će u konačnici obaviti čitav slijed transformacija u samo jednom koraku. Detaljnije o tome kako se ovo radi bit će objašnjeno u jednom od sljedećih poglavlja. Pogledajmo za sada samo jedan jednostavan primjer. U metodi display matricu smo resetirali na jediničnu matricu. Ako potom redom

pozovemo tri metode m1(), m2() i m3() koje obavljaju neku transformaciju, svaka metoda će trenutnu matricu pomnožiti matricom koja obavlja traženu transformaciju (neka je transformacija koju radi i-ta metoda definirana matricom $\mathbf{M_i}$), i rezultat postaviti kao trenutnu matricu. Neka je trenutna matrica označena s $\mathbf{M}$. Možemo pisati:

$$\begin{aligned}
\text{glLoadIdentity}();: &\quad \mathbf{M} \leftarrow \mathbf{I} \\
\text{m1}();: &\quad \mathbf{M} \leftarrow \mathbf{M} \cdot \mathbf{M_1} = \mathbf{I} \cdot \mathbf{M_1} \\
\text{m2}();: &\quad \mathbf{M} \leftarrow \mathbf{M} \cdot \mathbf{M_2} = \mathbf{I} \cdot \mathbf{M_1} \cdot \mathbf{M_2} \\
\text{m3}();: &\quad \mathbf{M} \leftarrow \mathbf{M} \cdot \mathbf{M_3} = \mathbf{I} \cdot \mathbf{M_1} \cdot \mathbf{M_2} \cdot \mathbf{M_3}
\end{aligned}$$

U konačnici, kada će krenuti u obavljanje transformacija, *OpenGL* će trenutnom matricom pomnožiti svaku od točaka $\vec{t}$ koje treba transformirati kako bi dobio transformirane točke $\vec{t'}$, pa možemo pisati:

$$\vec{t'} = \mathbf{M} \cdot \vec{t} = \mathbf{I} \cdot \mathbf{M_1} \cdot \mathbf{M_2} \cdot \mathbf{M_3} \cdot \vec{t}.$$

Ovaj mali izvod bio nam je potreban kako bismo utvrdili kojim redosljedom treba pozivati metode *OpenGL*-a. Prisjetimo se svojstava matričnog množenja: ono nije komutativno ali jest asocijativno. Stoga gornju transformaciju možemo zapisati ovako:

$$\vec{t'} = \mathbf{M} \cdot \vec{t} = \mathbf{I} \cdot (\mathbf{M_1} \cdot (\mathbf{M_2} \cdot (\mathbf{M_3} \cdot \vec{t}))).$$

Sada je jasno vidljivo: zadnja definirana transformacija na točku djeluje prva! Potom se nad tako transformiranom točkom primjenjuje predzadnja transformacija, itd. Sjetimo se što smo u našem primjeru htjeli postići: kvadrat smo najprije htjeli pomaknuti po osima x i y za -50 – sada je jasno da će ovo morati posljednji poziv, a ne prvi. Potom smo kvadrat htjeli zarotirati za zadani kut, skalirati s faktorom 1.5 po x- i y-osima, i konačno tako zarotirani i uvećani kvadrat pomaknuti na konačnu poziciju. Imajući to u vidu, korektan redosljed naredbi prikazan je u sljedećem isječku.

```
glTranslatef( 150.0f,  160.0f, 0.0f);
glScalef(1.5f, 1.5f, 1.0f);
glRotatef((float)kut, 0.0f, 0.0f, 1.0f);
glTranslatef(-50.0f, -50.0f, 0.0f);
drawSquare();
```

Crtanje drugog kvadrata, međutim, zahtjeva drugačije transformacije; tamo rotiramo u smjeru suprotnom od smjera kazaljke na satu, a i konačna je pozicija drugačija. Stoga nam treba mehanizam kojim bismo pojedinim transformacijama mogli ograničiti doseg. Upravo u tu svrhu, *OpenGL* ima na raspolaganju matrični stog. Evo koja je ideja. Nakon što sve podesimo u metodi display(), kopiju

trenutne matrice gurnut ćemo na stog (čime ćemo je sačuvati), promijenit ćemo što treba, nacrtati prvi kvadrat, i kada smo gotovi, sa stoga ćemo izvaditi očuvanu vrijednost i postaviti je kao trenutnu – efektivno vraćajući stanje na ono koje je bilo prije transformacija prvog kvadrata. Priču ponavljamo i za drugi kvadrat: kopiju trenutne matrice pohranjujemo na stog, radimo crtanje i kada smo gotovi, sa stoga restauriramo pohranjenu vrijednost.

1.5.2 Animacija temeljena na uporabi vremenskog sklopa

U prethodnom potpoglavlju uočili smo loše svojstvo uporabe metode idle – brzina animacije ovisna je o brzini sklopovlja. Radimo li računalnu igru, ovo nam ponašanje nikako ne odgovara. Naime, ne želimo da se na bržem računalu neprijateljske jedinice kreću brže! Kako bismo riješili taj problem, ažuriranje stanja sustava želimo raditi u dobro definiranim vremenskim trenutcima. U tu svrhu, *GLUT* nam na raspolaganje stavlja mehanizam kontrole vremena uporabom *virtualnog vremenskog sklopa* (engl. *timer*); *virtualno* označava da ovi sklopovi fizički ne postoje u računalu već ih emulira operacijski sustav. Umjesto da koristimo poziv glutIdleFunc(idle);, prilikom inicijalizacije programa registrirat ćemo našu funkciju koju će *GLUT* pozvati nakon što istekne određeni vremenski period. Zadatak te funkcije bit će isti kao i zadatak funkcije idle – ažurirati podatke o stanju sustava (konkretno, kut rotacije), zatražiti ponovno iscrtavanje i podesiti vremenski sklop tako da opet pozove tu funkciju.

Metoda koju nam *GLUT* nudi za rad s vremenskim sklopovima zove se glutTimerFunc, i prima tri parametra: vrijeme u milisekundama koje govori nakon koliko vremena treba pozvati funkciju; drugi parametar je pokazivač na funkciju koju treba pozvati; konačno, treći argument je broj koji će se predati registriranoj funkciji u trenutku poziva – u ovom primjeru, to nećemo koristiti. *Timer* koji smo dobili na ovaj način poznat je pod nazivom engl. *one-shot timer*, odnosno vremenski sklop koji okida samo jednom. *GLUT* nam ne nudi mogućnost stvaranja vremenskog sklopa koji periodički poziva registriranu funkciju. Stoga je zadatak registrirane funkcije (u našem primjeru metode animate(...)) da osim izmjene stanja sustava ponovno zatraži pozivanje uporabom novog vremenskog sklopa. Čitav kod prikazan je na ispisu 1.3.

Funkcija koju registriramo za pozivanje kada vremenski sklop "okine" mora primati jedan argument: cijeli broj (tipa **int**) kojim će primiti vrijednost koja je bila zadana prilikom stvaranja vremenskog sklopa.

Ispis 1.3: Primjer animacije

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <GL/glut.h>
```

```
void reshape(int width, int height);
void display();
void renderScene();
void animate(int value);
void drawSquare();
int kut = 0;

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE);
    glutInitWindowSize(600, 300);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Primjer animacije");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutTimerFunc(20, animate, 0);
    glutMainLoop();
}

void animate(int value) {
    kut++;
    if(kut>=360) kut = 0;
    glutPostRedisplay();
    glutTimerFunc(20, animate, 0);
}

void display() {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    renderScene();
    glutSwapBuffers();
}

void reshape(int width, int height) {
    glDisable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, width-1, 0, height-1, 0, 1);
    glViewport(0, 0, (GLsizei)width, (GLsizei)height);
    glMatrixMode(GL_MODELVIEW);
}

void drawSquare() {
    glBegin(GL_QUADS);
    glVertex2f(  0.0f,   0.0f);
    glVertex2f(100.0f,   0.0f);
    glVertex2f(100.0f, 100.0f);
    glVertex2f(  0.0f, 100.0f);
    glEnd();
}
```

```
void renderScene() {
    glPointSize(1);
    glColor3f(1.0f, 0.0f, 0.3f);

    glPushMatrix();
    glTranslatef( 150.0f,  160.0f, 0.0f);
    glScalef(1.5f, 1.5f, 1.0f);
    glRotatef((float)kut, 0.0f, 0.0f, 1.0f);
    glTranslatef(-50.0f, -50.0f, 0.0f);
    drawSquare();
    glPopMatrix();

    glPushMatrix();
    glTranslatef( 400.0f,  160.0f, 0.0f);
    glScalef(1.5f, 1.5f, 1.0f);
    glRotatef(-(float)kut, 0.0f, 0.0f, 1.0f);
    glTranslatef(-50.0f, -50.0f, 0.0f);
    drawSquare();
    glPopMatrix();
}
```

Osvrnimo se još na dobru praksu prilikom rada s *GLUT*-om (i sličnim bibliotekama): nije dobro direktno pozivati metodu za crtanje. Umjesto toga, kada nešto promijeni stanje "svijeta" koji se prikazuje, bolje je to dojaviti *GLUT*-u pozivom metode glutPostRedisplay();, i osloniti se na to da sam *GLUT* pokrenuti ponovno iscrtavanje. Naime, možda se još nešto promijenilo zbog čega će *GLUT* ionako pokrenuti ponovno crtanje (npr. uništen je dio prozora, došlo je do promjene veličine prozora i sl.), pa se time može izbjeći višestruko pozivanje metode za iscrtavanje.

Drugi detalj na koji bi trebalo paziti jest mjesto gdje se mijenja stanje "svijeta" koji se prikazuje. Ažuriranje tog stanja u metodi koja istovremeno crta nije dobra – naime, mi nemamo kontrolu kada će se i koliko često ta metoda za crtanje pozivati. Čak i kada koristimo mehanizam vremenskih sklopova, *GLUT* zbog različitih razloga metodu za crtanje može pozivati i češće. Kako je najčešće važno da se stanje svijeta mijenja zadanom brzinom, metoda za crtanje je loše mjesto za takve izmjene. Umjesto toga, izmjene treba raditi samo u metodi koju poziva vremenski sklop (u našem primjeru, to je metoda animate).

1.6 Ponavljanje

1. Što je OpenGL? Što označavaju pojmovi: OpenGL, GLU te GLUT?

2. Koja je razlika između poziva naredbe glutInitDisplayMode (...); s argumentima GLUT_SINGLE odnosno GLUT_DOUBLE?

3. Kako se radi s dvostrukim spremnikom, i što se dobiva njegovom uporabom?
4. Kako izgleda osnovna struktura programa koji koristi GLUT?
5. Ako program pisan uporabom GLUT-a treba reagirati na pritiske tipki na tipkovnici, što je potrebno napraviti?
6. Koje nam primitive za crtanje nudi OpenGL, tj. što može doći kao argument naredbe glBegin (...); ?
7. Koja je razlika između primitiva GL_LINE_LOOP i GL_POLYGON?
8. Koja nam dva tipična scenarija omogućava GLUT kada govorimo o potpori za animaciju?
9. Na koji se način može tražiti od GLUT-a da ponovno pokrene crtanje scene?

Poglavlje 2

Matematičke osnove u računalnoj grafici

Naše upoznavanje s računalnom grafikom, kako statičkom tako i interaktivnom, započet ćemo kroz matematički pogled na računalnu grafiku. U okviru ovog poglavlja najprije ćemo se upoznati s notacijom koju ćemo koristiti kroz ovu knjigu. Zatim slijedi upoznavanje s pojmovima *točka*, *vektor*, *pravac* i *ravnina*. Konačno, pred kraj poglavlja upoznat ćemo se s alternativnim načinom prikaza točaka koji se često koristi u računalnoj grafici: prikaz *homogenim* koordinatama.

2.1 Način označavanja

Točka će se obično označavati kao T_X, pri čemu će slovo X biti zamijenjeno u S ako se govori o početnoj točki pravca, u E ako se govori o završnoj točki pravca, odnosno u P ukoliko se govori o proizvoljnoj točki pravca. Termini početna točka pravca i završna točka pravca bit će jasniji nakon poglavlja 2.3; želimo li biti korektni, trebali bismo govoriti o početnoj i završnoj točki *linijskog segmenta* odnosno *dužine*.

Zapis točke T_X po komponentama bit će $(T_{X_1}, T_{X_2}, ..., T_{X_n})$ pri čemu je T_{X_i} i-ta komponenta točke.

Vektori će se označavati slično kao i točke. Vektor pravca bit će označen kao $\vec{v}_p$, odnosno po komponentama $(v_{p_1}, v_{p_2}, ..., v_{p_n})$.

Matrice ćemo označavati velikim štampanim i podebljanim slovima. Npr. karakteristična matrica pravca nosit će oznaku $\mathbf{L}$.

2.2 Točka i vektor

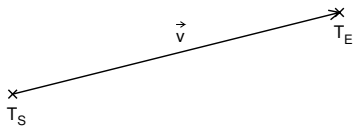
Točka je matematički pojam koji se u pravilu ne definira. No potrebno je uvesti neke osnovne pojmove. Točka je element prostora (odnosno njegova osnovna

građevna nedjeljiva cjelina). Zbog toga označavanje točke ovisi o prostoru u kojem se ta točka nalazi. Ono osnovno što određuje način označavanja točke jest dimenzionalnost prostora. Svaka točka bit će označena svojim koordinatama, i to s toliko koordinata kolika je dimenzija prostora. Tako će točka u 2D prostoru biti označena pomoću dvije koordinate: x i y. U 3D prostoru točka će biti označena pomoću tri koordinate: x, y i z. U nastavku ćemo točke označavati kao uređene n-torke koordinata, npr. (x, y, z) ili kao matricu $\begin{bmatrix} x & y & z \end{bmatrix}$.

Vektor ćemo promatrati kao usmjerenu dužinu, iako riječ "dužina" možda i nije baš najprikladnija. Naime, vektor će nam obično služiti kao gradijent, tj. pokazatelj koji govori za koliko se nešto mijenja. Vektore ćemo označavati pomoću strelice iznad imena što je uobičajena matematička notacija. Zapis vektora, isto kao i točke ovisi o prostoru u kojem se taj vektor opisuje, te će imati onoliko komponenti kolika je dimenzionalnost prostora. Zbog toga ćemo vektore također zapisivati kao uređene n-torke, npr. $\vec{v} = (x, y, z)$. Kako ćemo u nastavku govoriti o računalnoj grafici, vektore ćemo obično razapinjati između dvije točke, i to na sljedeći način (slika 2.1): vektor razapet između točke T_S i točke T_E kreće iz točke T_S a završava u točki T_E; računa se prema relaciji:

$$\vec{v} = T_E - T_S \tag{2.1}$$

Pri tome se točke promatraju kao radij-vektori, pa je oduzimanje upravo ono



Slika 2.1: Vektor između dviju točaka

vektorsko: i-ta komponenta rezultata dobije se tako da se oduzmu i-te komponente točaka. Treba imati na umu da vektore možemo zbrajati i oduzimati i da će rezultat biti opet vektor.

Primjer: 1

Zadane su dvije točke u 3D prostoru: $T_S = (3, 1, 7)$ i $T_E = (5, 0, 11)$. Izračunajte vektor koji one razapinju (uz prethodno utvrđenu konvenciju da je T_S početna točka).

Rješenje:

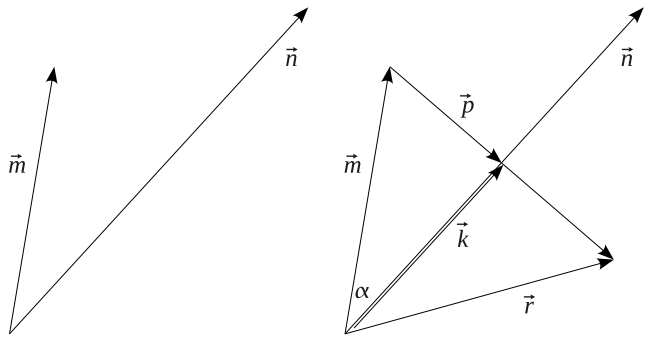
$$\begin{aligned} \vec{v} &= T_E - T_S \\ &= (5, 0, 11) - (3, 1, 7) \\ &= (5 - 3, 0 - 1, 11 - 7) \\ &= (2, -1, 4) \end{aligned}$$

Primjer: 2

Zadani su proizvoljni vektori $\vec{m}$ i $\vec{n}$. Pronaći vektor $\vec{r}$ koji čini reflektirani vektor vektora $\vec{m}$ s obzirom na vektor $\vec{n}$. Provjerite rezultat na konkretnom primjeru $\vec{n} = [\ 3 \quad 3\ ]$ i $\vec{m} = [\ 2 \quad 3\ ]$.

Rješenje:

Prvi korak u rješavanju ovog zadatka bit će izrada kvalitetne skice. Problem je prikazan na slici 2.2a. Pomoćni vektori kao i traženi vektor $\vec{r}$ prikazani su na slici 2.2b.



(a) Zadatak (b) Skica rješenja

Slika 2.2: Pronalazak reflektiranog vektora

Pogledamo li sliku 2.2b, vidimo da možemo pisati:

$$\vec{r} = \vec{m} + 2 \cdot \vec{p}.$$

Stoga ćemo se najprije fokusirati na pronalazak vektora $\vec{p}$. Vektor $\vec{m}$ projiciran na vektor $\vec{n}$ označen je s $\vec{k}$. To je vektor koji je kolinearan s vektorom $\vec{n}$, pri čemu mu je magnituda određena izrazom $||\vec{m}|| \cdot \cos(\alpha)$, pa možemo pisati:

$$\vec{k} = \frac{\vec{n}}{||\vec{n}||} \cdot ||\vec{m}|| \cdot \cos(\alpha).$$

Kosinus kuta možemo izračunati direktno iz vektora $\vec{m}$ i $\vec{n}$. Kako ti vektori nisu normirani, sjetimo se da vrijedi opći izraz:

$$\cos(\alpha) = \frac{\vec{m} \cdot \vec{n}}{||\vec{m}|| \cdot ||\vec{n}||}.$$

Uvrštavanjem dalje slijedi:

$$\vec{k} = \frac{\vec{n}}{||\vec{n}||} \cdot ||\vec{m}|| \cdot \frac{\vec{m} \cdot \vec{n}}{||\vec{m}|| \cdot ||\vec{n}||} = \frac{\vec{n}}{||\vec{n}||} \cdot \frac{\vec{m} \cdot \vec{n}}{||\vec{n}||}.$$

Sada možemo doći do traženog vektora $\vec{p}$. Iz slike 2.2b slijedi da je $\vec{m} + \vec{p} = \vec{k}$, pa je stoga $\vec{p} = \vec{k} - \vec{m}$. Uvrštavanjem u izraz za $\vec{r}$ slijedi:

$$\begin{aligned}\vec{r} &= \vec{m} + 2 \cdot \vec{p} \\ &= \vec{m} + 2 \cdot (\vec{k} - \vec{m}) \\ &= \vec{m} + 2 \cdot \vec{k} - 2 \cdot \vec{m} \\ &= 2 \cdot \vec{k} - \vec{m} \\ &= 2 \cdot \frac{\vec{n}}{||\vec{n}||} \cdot \frac{\vec{m} \cdot \vec{n}}{||\vec{n}||} - \vec{m}\end{aligned}$$

Primjenimo to na konkretnom primjeru. Kao reflektirani vektor morali bismo dobiti vektor $\vec{r} = [\ 3 \quad 2\ ]$, pa provjerimo to. Norma vektora $\vec{n}$ je $\sqrt{3^2 + 3^2} = \sqrt{18}$.

$$\begin{aligned}\vec{r} &= 2 \cdot \frac{\vec{n}}{||n||} \cdot \frac{\vec{m} \cdot \vec{n}}{||\vec{n}||} - \vec{m} \\ &= 2 \cdot \frac{[\ 3 \quad 3\ ]}{\sqrt{18}} \cdot \frac{[\ 2 \quad 3\ ] \cdot [\ 3 \quad 3\ ]}{\sqrt{18}} - [\ 2 \quad 3\ ] \\ &= 2 \cdot \frac{[\ 3 \quad 3\ ]}{\sqrt{18}} \cdot \frac{2 \cdot 3 + 3 \cdot 3}{\sqrt{18}} - [\ 2 \quad 3\ ] \\ &= 2 \cdot \frac{[\ 3 \quad 3\ ]}{18} \cdot 15 - [\ 2 \quad 3\ ] \\ &= [\ 5 \quad 5\ ] - [\ 2 \quad 3\ ] \\ &= [\ 3 \quad 2\ ]\end{aligned}$$

2.2.1 Skalarni i vektorski produkt

Skalarni i vektorski produkt dva su temeljna operatora linearne algebre, i kao takvi čine nezaobilazan alat računalne grafike. Stoga ćemo se ovdje samo ukratko podsjetiti kako su definirani, i kako ih najčešće koristimo.

Skalarni produkt

Skalarni produkt n-dimenzionalnih vektora $\vec{A} = (a_1, a_2, ..., a_n)$ i $\vec{B} = (b_1, b_2, ..., b_n)$ definiran je kao suma umnožaka parova odgovarajućih koordinata (odnosno komponenata) obaju vektora. Dakle:

$$\vec{A} \cdot \vec{B} = (a_1, a_2, ..., a_n) \cdot (b_1, b_2, ..., b_n) = a_1 \cdot b_1 + a_2 \cdot b_2 + ... + a_n \cdot b_n$$

Od zanimljivijih svojstava skalarnog produkta spomenimo sljedeća dva svojstva.

- Dva vektora su međusobno ortogonalna (tj. okomita) ako i samo ako im je skalarni produkt jednak nuli. Primjetite kako se u ovoj definiciji krije ekvivalencija: ako je skalarni produkt 0, vektori su okomiti; odnosno, ako su okomiti, skalarni produkt je nula. Ovo ćemo često koristiti u grafici. Možemo pisati:

$$\vec{A} \cdot \vec{B} = 0 \Leftrightarrow \vec{A} \perp \vec{B}.$$

- Skalarni produkt dvaju vektora $\vec{A}$ i $\vec{B}$ računa se pomoću kosinusa kuta koji oni zatvaraju. Točnije, vrijedi sljedeće:

$$\cos(\angle(\vec{A}, \vec{B})) = \frac{\vec{A} \cdot \vec{B}}{||\vec{A}|| \cdot ||\vec{B}||}.$$

Primjer: 3

Pronađite vektor koji je okomit na vektor $\vec{m} = (1, 2, 4)$.

Rješenje:

Poslužit ćemo se skalarnim produktom. Uočimo da je zadani vektor trodimenzionalni, pa će i traženi vektor $\vec{n}$ biti trodimenzionalni. Možemo pisati:

$$(n_x, n_y, n_z) \cdot (1, 2, 4) = 0 \qquad \Rightarrow \qquad n_x \cdot 1 + n_y \cdot 2 + n_z \cdot 4 = 0$$

Osim ove jednadžbe – jednadžbe od tri nepoznanice – nemamo niti jedan drugi izraz kojim bismo mogli više ograničiti traženi vektor. Evo i zašto. Zamislimo da smo u koordinatnom sustavu vektor $\vec{m}$ fiksirali u ishodište. Taj vektor tada je okomit na ravninu koja prolazi kroz ishodište. Svaki vektor koji se nalazi u toj ravnini bit će jedno moguće rješenje. Postavimo li u tu ravninu novi dvodimenzionalni koordinatni sustav, svaki vektor u toj ravnini bit će određen s dva parametra: svojom koordinatom vezanom uz prvu os, te svojom koordinatom vezanom uz drugu os. Tek kada ta dva parametra fiksiramo, gornja će nam jednadžba dati preostalu treću vrijednost. Zaključak ovog misaonog eksperimenta jest uočiti da imamo dva stupnja slobode, pa možemo sami dodati dvije linearno nezavisne jednadžbe s gornjom, koje će rezultirati jednoznačnim rješenjem.

Primjerice, gledajući gornji izraz, možemo tražiti da je $n_x = 1$. Tada ostajemo na jednadžbi:

$$1 \cdot 1 + n_y \cdot 2 + n_z \cdot 4 = 0 \qquad \Rightarrow \qquad n_y \cdot 2 + n_z \cdot 4 = -1.$$

Dalje, možemo poželjeti da je $n_y = 2$, čime se jednadžba dalje reducira na:

$$2 \cdot 2 + n_z \cdot 4 = -1 \qquad \Rightarrow \qquad n_z \cdot 4 = -5.$$

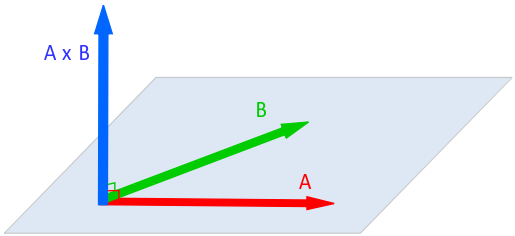
Slijedi:

$$n_z = \frac{-5}{4}.$$

Provjerimo da su vektori $(1, 2, 4)$ i pronađeni vektor $(1, 2, \frac{-5}{4})$ doista okomiti:

$$(1, 2, \frac{-5}{4}) \cdot (1, 2, 4) = 1 \cdot 1 + 2 \cdot 2 + \frac{-5}{4} \cdot 4 = 1 + 4 + (-5) = 0$$

Ako je potrebno pronaći bilo koji okomit vektor, postupak prikazan u prethodnom primjeru je sasvim prihvatljiv. Međutim, u praksi se češće umjesto ovakvih "proizvoljnih" ograničenja kao jedno od ograničenja nameće da je norma tog vektora jednaka 1 (dakle, da je pronađeni vektor normiran). I uz takvo ograničenje, ostaje nam još jedan stupanj slobode koji je potrebno fiksirati. U praksi, okomiti nam vektori najčešće trebaju kao normale ravnine. Kako je ravnina u parametarskom obliku zadana preko dva vektora koji leže u njoj, naš je zadatak pronaći vektor koji je okomit na oba – čime smo ograničili početni jednadžbu na samo jedan stupanj slobode, pa je dovoljno zadati još jedno ograničenje kojim ćemo sam vektor fiksirati.



Slika 2.3: Vektorski produkt

Treba međutim paziti kako zadajemo to treće ograničenje. Primjerice, ako tražimo da vrijedi:

$$n_x + n_y + n_z = 1$$

dobit ćemo jednoznačno definiran vektor. Međutim, ako tražimo da norma tog vektora bude 1, dakle:

$$\sqrt{n_x \cdot n_x + n_y \cdot n_y + n_z \cdot n_z} = 1 \qquad \Rightarrow \qquad n_x \cdot n_x + n_y \cdot n_y + n_z \cdot n_z = 1$$

bitno smo zakomplicirali izračun (jer imamo kvadratnu jednadžbu), i dodatno, opet nam je ostao jedan stupanj slobode – rekli smo da želimo vektor norme 1, ali takva postoje dva: onaj koji gleda na gornju stranu ravnine, te onaj koji gleda na donju stranu ravnine, pa i to treba dodatno specificirati. Stoga se u slučaju da na dva zadana vektora želimo na što jednostavniji način pronaći treći okomit (i ne zanimaju nas daljnja "svojstva" tog vektora), često koristi i vektorski produkt opisan u nastavku.

Vektorski produkt

Vektorski produkt tipično razmatramo u trodimenzionalnom prostoru. Vektorski produkt dvaju 3-dimenzionalnih vektora $\vec{A} = (a_1, a_2, a_3)$ i $\vec{B} = (b_1, b_2, b_3)$ definiran je kao:

$$\vec{A} \times \vec{B} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix},$$

što dalje možemo razriješiti kao:

$$\vec{A} \times \vec{B} = \vec{i} \cdot (a_2 \cdot b_3 - a_3 \cdot b_2) - \vec{j} \cdot (a_1 \cdot b_3 - a_3 \cdot b_1) + \vec{k} \cdot (a_1 \cdot b_2 - a_2 \cdot b_1).$$

Od zanimljivijih svojstava vektorskog produkta spomenimo sljedeća četiri svojstva.

- Vektorski produkt $\vec{n}$ dvaju vektora $\vec{A}$ i $\vec{B}$ okomit je i na $\vec{A}$ i na $\vec{B}$ (slika 2.3).

$$(\vec{A} \times \vec{B}) \perp \vec{A} \qquad \wedge \qquad (\vec{A} \times \vec{B}) \perp \vec{B}.$$

U ovo se možete jednostavno uvjeriti tako da pogledate skalarne produkte $\vec{n} \cdot \vec{A}$ i $\vec{n} \cdot \vec{B}$.

- Smjer vektora $\vec{n}$ koji predstavlja vektorski produkt dvaju vektora $\vec{A}$ i $\vec{B}$ određen je pravilom desne ruke. Ako prsti desne ruke pokazuju od vektora $\vec{A}$ prema vektoru $\vec{B}$, palac pokazuje smjer vektora $\vec{n}$.

- Norma vektorskog produkta $||\vec{n}||$ dvaju vektora $\vec{A}$ i $\vec{B}$ jednaka je:

$$||\vec{A} \times \vec{B}|| = ||\vec{A}|| \cdot ||\vec{B}|| \cdot \sin(\angle(\vec{A}, \vec{B}))$$

  pri čemu se kao kut gleda onaj manji. Geometrijska interpretacija ove činjenice jest da je norma vektorskog produkta vektora $\vec{A}$ i $\vec{B}$ jednaka površini paralelograma što ga razapinju vektori $\vec{A}$ i $\vec{B}$, odnosno jednaka je dvostrukoj vrijednosti površine trokuta koji razapinju ti vektori. Ovu činjenicu kasnije ćemo koristiti na više mjesta, a jedan od primjera će biti i izračun baricentričnih koordinata.

- Posljedica prethodnog svojstva jest da je vektorski produkt dvaju kolinearnih vektora jednak nul-vektoru. Dakle, ako su vektori kolinearni, vektorski produkt ima normu nula.

2.3 Pravac

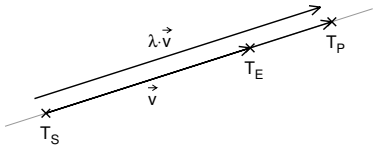
Pravac je također pojam koji se ne definira, no najopćenitije govoreći može se reći da je pravac skup točaka. U tom smislu, točke koje pripadaju pravcu možemo matematički opisati na nekoliko načina.

2.3.1 Jednadžba pravca

Kada govorimo o jednadžbi pravca, ideja je pronaći takav matematički izraz (jednadžbu) koji će vrijediti za sve točke koje pripadaju pravcu. Broj načina na koji ovo možemo napraviti ovisi o dimenzionalnosti prostora u kojem promatramo pravac. Ako se radi o 2D prostoru, uobičajeno možemo napisati *implicitni oblik* jednadžbe pravca, najčešće možemo napisati i *eksplicitni oblik* jednadžbe pravca, a uvijek možemo napisati jednadžbu pravca u *parametarskom obliku*. U 3D prostoru, broj mogućnosti se smanjuje, i tu uvijek koristimo parametarski oblik jednadžbe pravca. Pa upoznajmo se najprije s tim oblikom.

Parametarski oblik jednadžbe pravca

Ideja parametarskog zapisa krivulje – pa tako i pravca kao jedne specifične vrste krivulja – jest pomoću konačnog broja parametara opisati sve točke koje



Slika 2.4: Pravac dobiven skaliranjem vektora

pripadaju dotičnoj krivulji. Za opis pravca dovoljan je samo jedan parametar, s obzirom da su promjene po svim koordinatama linearno vezane. Jednadžbu možemo dobiti iz slike 2.4.

$$T_P = (T_E - T_S) \cdot \lambda + T_S = \vec{v}_p \cdot \lambda + T_S \tag{2.2}$$

Formula zapravo kaže: vektor $\vec{v}_p$ skaliran parametrom λ dodaj točki T_S i dobit ćeš jednu točku pravca. Ovo napravi za svaki $\lambda \in \mathbb{R}$ i dobit ćeš sve točke pravca.

Pojam "skalirati" parametrom λ znači svaku komponentu vektora $\vec{v}_p$ pomnožiti realnim brojem λ. Time se norma vektora $\vec{v}_p$ promijeni $|\lambda|$ puta, te vrijedi:

$$||\lambda \cdot \vec{v}_p|| = |\lambda| \cdot ||\vec{v}_p||$$

Npr. prema slici 2.4, ako točki T_S dodamo $\vec{v}_p$ uz $\lambda = 1$ doći ćemo do točke T_E. No ako je λ veći od jedan, tada ćemo doći do neke točke koja se krenuvši od točke T_S nalazi iza točke T_E. Ako je $\lambda \in \langle 0, 1\rangle$, dolazimo do točaka koje su između točke T_S i točke T_E, dok za $\lambda < 0$ dolazimo do točaka koje su ispred točke T_S. Primjerice, za $\lambda = 0.5$, dobit ćemo točku koja se nalazi točno na pola puta između T_S i T_E, a za $\lambda = -1$ dolazimo do točke koja je ispred točke T_S i to na jednakoj udaljenosti od točke T_S kao što je točka T_E udaljena od točke T_S.

U jednadžbi pravca koju smo prethodno napisali krije se zapravo onoliko jednadžbi koliko točke imaju komponenata. Važno je za uočiti da ovakav zapis pravca donekle i usmjerava pravac. Naime, porastom parametra λ točka T_P giba se u smjeru vektora $\vec{v}_p$ odnosno od točke T_S prema točki T_E i dalje. Ovakav oblik zapisa pravca pogodan je i za određivanje gdje se neka zadana točka pravca nalazi u odnosu na točke T_S i T_E. Vrijedi sljedeće. Ako je za zadanu točku T_P parametar λ:

- negativan, točka T_P je ispred točke T_S;
- nula, točka T_P se upravo poklapa s točkom T_S;
- pozitivan i manji od 1, točka T_P je između točaka T_S i T_E;
- jednak 1, točka T_P se upravo poklapa s točkom T_E; te

- pozitivan i veći od 1, točka T_P je iza točke T_E.

Raspišemo li jednadžbu (2.2) po komponentama, dobivamo:

$$(T_{P_1}, T_{P_2}, ..., T_{P_n}) = (v_{p_1}, v_{p_2}, ..., v_{p_n}) \cdot \lambda + (T_{S_1}, T_{S_2}, ..., T_{S_n}) \tag{2.3}$$

što možemo prikazati i u matričnom obliku:

$$(T_{P_1}, T_{P_2}, ..., T_{P_n}) = \begin{bmatrix} \lambda & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{p_1} & v_{p_2} & ... & v_{p_n} \\ T_{S_1} & T_{S_2} & ... & T_{S_n} \end{bmatrix} = \begin{bmatrix} \lambda & 1 \end{bmatrix} \cdot \mathbf{L} \tag{2.4}$$

gdje slovo **L** stoji za *karakterističnu matricu* pravca. Kako jednadžba mora biti zadovoljena za svaku komponentu zasebno, gornji zapis jednadžbe raspada se na n jednadžbi.

Matrični oblik zapisa u računalnoj grafici iznimno je važan jer je pogodan za zapis na računalu. Matematičke pojmove koje ovdje obrađujemo trebat ćemo implementirati na računalu. Kako pri toj implementaciji nije pogodno koristiti nizove varijabli već polja, karakterističnu matricu pravca, matricu **L**, zapisivat ćemo kao 2D polje podataka.

Računanje jednadžbe pravca kroz dvije točke

Neka su zadane dvije točke kroz koje prolazi pravac. Točka T_S neka bude početna točka pravca, a točka T_E neka bude završna točka pravca. Jednadžbu pravca kroz te dvije točke možemo izračunati temeljem izraza (2.2):

$$T_P = (T_E - T_S) \cdot \lambda + T_S = \vec{v}_p \cdot \lambda + T_S.$$

Jedina nepoznanica je vrijednost vektora smjera $\vec{v}_p$, no taj vektor možemo odrediti direktno iz izraza (2.2), čime dobivamo:

$$v_p = T_E - T_S. \tag{2.5}$$

Primjer: 4

Pravac u 2D prostoru zadan je dvjema točkama: $T_S = (1, 1)$ i $T_E = (3, 2)$. Kako glasi parametarski oblik jednadžbe tog pravca?

Rješenje:

Prema (2.5) računamo vektor smjera pravca:

$$\begin{aligned} \vec{v} &= T_E - T_S \\ &= (3, 2) - (1, 1) \\ &= (3 - 1, 2 - 1) \\ &= (2, 1) \end{aligned}$$

Označimo li prvu koordinatu s x a drugu s y, izračunati vektor smjera nam govori da svaki puta kada se y poveća za 1, x se poveća za 2. Iskoristimo ovaj zaključak da nabrojimo nekoliko točaka koje pripadaju pravcu. Znamo da točka $(1, 1)$ pripada pravcu. Prema uočenoj pravilnosti, povećamo li y za 1 i x za 2, dolazimo do točke $(3, 2)$ – uočimo da je ovo dodavanje zapravo odgovaralo dodavanju čitavog vektora v_p, odnosno izraza $v_p \cdot \lambda$ uz $\lambda = 1$ točki T_S, čime smo dobili upravo točku T_E, što je u skladu s prethodno utvrđenim ponašanjem. Uvećamo li ponovno koordinate točke, stižemo do točke $(5, 3)$ koja također pripada pravcu.

Traženi parametarski oblik jednadžbe pravca tada je:

$$T_P = (2, 1) \cdot \lambda + (1, 1).$$

Evo još jednog načina kako se može doći do parametarske jednadžbe pravca ako znamo dvije točke koje mu pripadaju. Krenimo iz parametarskog oblika jednadžbe pravca zapisanog matrično. Prisjetimo se izraza (2.4):

$$(T_{P_1}, T_{P_2}, ..., T_{P_n}) = \begin{bmatrix} \lambda & 1 \end{bmatrix} \cdot \mathbf{L}$$

U ovom izrazu jedina je nepoznanica matrica $\mathbf{L}$ koju treba odrediti. Budući da pravac kojeg tražimo za neku vrijednost parametra λ prolazi kroz točku T_S, a za neku drugu vrijednost parametra λ prolazi kroz točku T_E, ostavljena nam je sloboda izbora da sami odlučimo za koje će se to vrijednosti parametra λ dogoditi. Zbog toga ćemo se odlučiti za uobičajeni izbor: neka pravac prođe kroz točku T_S za $\lambda = 0$, a kroz točku T_E za $\lambda = 1$. Tada možemo pisati:

$$T_S = \begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \mathbf{L}$$

$$T_E = \begin{bmatrix} 1 & 1 \end{bmatrix} \cdot \mathbf{L}$$

gdje sada T_S i T_E promatramo kao jednoretčane matrice s onoliko stupaca koliko točke imaju koordinata. Ove dvije jednadžbe možemo stopiti u jednu matričnu jednadžbu:

$$\begin{bmatrix} T_S \\ T_E \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \mathbf{L}$$

Treba uočiti da T_S i T_E na lijevoj strani jednadžbe predstavljaju matrice $1 \times n$ pa matrica na lijevoj strani nije 2×1 već $2 \times n$. Kako je u jednadžbi sve poznato osim matrice $\mathbf{L}$, množeći jednadžbu s lijeva inverznom matricom uz $\mathbf{L}$ dobiva se:

$$\mathbf{L} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_S \\ T_E \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} T_S \\ T_E \end{bmatrix} = \begin{bmatrix} T_E - T_S \\ T_S \end{bmatrix} \tag{2.6}$$

pri čemu je matrica $\mathbf{L}$ dimenzija $2 \times n$.

Primjer: 5

Pravac u 2D prostoru zadan je dvjema točkama: $T_S = (1, 1)$ i $T_E = (3, 2)$. Kako glasi parametarski oblik jednadžbe tog pravca u matričnom zapisu? Za koju se vrijednost parametra dobiva točka $(-3, -1)$?

Rješenje:

Matricu $\mathbf{L}$ izračunat ćemo prema izrazu (2.6):

$$\mathbf{L} = \begin{bmatrix} T_E - T_S \\ T_S \end{bmatrix} = \begin{bmatrix} 3-1 & 2-1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

Traženi oblik jednadžbe pravca tada glasi:

$$[\ x \quad y\ ] = [\ \lambda \quad 1\ ] \cdot \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

Kako znamo da tražena točka $(-3, -1)$ leži na pravcu, možemo pisati:

$$[\ -3 \quad -1\ ] = [\ \lambda \quad 1\ ] \cdot \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

Slijedi:

$$[\ \lambda \quad 1\ ] = [\ -3 \quad -1\ ] \cdot \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}^{-1} = [\ -3 \quad -1\ ] \cdot \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} = [\ -2 \quad 1\ ].$$

Dakle, tražena vrijednost parametra je $\lambda = -2$.

Ovim korakom smo u osnovi rješavali sustav od dvije jednadžbe s jednom nepoznanicom, kojeg možemo dobiti i izravno iz prethodnog koraka, odnosno:

$$-3 = 2\lambda + 1$$

$$-1 = \lambda + 1$$

Ako u obje jednadžbe dobijemo istu vrijednost za nepoznanicu, znači da točka leži na pravcu, u protivnom točka ne leži na pravcu, odnosno ne postoji jedinstvani parametar λ za koji to vrijedi.

Valja napomenuti da u jednadžbi (2.4) točka na pravcu T_S može biti bilo koja točka, bitno je da je ta točka na pravcu. Vektor smjera pravca $\vec{v}_p$ isto tako može biti bilo koji vektor koji ima smjer dotičnog pravca. Drugim riječima, isti pravac možemo zapisati na beskonačno mnogo načina. Radi li se upravo o nekom zadanom pravcu provjerit ćemo tako da provjerimo kolinearnost vektora pravaca koji ih određuju, te pripadnosti točke promatranog pravca zadanom pravcu.

2.3.2 Posebni slučajevi jednadžbe pravca

2D prostor

U 2D prostoru svaka točka ima po dvije komponente, pa izraz (2.3):

$$(T_{P_1}, T_{P_2}, ..., T_{P_n}) = (v_{p_1}, v_{p_2}, ..., v_{p_n}) \cdot \lambda + (T_{S_1}, T_{S_2}, ..., T_{S_n})$$

prelazi u:

$$(T_{P_1}, T_{P_2}) = (v_{p_1}, v_{p_2}) \cdot \lambda + (T_{S_1}, T_{S_2}) \tag{2.7}$$

Umjesto oznaka T_{P_1} i T_{P_2} mogli smo koristi i oznake x_p, y_p, a umjesto oznaka T_{S_1} i T_{S_2} mogli smo koristi i oznake x_s, y_s, a umjesto oznaka v_{p_1} i v_{p_2} mogli smo koristi i oznake v_x, v_y. Tada bismo dobili prepoznatljiv oblik izraza:

$$(x_p, y_p) = (v_x, v_y) \cdot \lambda + (x_s, y_s).$$

Jednadžbu (2.7) možemo raspisati po komponentama i tako dobiti dvije jednadžbe:

$$T_{P_1} = v_{p_1} \cdot \lambda + T_{S_1}$$
$$T_{P_2} = v_{p_2} \cdot \lambda + T_{S_2}$$

Eliminacijom parametra λ dobiva se *eksplicitni oblik* jednadžbe pravca:

$$T_{P_2} - T_{S_2} = \frac{v_{p2}}{v_{p_1}}(T_{P_1} - T_{S_1}) \tag{2.8}$$

Vektor smjera $\vec{v_p}$ definiran je izrazom (2.5), što po komponentama možemo raspisati kao:

$$v_{p_1} = T_{E_1} - T_{S_1},$$
$$v_{p_2} = T_{E_2} - T_{S_2}.$$

Uvrštavanjem ovih izraza u eksplicitni oblik – izraz (2.7) – dobiva se:

$$T_{P_2} - T_{S_2} = \frac{T_{E_2} - T_{S_2}}{T_{E_1} - T_{S_1}}(T_{P_1} - T_{S_1}), \tag{2.9}$$

što nakon promjene imena odgovarajućih varijabli u "školska" imena daje:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1). \tag{2.10}$$

Pomnožimo li izraz (2.9) sa zajedničkim nazivnikom i potom malo preuredimo, dobit ćemo *implicitni oblik* jednadžbe pravca:

$$(T_{S_2} - T_{E_2}) \cdot T_{P_1} - (T_{S_1} - T_{E_1}) \cdot T_{P_2} - (T_{S_2} - T_{E_2}) \cdot T_{S_1} + (T_{S_1} - T_{E_1}) \cdot T_{S_2} = 0. \tag{2.11}$$

Koristimo li opet uobičajenu školsku notaciju, izraz prelazi u:

$$(y_1 - y_2) \cdot x - (x_1 - x_2) \cdot y - (y_1 - y_2) \cdot x_1 + (x_1 - x_2) \cdot y_1 = 0. \tag{2.12}$$

Općenito implicitni oblik zapisujemo kao:

$$a \cdot T_{P_1} + b \cdot T_{P_2} + c = 0, \tag{2.13}$$

odnosno u uobičajenoj notaciji, prepoznatljiv nam je oblik:

$$a \cdot x + b \cdot y + c = 0. \tag{2.14}$$

U ovom obliku treba primjetiti interpretaciju pojedinih komponenti. Vektorski gledano, vektor (a, b) određuje normalu na pravac (2.14) a vektor $(b, -a)$ određuje smjer pravca odnosno vektor koji je kolinearan s pravcem. Ako oblik (2.14) normiramo (tj. sve komponente podijelimo s $\sqrt{a^2 + b^2}$), tada uvrštavanje točke u jednadžbu (2.14) daje udaljenost te točke od pravca. Iz toga je očito da je koeficijent c upravo udaljenost ishodišta do zadanog pravca (vidi se ako se u izraz uvrste koordinate ishodišta).

Od zanimljivijih oblika vrijedno je još spomenuti i *segmentni oblik* jednadžbe pravca, koji direktno daje odsječke pravca na obje osi. Dobije se svođenjem implicitnog oblika na oblik:

$$\frac{T_{P_1}}{l_x} + \frac{T_{P_2}}{l_y} = 1. \tag{2.15}$$

Pri tome vrijednosti l_x i l_y predstavljaju odsječke na obje osi. Uočimo međutim da ovaj oblik ne postoji za sve pravce u 2D prostoru – primjerice, pokušajte ga napisati za pravac koji je paralelan s osi x ili paralelan s osi y. Iz eksplicitnog se oblika dobije:

$$\frac{T_{P_1}}{\frac{T_{S_2}v_{p_1} - T_{S_1}v_{p_2}}{-v_{p_2}}} + \frac{T_{P_2}}{\frac{T_{S_2}v_{p_1} - T_{S_1}v_{p_2}}{v_{p_1}}} = 1. \tag{2.16}$$

3D prostor

U 3D prostoru pravac nije moguće prikazati jednadžbom u eksplicitnom obliku. Naime, parametarski oblik jednadžbe pravca definiran izrazom (2.3):

$$(T_{P_1}, T_{P_2}, ..., T_{P_n}) = (v_{p_1}, v_{p_2}, ..., v_{p_n}) \cdot \lambda + (T_{S_1}, T_{S_2}, ..., T_{S_n})$$

u tom slučaju prelazi u:

$$(T_{P_1}, T_{P_2}, T_{P_3}) = (v_{p_1}, v_{p_2}, v_{p_3}) \cdot \lambda + (T_{S_1}, T_{S_2}, T_{S_3}) \tag{2.17}$$

pri čemu se svaka točka (ili vektor) opisuje pomoću tri koordinate. Ova jednadžba ekvivalent je tri jednadžbe (po jedna za svaku koordinatu), te je parametar λ nemoguće eliminirati tako da se dobije jedna jednadžba. No eliminacijom parametra λ iz dva para jednadžbi moguće je dobiti implicitni oblik jednadžbe pravca u tri dimenzije:

$$\frac{T_{P_1} - T_{S_1}}{v_{p_1}} = \frac{T_{P_2} - T_{S_2}}{v_{p_2}} = \frac{T_{P_3} - T_{S_3}}{v_{p_3}} \tag{2.18}$$

odnosno:

$$\frac{T_{P_1} - T_{S_1}}{T_{E_1} - T_{S_1}} = \frac{T_{P_2} - T_{S_2}}{T_{E_2} - T_{S_2}} = \frac{T_{P_3} - T_{S_3}}{T_{E_3} - T_{S_3}}. \tag{2.19}$$

Uz školske oznake točaka, izraz glasi:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} = \frac{z - z_1}{z_2 - z_1}.$$

Poteškoće

Kada zadajemo jednadžbu pravca, želimo biti u stanju pomoću nje prikazati sve moguće pravce. Krenemo li od segmentnog oblika (uz ograničenje razmatranja na zapisa pravca samo u 2D prostoru), vidimo da on ne može prikazati niti jedan

pravac paralelan s bilo kojom osi. Eksplicitni oblik ima poteškoća s pravcima paralelnim s ordinatom. Jedini oblik koji nema problema je implicitni oblik. Međutim, implicitni oblik pravca za primjenu u računalima nije praktičan. Zbog toga se kao relativno dobar oblik pokazuje parametarski oblik. Dodatan "plus" ovom obliku nosi i mogućnost matričnog zapisa, koji je idealan za primjenu u računalima, i obilno se koristi u grafičkom sklopovlju.

2.4 Homogeni prostor i homogene koordinate

2.4.1 Ideja

Dijeljenje s nulom jedan je od čestih problema pri geometrijskim proračunima. Npr. traženje sjecišta dva paralelna pravca na uobičajeni način rezultirat će dijeljenjem s nulom budući da se sjecište nalazi u beskonačnosti. Također, operacije translacije i perspektivne projekcije ne mogu se prikazati kao linearni operatori a to bismo htjeli. Da bi se ovakvi problemi izbjegli, uvodi se *homogeni prostor* i homogene koordinate. U tom prostoru translacija i perspektivna projekcija jesu linearni operatori; lakše je zapisivanje i računanje racionalnih Bézierovih krivulja, što ćemo vidjeti kasnije.

Evo razmišljanja vezanog uz prikaz točaka u beskonačnosti. Ako je neka od koordinata jednaka beskonačno, to znači da se može dobiti dijeljenjem s nulom. Da bi se ovo izbjeglo, potrebno je sve koordinate napisati u obliku razlomka:

$$T_{P_1} = \frac{T_{Ph_1}}{h}, T_{P_2} = \frac{T_{Ph_2}}{h}, \tag{2.20}$$

pri čemu su T_{P_1} i T_{P_2} standardne koordinate (kažemo još i koordinate u *radnom prostoru* odnosno *Euklidskom prostoru*), a T_{Ph_1} i T_{Ph_2} su homogene koordinate. Uvrštavanjem ovih izraza u implicitni oblik jednadžbe pravca dobivamo:

$$a \cdot \frac{T_{Ph_1}}{h} + b \cdot \frac{T_{Ph_2}}{h} + c = 0$$

odnosno

$$a \cdot T_{Ph_1} + b \cdot T_{Ph_2} + c \cdot h = 0. \tag{2.21}$$

U ovom slučaju koordinate T_{Ph_1} i T_{Ph_2} nikada neće biti jednake beskonačno. Općenito se može reći ovako. Zadana je neka proizvoljna točka T u n-dimenzionalnom prostoru. Ta točka T u homogenom će se prostoru opisivati pomoću $n+1$ koordinate. Pri tome će jednoj konkretnoj točki radnog prostora biti pridruženo beskonačno homogenih točaka. Npr. točka $(12, 24, 12)$ u radnom prostoru u tri dimenzije imat će u homogenom prostoru zapise: $(12, 24, 12, 1)$, $(6, 12, 6, 0.5)$, $(24, 48, 24, 2)$, ... Naime, uočite da se svaka od tih homogenih točaka nakon dijeljenja s homogenim parametrom pretvara u istu točku radnog prostora: $(12, 24, 12)$.

Poseban slučaj je zapis beskonačnosti: ako je homogena koordinata h točke T jednaka 0, točka leži u beskonačnosti. Ako dobijemo da se dva pravca sijeku u homogenoj točki $(6, 6, 4, 2)$, to znači da se u radnom prostoru sijeku u točki $(3, 3, 2)$. Ako se pak sijeku u homogenoj točki $(10, 10, 3, 0)$, tada su ti pravci u radnom prostoru paralelni i nemaju sjecišta. Valja napomenuti da kombinacija u kojoj je svih $n+1$ koordinata (tj. $(0, 0, 0, 0)$) nije dozvoljena. Međutim, homogene točke u kojima je barem jedna od prvih n koordinata različita od nule (ostale ne moraju biti) te u kojoj je $n + 1$-va koordinata jednaka 0 su dozvoljene (npr. $(1, 0, 0, 0)$, $(1, 0, 3, 0)$ i slično) - sve one predstavljaju točke u beskonačnosti.

Ponovimo ukratko najvažnije. Ako je točka u radnom prostoru zadana kao:

$$T_P = (T_{P_1}, T_{P_2}, ..., T_{P_n})$$

tada ta točka u homogenom prostoru ima zapis:

$$T_{Ph} = (T_{Ph_1}, T_{Ph_2}, ..., T_{Ph_n}, h)$$

pri čemu je definirana veza:

$$T_{P_i} = \frac{T_{Ph_i}}{h}, \text{ odnosno } T_{Ph_i} = T_{P_i} \cdot h. \tag{2.22}$$

2.4.2 Jednadžba 2D pravca u homogenom prostoru

U poglavlju 2.3.2 dana je jednadžba pravca u 2D u radnom prostoru (izraz (2.7)):

$$(T_{P_1}, T_{P_2}) = (v_{p_1}, v_{p_2}) \cdot \lambda + (T_{S_1}, T_{S_2})$$

Uvrštavanjem izraza (2.22) u tu jednadžbu dobiva se:

$$(T_{Ph_1}, T_{Ph_2}, h_P) = (v_{ph_1}, v_{ph_2}, v_h) \cdot \lambda + (T_{Sh_1}, T_{Sh_2}, h_S). \tag{2.23}$$

2.4.3 Jednadžba 3D pravca u homogenom prostoru

U poglavlju 2.3.2 dana je jednadžba pravca u 3D u radnom prostoru (izraz (2.17)):

$$(T_{P_1}, T_{P_2}, T_{P_3}) = (v_{p_1}, v_{p_2}, v_{p_3}) \cdot \lambda + (T_{S_1}, T_{S_2}, T_{S_3}).$$

Proširivanjem tog izraza uz izraz (2.22) dolazi se do:

$$(T_{Ph_1}, T_{Ph_2}, T_{Ph_3}, h_P) = (v_{ph_1}, v_{ph_2}, v_{ph_3}, v_h) \cdot \lambda + (T_{Sh_1}, T_{Sh_2}, T_{Sh_3}, h_S). \tag{2.24}$$

2.4.4 Implicitni oblik jednadžbe 2D pravca u homogenom prostoru

Krenemo li od implicitnog oblika jednadžbe pravca u 2D (izraz 2.13):

$$a \cdot T_{P_1} + b \cdot T_{P_2} + c = 0$$

i uvrstimo li u jednadžbu izraz za homogene koordinate 2.22

$$T_{P_i} = \frac{T_{Ph_i}}{h}, \text{ odnosno } T_{Ph_i} = T_{P_i} \cdot h$$

dolazimo do jednadžbe:

$$a \cdot \frac{T_{Ph_1}}{h} + b \cdot \frac{T_{Ph_2}}{h} + c = 0$$

odnosno nakon sređivanja:

$$a \cdot T_{Ph_1} + b \cdot T_{Ph_2} + c \cdot h = 0 \tag{2.25}$$

Ovaj se izraz može prikazati i u matričnom obliku, što je dosta čest slučaj. Tada on glasi:

$$\begin{bmatrix} T_{Ph_1} & T_{Ph_2} & h \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = 0 \tag{2.26}$$

Uvođenjem uobičajenih oznaka

$$T_P = \begin{bmatrix} T_{Ph_1} & T_{Ph_2} & h \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{2.27}$$

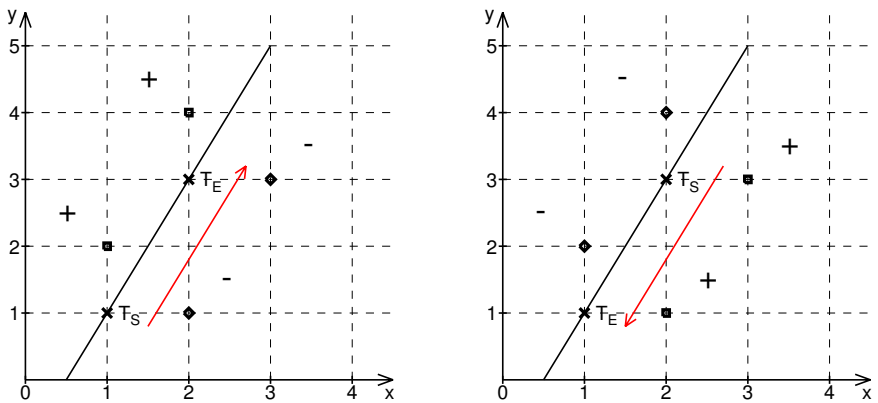
dolazi se do jednadžbe:

$$T_P \cdot \mathbf{G} = 0. \tag{2.28}$$

Ovdje se može razmotriti još jedan interesantan slučaj. Ako odaberemo točku T_P takvu da leži na pravcu, tada će gornja relacija biti zadovoljena. No što ako ubacimo proizvoljnu točku u jednadžbu pravca? Gornja relacija očito neće biti jednaka nuli, i na temelju tog rezultata možemo definirati odnos točke i pravca prema sljedećem kriteriju:

$$T_P \cdot \mathbf{G} \begin{cases} > 0, & T_P \text{ je iznad pravca,} \\ = 0, & T_P \text{ je na pravcu,} \\ < 0, & T_P \text{ je ispod pravca.} \end{cases} \tag{2.29}$$

Ovakva interpretacija vrijedi ako smo matricu $\mathbf{G}$ izračunali prema izrazu (2.12) odnosno (2.13) te ako je h pozitivan. U tom kontekstu je posebno važno obratiti pažnju na redosljed točaka – koja je početna a koja krajnja. Zamjenom redosljeda točaka okreću svi izvedeni zaključci; naime, dobit će se koeficijenti jednadžbe pravca koji se razlikuju u predznaku (svi su pomnoženi s -1) čime će se promijeniti smjer normale a time i interpretacija što je iznad a što ispod. Da bismo dobili bolji osjećaj što nam ovo pravilo zapravo govori, zamislite da se fizički krećete po zadanom pravcu, i to na način da ste krenuli od točke T_S i hodate prema točki T_E. U tom će slučaju za sve točke ravnine koje ne pripadaju pravcu a nalaze Vam se s lijeve strane vrijediti $T_P \cdot \mathbf{G} > 0$; za te točke kažemo da su iznad pravca po kojem se krećete. Za sve točke ravnine koje ne pripadaju pravcu a nalaze Vam se s desne strane vrijedit će $T_P \cdot \mathbf{G} < 0$; za te točke kažemo da su ispod pravca po kojem se krećete. Određivanje odnosa točke i pravca na ovaj način koristit ćemo kasnije u nizu algoritama (primjerice, kod ispitivanja odnosa točke i poligona), pa je važno razumjeti o čemu se tu radi. Stoga ćemo ovo pogledati i na konkretnom primjeru. Neka je zadan pravac (u radnom prostoru) s početnom točkom $T_S = [\ 1 \quad 1\ ]$ i završnom točkom $T_E = [\ 2 \quad 3\ ]$, kako je to prikazano na slici 2.5a.



(a) Prvi slučaj (b) Drugi slučaj

Slika 2.5: Odnos točke i pravca

Koristeći izraz (2.12) dolazimo do jednadžbe pravca:

$$-2x + 1y + 1 = 0$$

čija je pripadna matrica koeficijenata u skladu s izrazom (2.27) jednaka:

$$\mathbf{G} = \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}.$$

Izračunajmo sada za nekoliko točaka koje ne leže na tom pravcu umnožak $T_P \cdot \mathbf{G}$. Promotrit ćemo točke: [2 1], [3 3], [1 2] i [2 4]. Računamo redom (uz proširenje svake točke homogenom koordinatom 1):

$$\begin{bmatrix} 2 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix} = -2$$

$$\begin{bmatrix} 3 & 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix} = -2$$

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix} = 1$$

$$\begin{bmatrix} 2 & 4 & 1 \end{bmatrix} \cdot \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix} = 1$$

Zamislite li se sada da šećete po zadanom pravcu od točke T_S prema točki T_E, prve dvije točke koje smo upravo provjerili nalazit će Vam se desne strane; upravo smo pokazali da je za njih umnožak $T_P \cdot G$ negativan. Druge dvije točke koje smo upravo provjerili nalazit će Vam se lijeve strane; i za njih smo upravo pokazali da je umnožak $T_P \cdot G$ pozitivan.

Idemo sada okrenuti priču; zamijenimo vrijednosti točaka T_S i T_E, kako je to prikazano na slici 2.5b; neka je dakle $T_S = [\ 2 \quad 3\ ]$ i završnom točkom $T_E = [\ 1 \quad 1\ ]$. Koristeći izraz (2.12) dolazimo do jednadžbe pravca:

$$2x - 1y - 1 = 0$$

čija je pripadna matrica koeficijenata u skladu s izrazom (2.27) jednaka:

$$\mathbf{G} = \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix}.$$

Uočite da su vrijednost koeficijenata koje smo dobili jednakog iznosa samo suprotnog predznaka u odnosu na prethodni slučaj. Izračunajmo sada za iste četiri

točke koje ne leže na tom pravcu umnožak $T_P \cdot \mathbf{G}$. Računamo redom (uz proširenje svake točke homogenom koordinatom 1):

$$\begin{bmatrix} 2 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix} = 2$$

$$\begin{bmatrix} 3 & 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix} = 2$$

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix} = -1$$

$$\begin{bmatrix} 2 & 4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix} = -1$$

Dobili smo umnoške koji su jednakog iznosa i suprotnog predznaka u odnosu na prethodni slučaj. Zamislite li sada da šećete po zadanom pravcu od točke T_S prema točki T_E, prve dvije točke koje smo upravo provjerili nalazit će Vam se lijeve strane; upravo smo pokazali da je za njih umnožak $T_P \cdot G$ pozitivan. Druge dvije točke koje smo upravo provjerili nalazit će Vam se desne strane; i za njih smo upravo pokazali da je umnožak $T_P \cdot G$ negativan. Dakle, neovisno o situaciji koju imamo, umnožak će za točke s desne strane biti negativan a za točke s lijeve strane pozitivan. Ovo je vrlo važno za zapamtiti jer će nam omogućiti izradu niza jednostavnih algoritama koje ćemo koristiti u računalnoj grafici. Primjerice, poligone ćemo zadavati navođenjem vrhova na način da površina poligona bude uvijek desno od brida. Tada ćemo automatski znati da su vrhovi poligona zadani u smjeru kazaljke na satu. Znat ćemo da je neka točka unutar poligona ako se nalazi desno od svih bridova. Moći ćemo napisati jednostavan algoritam koji će provjeriti je li zadani poligon konveksan ili konkavan, i tako dalje.

Poznavanjem dviju točaka T_A i T_B na jednostavan se način može doći do pravca na kojem one leže ako se napravi vektorski produkt tih točaka. Neka su točke T_A i T_B zadane na sljedeći način:

$$T_A = (T_{A_1}, T_{A_2}, h_A) \text{ i } T_B = (T_{B_1}, T_{B_2}, h_B).$$

Njihov vektorski produkt (ili kraće $\times$-produkt) tada je:

$$\begin{aligned} T_A \times T_B &= \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ T_{A_1} & T_{A_2} & h_A \\ T_{B_1} & T_{B_2} & h_B \end{vmatrix} \\ &= \vec{i} \cdot (T_{A_2} h_B - T_{B_2} h_A) - \vec{j} \cdot (T_{A_1} h_B - T_{B_1} h_A) + \vec{k} \cdot (T_{A_1} T_{B_2} - T_{A_2} T_{B_1}). \end{aligned}$$

Ovo možemo dalje raspisati kao:

$$T_A \times T_B = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \begin{bmatrix} T_{A_2}h_B - T_{B_2}h_A \\ -(T_{A_1}h_B - T_{B_1}h_A) \\ T_{A_1}T_{B_2} - T_{A_2}T_{B_1} \end{bmatrix}$$
$$= \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \mathbf{G}.$$

Potrebno je skrenuti pažnju da je bitan redoslijed točaka pri izračunu vektorskog produkta. Ako zamijenimo redoslijed točaka tako da načinimo produkt $T_B \times T_A$, dobit ćemo isti pravac, no kriterij koji određuje što je iznad, a što ispod postaje upravo obrnuti. Isti učinak postići ćemo ako jednadžbu pravca pomnožimo s (-1).

Slično se poznavanjem dvaju pravaca može odrediti točka u kojoj se oni sijeku ako se napravi vektorski produkt. Neka su pravci određeni s:

$$\mathbf{G}_1 = \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix}, \mathbf{G}_2 = \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}.$$

$$G_1^T \times G_2^T = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{vmatrix}$$
$$= \vec{i} \cdot (b_1c_2 - b_2c_1) - \vec{j} \cdot (a_1c_2 - a_2c_1) + \vec{k} \cdot (a_1b_2 - a_2b_1).$$

Ovo možemo dalje raspisati kao:

$$G_1^T \times G_2^T = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \begin{bmatrix} b_1c_2 - b_2c_1 \\ -(a_1c_2 - a_2c_1) \\ a_1b_2 - a_2b_1 \end{bmatrix}$$
$$= T_P^T,$$

gdje je T_P prikaz točke sjecišta u homogenom obliku.

Operacija transponiranja nužna je ako se žele napisati jednadžbe u skladu s pravilima matričnog računa. Ovdje se može prodiskutirati opravdanost uvođenja homogenih koordinata. Naime, zahvaljujući homogenim koordinatama, matrični račun ne daje niti jedno dijeljenje. Ovo znači da će i paralelni pravci imati sjecište. Doista, ako su pravci paralelni, tada su im koeficijenti smjera jednaki i treća komponenta točke T_P iznosit će nula, što prema definiciji znači točku u beskonačnosti. To možemo jednostavno provjeriti. Jednadžba pravca u 2D s homogenim koordinatama glasi:

$$a \cdot T_{Ph_1} + b \cdot T_{Ph_2} + c \cdot h = 0$$

što se može napisati kao:

$$T_{Ph_2} = -\frac{a}{b} \cdot T_{Ph_1} - \frac{c}{b} \cdot h = -k \cdot T_{Ph_1} - \frac{c}{b} \cdot h$$

gdje je k koeficijent smjera pravca. Ako pravci G_1 i G_2 imaju jednake koeficijente smjera, to prema gornjoj relaciji znači da vrijedi:

$$-\frac{a_1}{b_1} = -\frac{a_2}{b_2} \Rightarrow a_1 b_2 = a_2 b_1 \Rightarrow a_1 b_2 - a_2 b_1 = 0$$

kako je poslijednja relacija upravo jednadžba zadnje koordinate točke sjecišta, slijedi da je uz paralelne pravce zadnja koordinata (tj. homogeni faktor) točke sjecišta jednaka 0.

2.5 Ravnina

2.5.1 Jednadžba ravnine

Ravnina se kao pojam u općem smislu može definirati u n-dimenzionalnom prostoru. Međutim, kako se u računalnoj grafici koriste jedino ravnine u 3D prostoru, razmatranja ćemo ograničiti na taj tip. U 3D prostoru ravnina je određena s dva vektora koji leže u njoj i nisu kolinearni, te jednom točkom ravnine, koja fiksira te vektore. Krenuvši ovakvom definicijom dolazimo vrlo jednostavno do jednadžbe ravnine u parametarskom obliku:

$$T_R = \vec{v}_A \cdot \lambda + \vec{v}_B \cdot \mu + T_S \tag{2.30}$$

što možemo pročitati i na sljedeći način.

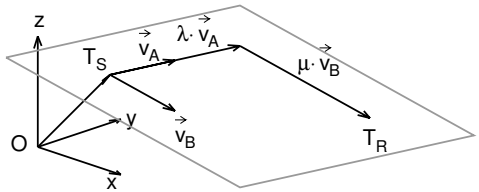
> *Krenuvši od točke T_S svaka točka ravnine može se dobiti pomakom za $\vec{v}_A \cdot \lambda$ i $\vec{v}_B \cdot \mu$, pri čemu su λ i μ realni parametri. Pri tome točka T_R predstavlja bilo koju točku ravnine.*

Slika 2.6 ovo jasno ilustrira.
Izraz (2.30) ekvivalentan je sustavu od tri jednadžbe:

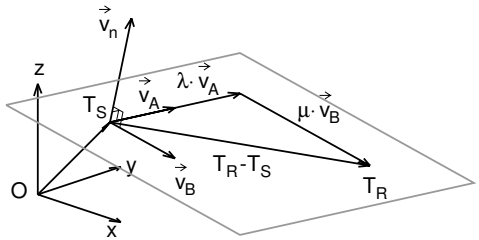
$$\begin{aligned} T_{R_1} &= v_{A_1} \cdot \lambda + v_{B_1} \cdot \mu + T_{S_1}, \\ T_{R_2} &= v_{A_2} \cdot \lambda + v_{B_2} \cdot \mu + T_{S_2}, \\ T_{R_3} &= v_{A_3} \cdot \lambda + v_{B_3} \cdot \mu + T_{S_3}, \end{aligned}$$

što se može prikazati i matričnim zapisom:

$$T_R = \begin{bmatrix} T_{R_1} & T_{R_2} & T_{R_3} \end{bmatrix} = \begin{bmatrix} \lambda & \mu & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{A_1} & v_{A_2} & v_{A_3} \\ v_{B_1} & v_{B_2} & v_{B_3} \\ T_{S_1} & T_{S_2} & T_{S_3} \end{bmatrix}.$$



Slika 2.6: Ravnina određena točkom i dvama vektorima



Slika 2.7: Ravnina i njezina normala

No, ovaj se oblik može i pojednostavniti. Pretpostavimo da imamo vektor $\vec{v}_n$ koji je okomit i na vektor $\vec{v}_A$ i na vektor $\vec{v}_B$. Takav vektor zovemo *normala ravnine*, i ilustriran je na slici 2.7.
U parametarskoj jednadžbi T_S prebacimo na lijevu stranu:

$$T_R - T_S = \vec{v}_A \cdot \lambda + \vec{v}_B \cdot \mu$$

te razliku točaka na lijevoj strani proglasimo vektorom $\vec{v}_R$:

$$\vec{v}_R = \vec{v}_A \cdot \lambda + \vec{v}_B \cdot \mu.$$

Pomnožimo sada sve vektorom $\vec{v}_n$:

$$\vec{v}_R \cdot \vec{v}_n = 0. \tag{2.31}$$

Cijela desna strana jednadžbe je nestala jer su vektori $\vec{v}_A$ i $\vec{v}_n$ kao i vektori $\vec{v}_B$ i $\vec{v}_n$ međusobno okomiti, pa im je skalarni produkt jednak nuli. Ovo nas vodi na zapis ravnine pomoću njezine normale:

$$(T_R - T_S) \cdot \vec{v}_n = 0. \tag{2.32}$$

Vektor $\vec{v}_n$ naziva se vektorom normale (kraće normala) ravnine iz očitih razloga. Ako znamo vektore $\vec{v}_A$ i $\vec{v}_B$, vektor $\vec{v}_n$ možemo dobiti na različite načine, i taj vektor nije jednoznačan. Naime, postoji beskonačno mnogo vektora koji su okomiti na zadanu ravninu; međutim, svi su međusobno kolinearni i razlika između njih je isključivo u njihovoj duljini (normi) te smjeru, gdje su moguća dva smjera. Jedan od načina dobivanja ovakvog vektora je i vektorski produkt. Možemo odabrati:

$$\vec{v}_n = \vec{v}_A \times \vec{v}_B. \tag{2.33}$$

Ako zapis ravnine pomoću njezine normale – izraz (2.31) – dalje razriješimo raspisivanjem skalarnog produkta, dobivamo:

$$(T_{R_1} - T_{S_1}) \cdot v_{n_1} + (T_{R_2} - T_{S_2}) \cdot v_{n_2} + (T_{R_3} - T_{S_3}) \cdot v_{n_3} = 0.$$

Nakon množenja i grupiranja slijedi:

$$v_{n_1} \cdot T_{R_1} + v_{n_2} \cdot T_{R_2} + v_{n_3} \cdot T_{R_3} - (T_{S_1} v_{n_1} + T_{S_2} v_{n_2} + T_{S_3} v_{n_3}) = 0 \tag{2.34}$$

ili:

$$A \cdot T_{R_1} + B \cdot T_{R_2} + C \cdot T_{R_3} + D = 0 \tag{2.35}$$

što je poznato kao *implicitni oblik* jednadžbe ravnine.

Uočimo odmah jedan važan detalj: koeficijenti uz komponente T_{R_1}, T_{R_2} i T_{R_3} direktno odgovaraju komponentama vektora normale ravnine. Ovo je zgodno svojstvo koje se često zaboravi. Dodatno, ako se prethodni izraz podijeli izrazom $\sqrt{A^2 + B^2 + C^2}$ – čime se vektor normale normira (svodi na vektor čija je norma jednaka 1) – apsolutna vrijednost slobodnog koeficijenta tada će odgovarati udaljenosti ravnine do ishodišta koordinatnog sustava (vidi sljedeći primjer).

Primjer: 6

Pokažite da tvrdnja iz prethodnog odlomka doista vrijedi: ako je vektor normale ravnine prisutan u implicitnoj jednadžbi ravnine normiran, tada apsolutna vrijednost slobodnog člana predstavlja udaljenost ishodišta do te ravnine.

Rješenje:

Krenut ćemo od implicitnog oblika jednadžbe ravnine, zapisane na uobičajeni način.

$$a \cdot x + b \cdot y + c \cdot z + d = 0$$

Normala ove ravnine je vektor $\vec{n} = [\ a \quad b \quad c\ ]$. Ako je taj vektor normiran, znači da mu je norma jednaka 1:

$$||\vec{n}|| = \sqrt{a^2 + b^2 + c^2} = 1 \Rightarrow a^2 + b^2 + c^2 = 1$$

Ova jednakost brzo će nam zatrebati. Sljedeći korak: kako se definira udaljenost proizvoljne točke od ravnine? Iz te točke spusti se okomica na ravninu. Udaljenost je duljina te okomice. Posljedica je interesantna: iz ishodišta se trebamo pomaknuti upravo u smjeru normale ravnine – samo što ne znamo koliko točno. Zapišimo ono što znamo:

$$T_R = [\ 0 \quad 0 \quad 0\ ] + k \cdot [\ a \quad b \quad c\ ] = [\ ka \quad kb \quad kc\ ]$$

Pri tome točka T_R leži u ravnini. Iz ishodišta smo se pomaknuli za vektor $k \cdot \vec{n}$, gdje je k u ovom trenutku još nepoznata konstanta koja skalira vektor normale. Također, ako točka T_R leži u ravnini, onda ona zadovoljava implicitni oblik jednadžbe ravnine. Uvrstimo stoga $x = ka$, $y = kb$ i $z = kc$:

$$\begin{aligned} a \cdot (ka) + b \cdot (kb) + c \cdot (kc) + d &= 0 \\ \Rightarrow ka^2 + kb^2 + kc^2 + d &= 0 \\ \Rightarrow k \cdot (a^2 + b^2 + c^2) + d &= 0 \end{aligned}$$

Iskoristimo sada činjenicu da je vektor normale normiran, odnosno da vrijedi: $a^2 + b^2 + c^2 = 1$:

$$k \cdot (a^2 + b^2 + c^2) + d = 0 \Rightarrow k \cdot 1 + d = 0 \Rightarrow k = -d.$$

I ovime smo praktički gotovi s tvrdnjom. Naime, iz ishodišta smo se do ravnine pomaknuli za vektor $k \cdot \vec{n}$, pa je upravo njegova norma tražena udaljenost ishodišta do ravnine. Norma tog vektora je:

$$\begin{aligned} ||k \cdot \vec{n}|| &= \sqrt{(k \cdot a)^2 + (k \cdot b)^2 + (k \cdot c)^2} \\ &= \sqrt{k^2 \cdot (a^2 + b^2 + c^2)} \\ &= \sqrt{(-d)^2 \cdot (1)} \\ &= \sqrt{d^2} \\ &= |d| \end{aligned}$$

Ako koeficijenti normale nisu normirani, dovoljno je podijeliti čitavu implicitnu jednadžbu s normom normale; time će slobodni koeficijent automatski odgovarati udaljenosti ishodišta od ravnine.

Za vježbu se uvjerite da vrijedi i sljedeća tvrdnja: ako su koeficijenti normale ravnine u jednadžbi ravnine normirani, tada se uvrštavanjem proizvoljne točke T u lijevu stranu jednadžbe ravnine, dakle u izraz $a \cdot x + b \cdot y + c \cdot z + d$, direktno dobiva udaljenost točke T od te ravnine (uz eventualnu korekciju predznaka dobivenog broja, jer je udaljenost po definiciji nenegativan broj). Dokaz se radi na sličan način kao što smo to dokazali za udaljenost ishodišta od ravnine – uvjerite se!

Ako u jednadžbu (2.35) ubacimo homogene koordinate, dobiva se:

$$A \cdot \frac{T_{Rh_1}}{h_R} + B \cdot \frac{T_{Rh_2}}{h_R} + C \cdot \frac{T_{Rh_3}}{h_R} + D = 0$$

ili nakon sređivanja:

$$A \cdot T_{Rh_1} + B \cdot T_{Rh_2} + C \cdot T_{Rh_3} + D \cdot h_R = 0. \tag{2.36}$$

Izraz se dalje može prikazati i matrično:

$$\begin{bmatrix} T_{Rh_1} & T_{Rh_2} & T_{Rh_3} & h_R \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = 0. \tag{2.37}$$

Ovaj oblik koristi se vrlo često i korisno ga je poznavati. No, bitan je iz još jednog razloga. Kod jednadžbe pravca na temelju slične relacije definirali smo

odnos točke i pravca. Pomoću ove jednadžbe definirat ćemo odnos točke i ravnine. Jednadžbu (2.37) simbolički možemo zapisati:

$$T_{Rh} \cdot \mathbf{R} = 0. \tag{2.38}$$

Dakle, za svaku točku koja je na ravnini jednadžba je zadovoljena. No ako uvrstimo za T_R neku točku koja ne pripada ravnini, gornja jednadžba neće biti zadovoljena. Na temelju tog rezultata definira se odnos točke i ravnine:

$$T_{Rh} \cdot \mathbf{R} \begin{cases} > 0, & T_{Rh} \text{ je iznad ravnine,} \\ = 0, & T_{Rh} \text{ je na ravnini,} \\ < 0, & T_{Rh} \text{ je ispod ravnine,} \end{cases} \tag{2.39}$$

uz pretpostavku da je h_R pozitivan.

Ovaj odnos moguće je pojasniti ako se prisjetimo da je jedan od načina definiranja ravnine bio upravo putem normale ravnine. Svaka ravnina može imati beskonačno mnogo normala. Pola ih je orijentirano u jednom smjeru, pola u drugom smjeru. Međutim, jednom kada napišemo jednadžbu ravnine (ili očitamo matricu $\mathbf{R}$), fiksirali smo smjer u kojem je normala okrenuta. Nakon što je to fiksirano, za sve točke koje ne pripadaju ravnini a do kojih se dolazi tako da se s ravnine pomičemo u smjeru normale kažemo da su iznad ravnine; za njih će umnožak $T_{Rh} \cdot \mathbf{R}$ biti pozitivan. S druge pak strane, za sve točke koje ne pripadaju ravnini a do kojih se dolazi tako da se s ravnine pomičemo u smjeru suprotnom od smjera normale umnožak $T_{Rh} \cdot \mathbf{R}$ će biti negativan. Za takve točke kažemo da su ispod ravnine.

Ispitivanje odnosa točke i ravnine bit će nam vrlo važno kod crtanja tijela u 3D prostoru, gdje će tijelo uobičajeno biti zadano dijelovima ravnina koje čine njegov plašt. Tamo će nas zanimati koje su nam površine vidljive a koje nisu, i do tih informacija djelomično ćemo dolaziti i temeljem ispitivanja odnosa točke i ravnine.

2.5.2 Jednadžba ravnine kroz tri točke

Svaka ravnina jednoznačno je određena s tri točke koje ne leže na istom pravcu. Da bismo odredili jednadžbu ravnine kroz tri točke, krenut ćemo od parametarskog oblika ravnine u homogenom prostoru:

$$T_{Rh} = \begin{bmatrix} T_{Rh_1} & T_{Rh_2} & T_{Rh_3} & h_R \end{bmatrix} = \begin{bmatrix} \lambda & \mu & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{A_1} & v_{A_2} & v_{A_3} & h_A \\ v_{B_1} & v_{B_2} & v_{B_3} & h_B \\ T_{S_1} & T_{S_2} & T_{S_3} & h_S \end{bmatrix}$$

što kraće možemo zapisati kao:

$$T_{Rh} = \begin{bmatrix} \lambda & \mu & 1 \end{bmatrix} \cdot \mathbf{G}$$

U ovom slučaju ravnina je određena s dva nekolinearna vektora i točkom. Pri tome su vektori i točke četverokomponentni jer radimo u tri dimenzije (3 komponente) i u homogenom prostoru (još jedna komponenta).

Kao i kod određivanja jednadžbe pravca, i ovdje možemo raditi na isti način. Ravnina će kroz točku T_A proći za neki λ i neki μ, kroz točku T_B će proći za neki drugi λ i neki drugi μ, i kroz točku T_C će proći za neki treći λ i neki treći μ. Idemo stoga odabrati za koje će se to λ i μ dogoditi. Neka ravnina prolazi kroz T_A za $\lambda = 0$ i $\mu = 0$, kroz T_B za $\lambda = 1$ i $\mu = 0$ te kroz T_C za $\lambda = 1$ i $\mu = 1$. Tada vrijedi:

$$T_{Ah} = \begin{bmatrix} T_{Ah_1} & T_{Ah_2} & T_{Ah_3} & h_A \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{G}$$

$$T_{Bh} = \begin{bmatrix} T_{Bh_1} & T_{Bh_2} & T_{Bh_3} & h_B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \cdot \mathbf{G}$$

$$T_{Ch} = \begin{bmatrix} T_{Ch_1} & T_{Ch_2} & T_{Ch_3} & h_C \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \cdot \mathbf{G}$$

što nakon stapanja u jednu matričnu jednadžbu daje:

$$\begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \cdot \mathbf{G}.$$

Sređivanjem ovog izraza dobivamo:

$$\mathbf{G} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} T_{Bh} - T_{Ah} \\ T_{Ch} - T_{Bh} \\ T_{Ah} \end{bmatrix}. \tag{2.40}$$

2.6 Dodatni često korišteni pojmovi

2.6.1 Baricentrične koordinate

Baricentrične koordinate često su korištene u računalnoj grafici pri radu s trokutima. Trokut je dio ravnine određen trima točkama koje zovemo vrhovima trokuta. Označimo ih A, B i C. Prisjetimo se: tri točke koje ne leže na istom pravcu u prostoru određuju ravninu, i u toj ravnini leži trokut ABC. Također, već znamo da svaku točku ravnine možemo zapisati kao linearnu kombinaciju dvaju nekolinearnih vektora koji leže u toj ravnini, i koja "kreće" iz neke proizvoljne fiksne točke u ravnini. Napravimo sada jedan specifični odabir: neka fiksna točka bude sam vrh A, neka prvi vektor bude onaj razapet između vrhova B i A: $\vec{v}_1 = \vec{B} - \vec{A}$, te neka drugi vektor bude onaj razapet između vrhova C i A: $\vec{v}_2 = \vec{C} - \vec{A}$. Tada se proizvoljna točka T koja leži u toj ravnini može zapisati kao:

$$\vec{T} = \vec{A} + \lambda \cdot (\vec{B} - \vec{A}) + \mu \cdot (\vec{C} - \vec{A}).$$

Krenuvši od ovog izraza, možemo to dalje raspisati i svesti na drugačiji oblik:

$$\begin{aligned}\vec{T} &= \vec{A} + \lambda \cdot \vec{B} - \lambda \cdot \vec{A} + \mu \cdot \vec{C} - \mu \cdot \vec{A} \\ &= (1 - \lambda - \mu)\vec{A} + \lambda \cdot \vec{B} + \mu \cdot \vec{C}\end{aligned}$$

čime dolazimo do konačnog zapisa:

$$\vec{T} = (1 - \lambda - \mu)\vec{A} + \lambda \cdot \vec{B} + \mu \cdot \vec{C}. \tag{2.41}$$

Ako parametre uz svaku točku zamijenimo novom oznakom (koristit ćemo oznake t_i), dobivamo izraz:

$$\vec{T} = t_1\vec{A} + t_2 \cdot \vec{B} + t_3 \cdot \vec{C}, \tag{2.42}$$

gdje su (t_1, t_2, t_3) *baricentrične koordinate* točke T. Uočimo dakle: baricentrične koordinate su drugačiji način za prikaz točaka koje leže u istoj ravnini. Kada koristimo baricentrične koordinate, moramo znati *s obzirom na koje vrhove* su te koordinate definirane. Baricentrične koordinate postoje za svaku točku ravnine koju definiraju tri zadana vrha trokuta. Međutim, uporabom baricentričnih koordinata na jednostavan ćemo način moći utvrditi u kakvom je odnosu bilo koja promatrana točka ravnine i zadani trokut. Prije no što pokažemo, uočimo i da vrijedi sljedeća jednakost:

$$t_1 + t_2 + t_3 = 1. \tag{2.43}$$

Naime, ako iskoristimo (2.41), i uočimo da vrijedi: $t_1 = 1 - \lambda - \mu$, $t_2 = \lambda$ i $t_3 = \mu$, dokaz je trivijalan.

$$t_1 + t_2 + t_3 = (1 - \lambda - \mu) + \lambda + \mu = 1.$$

2.6.2 Izračun baricentričnih koordinata

Baricentrične koordinate možemo izračunati na nekoliko načina. Prvi način jest direktno uporabom izraza (2.42). Naime, taj izraz dovoljan je da napišemo sustav od tri jednadžbe s tri nepoznanice (u slučaju 3D točaka), kao što je ilustrirano u primjeru u nastavku.

Primjer: 7

Trokut je zadan vrhovima $A = (1, 1, 0)$, $B = (6, 11, 2)$ i $C = (11, 1, 0)$. Izračunati baricentrične koordinate točke $T = (6, 6, 1)$ primjenom izraza (2.42).

Rješenje:

Prema (2.42) vrijedi:

$$\vec{T} = t_1\vec{A} + t_2 \cdot \vec{B} + t_3 \cdot \vec{C}$$

Vektore ćemo prikazati kao troretčane jednostupčaste matrice, pa možemo pisati:

$$\begin{bmatrix} 6 \\ 6 \\ 1 \end{bmatrix} = t_1 \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + t_2 \cdot \begin{bmatrix} 6 \\ 11 \\ 2 \end{bmatrix} + t_3 \cdot \begin{bmatrix} 11 \\ 1 \\ 0 \end{bmatrix}$$

Ovo je sustav od tri jednadžbe s tri nepoznanice:

$$\begin{aligned} 6 &= 1 \cdot t_1 + 6 \cdot t_2 + 11 \cdot t_3 \\ 6 &= 1 \cdot t_1 + 11 \cdot t_2 + 1 \cdot t_3 \\ 1 &= 0 \cdot t_1 + 2 \cdot t_2 + 0 \cdot t_3 \end{aligned}$$

Rješavanjem dobivamo $t_1 = \frac{1}{4}$, $t_2 = \frac{1}{2}$ i $t_3 = \frac{1}{4}$, pa su baricentrične koordinate točke T upravo $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$.

Drugi način jest uočiti da sve tri baricentrične koordinate nisu potpuno slobodne (vidi izraz (2.43)), i posegnuti za matričnim računom. Naime, koristeći izraze (2.42) i (2.43) za proizvoljnu točku ravnine T možemo pisati:

$$\begin{aligned} \vec{T} &= t_1 \cdot \vec{A} + t_2 \cdot \vec{B} + t_3 \cdot \vec{C} \\ &= (1 - t_2 - t_3) \cdot \vec{A} + t_2 \cdot \vec{B} + t_3 \cdot \vec{C} \\ &= \vec{A} + t_2 \cdot (\vec{B} - \vec{A}) + t_3 \cdot (\vec{C} - \vec{A}) \end{aligned}$$

Sređivanjem ovog izraza dobivamo:

$$t_2 \cdot (\vec{B} - \vec{A}) + t_3 \cdot (\vec{C} - \vec{A}) = \vec{T} - \vec{A}$$

U ovom izrazu imamo dvije nepoznanice: t_2 i t_3, i tri jednadžbe. Zanemarimo stoga na tren treću jednadžbu, i promatrajmo samo prve dvije. Možemo pisati:

$$\begin{aligned} t_2 \cdot (B_x - A_x) + t_3 \cdot (C_x - A_x) &= T_x - A_x \\ t_2 \cdot (B_y - A_y) + t_3 \cdot (C_y - A_y) &= T_y - A_y \end{aligned}$$

Ovo možemo zapisati matrično:

$$\begin{bmatrix} B_x - A_x & C_x - A_x \\ B_y - A_y & C_y - A_y \end{bmatrix} \cdot \begin{bmatrix} t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} T_x - A_x \\ T_y - A_y \end{bmatrix}$$

Označimo li lijevu matricu kao $\mathbf{M}$, možemo pisati:

$$\mathbf{M} \cdot \begin{bmatrix} t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} T_x - A_x \\ T_y - A_y \end{bmatrix}$$

Množenjem tog izraza s lijeva inverznom matricom od $\mathbf{M}$ direktno dobivamo tražene baricentrične koordinate:

$$\begin{bmatrix} t_2 \\ t_3 \end{bmatrix} = \mathbf{M}^{-1} \cdot \begin{bmatrix} T_x - A_x \\ T_y - A_y \end{bmatrix} \tag{2.44}$$

Preostalu baricentričnu koordinatu dobit ćemo preko izraza (2.43). Uočimo da se u ovom slučaju radi o matrici $\mathbf{M}$ dimenzija 2×2, čiji je inverz trivijalno pronaći. Također, kako su stupci te matrice upravo vektori između vrhova trokuta, matrica je sigurno invertibilna, budući da su ti vektori nekolinearni.

Najjednostavniji način brzog pronalaska matričnog inverza matrice dimenzija 2×2 jest uporabom matrice kofaktora $\mathbf{C}$, te jednakosti:

$$\mathbf{A}^{-1} = \frac{1}{det\mathbf{A}} \cdot C^T \tag{2.45}$$

gdje je $\mathbf{A}$ matrica koju želimo invertirati, $\mathbf{C}$ njezina matrica kofaktora a $\mathbf{C}^T$ transponirana matrica kofaktora. U slučaju da se radi o kvadratnoj matrici dimenzija 2×2, izraz glasi:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{a \cdot d - b \cdot c} \cdot \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

Primjer: 8

Trokut je zadan vrhovima $A = (1, 1, 0)$, $B = (6, 11, 2)$ i $C = (11, 1, 0)$. Izračunati baricentrične koordinate točke $T = (6, 6, 1)$ primjenom izraza (2.44).

Rješenje:

Matrica $\mathbf{M}$ definirana je kao dvoretčana matrica čiji su stupci $\vec{B} - \vec{A}$ i $\vec{C} - \vec{A}$. Ona dakle glasi:

$$\mathbf{M} = \begin{bmatrix} 6-1 & 11-1 \\ 11-1 & 1-1 \end{bmatrix} = \begin{bmatrix} 5 & 10 \\ 10 & 0 \end{bmatrix}$$

Idemo izračunati njezin inverz, koristeći izraz (2.45). Determinantu lagano izračunamo:

$$det\mathbf{M} = 5 \cdot 0 - 10 \cdot 10 = -100$$

Matrica kofaktora matrice $\mathbf{M}$ je matrica (označimo je s $\mathbf{C}$) jednakih dimenzija kao i $\mathbf{M}$. Pri tome se element na poziciji $c_{i,j}$ dobije tako da se u originalnoj matrici $\mathbf{M}$ izbriše i-ti redak i j-ti stupac, pa se od tako dobivene matrice koja je za jednu dimenziju manja izračuna determinanta. Ako je $i + j$ parno, $c_{i,j}$ jednak je tako izračunatoj determinanti. Ako je $i + j$ neparno, $c_{i,j}$ jednak je vrijednosti izračunate determinante uz promjenu predznaka. Zvuči komplicirano, ali za matricu dimenzija 2×2 postupak je izuzetno jednostavan - brisanjem jednog retka i stupca ostaje matrica dimenzija 1×1, čija je determinanta upravo jednaka jedinom elementu te matrice.

Izračunajmo elemente matrice kofaktora. Pogledajmo naprije element $c_{1,1}$. Brisanjem prvog retka i prvog stupca matrice $\mathbf{M}$ ostaje nam matrica [0] čija je determinanta 0, pa je $c_{1,1} = 0$. Pogledajmo element $c_{2,1}$. Brisanjem drugog retka i prvog stupca matrice $\mathbf{M}$ ostaje nam matrica [10] čija je determinanta 10. Kako je $2+1 = 3$ neparno, uzimamo vrijednost determinante uz promijenjen predznak, pa je $c_{2,1} = -10$. Na taj način dobivamo matricu kofaktora:

$$\mathbf{C} = \begin{bmatrix} 0 & -10 \\ -10 & 5 \end{bmatrix}$$

Transponirana matrica kofaktora tada je:

$$\mathbf{C}^T = \begin{bmatrix} 0 & -10 \\ -10 & 5 \end{bmatrix}$$

Prema (2.45) slijedi da je:

$$\mathbf{M}^{-1} = \frac{1}{det\mathbf{M}} \cdot C^T = \frac{1}{-100} \cdot \begin{bmatrix} 0 & -10 \\ -10 & 5 \end{bmatrix}$$

Uvrštavanjem u izraz (2.44) slijedi:

$$\begin{bmatrix} t_2 \\ t_3 \end{bmatrix} = \frac{1}{det\mathbf{M}} \cdot C^T \cdot \begin{bmatrix} T_x - A_x \\ T_y - A_y \end{bmatrix} = \frac{1}{-100} \cdot \begin{bmatrix} 0 & -10 \\ -10 & 5 \end{bmatrix} \cdot \begin{bmatrix} 6-1 \\ 6-1 \end{bmatrix} = \frac{1}{-100} \cdot \begin{bmatrix} 0 & -10 \\ -10 & 5 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

Od tuda direktno čitamo:

$$t_2 = \frac{1}{-100} \cdot (0 \cdot 5 + -10 \cdot 5) = \frac{1}{-100} \cdot (-50) = \frac{1}{2}$$

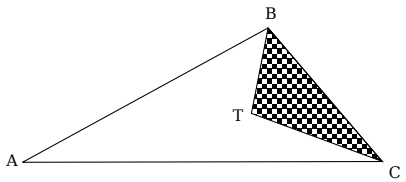
$$t_3 = \frac{1}{-100} \cdot (-10 \cdot 5 + 5 \cdot 5) = \frac{1}{-100} \cdot (-25) = \frac{1}{4}$$

Preostala baricentrična koordinata tada je:

$$t_1 = 1 - t_2 - t_3 = 1 - \frac{1}{2} - \frac{1}{4} = \frac{1}{4}.$$

Tražene baricentrične koordinate su: $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$.

Spomenimo još jedan način kako možemo izračunati baricentrične koordinate. Pretpostavimo da nas zanimaju baricentrične koordinate točke T koja se nalazi u trokutu ABC. Da bismo izračunali vrijednost baricentrične koordinate t_1 (koja je uz vrh A), promotrimo trokut koji točka T definira s preostalim vrhovima trokuta (dakle, B i C). Slika 2.8 ilustrira ovu situaciju. Omjer površine trokuta TBC i ABC odgovara baricentričnoj koordinati t_1. Na jednak način definiraju se i ostale baricentrične koordinate.



Slika 2.8: Baricentrične koordinate preko omjera površina trokuta

Možda nije odmah očito kako jednostavno doći do površine promatranih trokuta. Međutim, prisjetimo se samo svojstava vektorskog produkta. Norma vektorskog produkta odgovara površini paralelograma što ga razapinju promatrani vektori, što je točno jednako dvostrukoj površini trokuta što ga razapinju ti vektori. Dakle, površinu trokuta ABC dobit ćemo kao:

$$P(ABC) = \frac{\|(\vec{B} - \vec{A}) \times (\vec{C} - \vec{A})\|}{2}$$

Površina trokuta TBC po istom principu je:

$$P(TBC) = \frac{\|(\vec{B} - \vec{T}) \times (\vec{C} - \vec{T})\|}{2}$$

Njihov omjer odgovara baricentričnoj koordinati t_1:

$$t_1 = \frac{P(TBC)}{P(ABC)} = \frac{\frac{\|(\vec{B}-\vec{T})\times(\vec{C}-\vec{T})\|}{2}}{\frac{\|(\vec{B}-\vec{A})\times(\vec{C}-\vec{A})\|}{2}} = \frac{\|(\vec{B}-\vec{T})\times(\vec{C}-\vec{T})\|}{\|(\vec{B}-\vec{A})\times(\vec{C}-\vec{A})\|} \tag{2.46}$$

Na sličan način dobiju se i izrazi za preostale baricentrične koordinate:

$$t_2 = \frac{P(TAC)}{P(ABC)} = \frac{\|(\vec{A}-\vec{T})\times(\vec{C}-\vec{T})\|}{\|(\vec{B}-\vec{A})\times(\vec{C}-\vec{A})\|} \tag{2.47}$$

$$t_3 = \frac{P(TAB)}{P(ABC)} = \frac{\|(\vec{A}-\vec{T})\times(\vec{B}-\vec{T})\|}{\|(\vec{B}-\vec{A})\times(\vec{C}-\vec{A})\|} \tag{2.48}$$

Primjer: 9

Trokut je zadan vrhovima $A = (1, 1, 0)$, $B = (6, 11, 2)$ i $C = (11, 1, 0)$. Izračunati baricentrične koordinate točke $T = (6, 6, 1)$ primjenom omjera površina.

Rješenje:

Trebat će nam vektori $\vec{T} - \vec{A}$, $\vec{B} - \vec{A}$ i $\vec{C} - \vec{A}$. Izračunajmo ih.

$$\vec{T} - \vec{A} = (6, 6, 1) - (1, 1, 0) = (5, 5, 1)$$

$$\vec{B} - \vec{A} = (6, 11, 2) - (1, 1, 0) = (5, 10, 2)$$

$$\vec{C} - \vec{A} = (11, 1, 0) - (1, 1, 0) = (10, 0, 0)$$

Vektorski produkt $(\vec{B} - \vec{A}) \times (\vec{C} - \vec{A})$ jednak je $(0, 20, -100)$ pa je njegova norma $\sqrt{400 + 10000} = \sqrt{10400}$, što odgovara dvostrukoj površini trokuta ABC. Vektorski produkt $(\vec{T} - \vec{A}) \times (\vec{C} - \vec{A})$ jednak je $(0, 10, -50)$ pa je njegova norma $\sqrt{100 + 2500} = \sqrt{2600}$, što odgovara dvostrukoj površini trokuta ATC.

$$t_2 = \frac{\sqrt{2600}}{\sqrt{10400}} = \sqrt{\frac{2600}{10400}} = \sqrt{0.25} = 0.5 = \frac{1}{2}$$

Vektorski produkt $(\vec{T} - \vec{A}) \times (\vec{B} - \vec{A})$ jednak je $(0, -5, 25)$ pa je njegova norma $\sqrt{25 + 625} = \sqrt{650}$, što odgovara dvostrukoj površini trokuta ABT.

$$t_3 = \frac{\sqrt{650}}{\sqrt{10400}} = \sqrt{\frac{650}{10400}} = \sqrt{0.0625} = 0.25 = \frac{1}{4}$$

Sada još možemo izračunati t_1 iz jednakosti $t_1 + t_2 + t_3 = 1$ što daje:

$$t_1 = 1 - t_2 - t_3 = 1 - \frac{1}{2} - \frac{1}{4} = \frac{1}{4}$$

Baricentrične koordinate točke T su dakle $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$.

2.6.3 Odnos trokuta i točke preko baricentričnih koordinata

Baricentrične koordinate mogu nam poslužiti kako bismo utvrdili nalazi li se proizvoljna točka T ravnine unutar trokuta, na rubu trokuta ili pak izvan trokuta. Vrijedi sljedeće. Neka točka T ima baricentrične koordinate (t_1, t_2, t_3).

$$\begin{cases} \text{ako } \forall i(t_i \in (0,1)) & \Rightarrow T \text{ je unutar trokuta,} \\ \text{ako } \forall i(t_i \in [0,1]) \wedge \exists j(t_j = 1) & \Rightarrow T \text{ je vrh trokuta,} \\ \text{ako } \forall i(t_i \in [0,1]) \wedge \exists j(t_j = 0) \wedge \nexists j(t_j = 1) & \Rightarrow T \text{ je na rubu trokuta,} \\ \text{ako } \exists i(t_i \notin [0,1]) & \Rightarrow T \text{ je izvan trokuta.} \end{cases} \tag{2.49}$$

Prva tri slučaja razmatraju situacije u kojima niti jedna baricentrična koordinata nije manja od 0 niti veća od 1. Prvi slučaj nam govori da ako su sve baricentrične koordinate strogo između 0 i 1 (0 i 1 su isključeni iz intervala), točka se nalazi unutar trokuta. Drugi slučaj razmatra situaciju u kojoj je jedna baricentrična koordinata upravo 1 (a sve ostale moraju biti 0); tada točka odgovara vrhu trokuta (onom uz koji je baricentrična koordinata iznosa 1). Treći slučaj razmatra situaciju u kojoj se točka nalazi na jednom od bridova trokuta ali ne u vrhu: u tom slučaju nužno je, da je jedna baricentrača koordinata jednaka 0 dok ostale ne smiju biti 0. Četvrti slučaj razmatra situaciju u kojoj postoji baricentrična koordinata čiji je iznos manji od 0 ili veći od jedan. U tom slučaju točka je izvan trokuta.

Primjer: 10

Trokut je zadan vrhovima $A = (1,1,0)$, $B = (6,11,2)$ i $C = (11,1,0)$. Odrediti položaj točaka $T_1 = (6,6,1)$ i $T_2 = (8.5, 8.5, 1.5)$ uporabom baricentričnih koordinata.

Rješenje:

Baricentrične koordinate točke T_1 već smo odredili: $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$. Kako su sve baricentrične koordinate unutar intervala $(0,1)$, točka se nalazi unutar trokuta.

Baricentrične koordinate točke T_2 su: $(-\frac{1}{8}, \frac{3}{8}, \frac{3}{4})$ (izračunajte ih za vježbu). Kako je koordinata t_1 manja od 0, točka se nalazi izvan trokuta.

2.7 Česti zadatci

U nastavku ove knjige često ćemo se susretati s nekim tipičnim problemima poput pronalaska sjecišta pravca i ravnine, pravca i sfere, razmatranjem prolazi li pravac kroz trokut u 3D prostoru i sl. Samo neki od slučajeva gdje će nam trebati odgovori na ova pitanja su algoritmi bacanja zrake (engl. *Ray Casting*) i praćenja zrake (engl. *Ray Tracing*). Stoga ćemo se ovdje pozabaviti rješavanjem upravo tih problema.

Pretpostavit ćemo da je pravac zadan točkama T_S i T_E. Pri tome će se promatrač nalaziti u točki T_S i gledat će prema točki T_E. Takav pravac u 3D

prostoru zapisivat ćemo u vektorskom parametarskom obliku:

$$\vec{t}(\lambda) = \vec{T_S} + \lambda \cdot \vec{d}$$

pri čemu je $\vec{t}(\lambda)$ točka na pravcu a $\vec{d}$ predstavlja normirani (jedinični) vektor pravca:

$$\vec{d} = \frac{\vec{T_E} - \vec{T_S}}{||\vec{T_E} - \vec{T_S}||}.$$

Držimo li se analogije promatrača koji se nalazi u T_S i gleda u smjeru T_E, tada su sve točke $\vec{t}(\lambda)$ takve da je $\lambda > 0$ ispred promatrača, a sve točke $\vec{t}(\lambda)$ takve da je $\lambda < 0$ iza promatrača (pa ih promatrač ne vidi).

2.7.1 Probodište pravca i ravnine

Neka je ravnina zadana u implicitnom obliku: $A \cdot x + B \cdot y + C \cdot z + D = 0$. Prisjetimo li se da koeficijenti A, B i C zapravo predstavljaju koeficijente normale ravnine, jednadžbu možemo zapisati i ovako: $\vec{n} \cdot \vec{t} + D = 0$, gdje je $\vec{n}$ normala ravnine a $\vec{t}$ točka koja leži u ravnini, i čije su komponente (x, y, z). Uvrštavanjem parametarskog oblika jednadžbe pravca umjesto $\vec{t}$ slijedi:

$$\vec{n} \cdot \vec{t} + D = 0$$

$$\vec{n} \cdot (\vec{T_S} + \lambda \cdot \vec{d}) + D = 0$$

$$\vec{n} \cdot \vec{T_S} + \lambda \cdot \vec{n} \cdot \vec{d} + D = 0$$

$$\lambda = \frac{-(\vec{n} \cdot \vec{T_S} + D)}{\vec{n} \cdot \vec{d}}$$

Ako je $\vec{n} \cdot \vec{d} = 0$, normala ravnine i pravac su okomiti, što znači da je pravac paralelan s ravninom. Postoje dvije mogućnosti - ili pravac leži u ravnini, pa ima beskonačno sjecišta (svaka točka je jedno sjecište), ili pravac ne leži u ravnini a kako je paralelan, nema niti jedno sjecište. U praksi ćemo slučaj kada je $\vec{n} \cdot \vec{d} = 0$ najčešće zanemariti (odnosno, tada ćemo reći da pravac i ravnina nemaju probodište).

Slučaj kada je $\vec{n} \cdot \vec{d} \neq 0$ je slučaj koji će nas zanimati. Ako je dodatno $\lambda > 0$, postoji probodište ispred promatrača, a dobit ćemo ga tako da izračunati λ uvrstimo u parametarski oblik jednadžbe pravca. Ako je dodatno $\lambda < 0$, postoji probodište no kako je ono iza promatrača, slučaj nas neće zanimati. Ako je $\lambda = 0$, to znači da je početna točka već u ravnini.

2.7.2 Probodište pravca i sfere

Sfera radijusa r i centra $\vec{C}$ definirana je jednadžbom $(x-C_x)^2+(y-C_y)^2+(z-C_z)^2=r^2$. Poslužimo se opet istim oznakama kao u prethodnoj sekciji; neka je $\vec{t}$ točka koja leži na sferi, $\vec{t}=(x,y,z)$. Prethodnu jednadžbu tada možemo pisati i ovako: $(\vec{t}-\vec{C})\cdot(\vec{t}-\vec{C})=r^2$. Uvrstimo sada u jednadžbu parametarski oblik jednadžbe pravca:

$$(\vec{t}-\vec{C})\cdot(\vec{t}-\vec{C})=r^2$$
$$(\vec{T_S}+\lambda\cdot\vec{d}-\vec{C})\cdot(\vec{T_S}+\lambda\cdot\vec{d}-\vec{C})=r^2$$
$$(\lambda\cdot\vec{d}+\vec{T_S}-\vec{C})\cdot(\lambda\cdot\vec{d}+\vec{T_S}-\vec{C})=r^2$$
$$\lambda^2\vec{d}\cdot\vec{d}+2\lambda\vec{d}\cdot(\vec{T_S}-\vec{C})+(\vec{T_S}-\vec{C})\cdot(\vec{T_S}-\vec{C})-r^2=0.$$

Uočimo da smo dobili kvadratnu jednadžbu oblika $a\lambda^2+b\lambda+c=0$, uz:

$$a=\vec{d}\cdot\vec{d}=1$$
$$b=2\vec{d}\cdot(\vec{T_S}-\vec{C})$$
$$c=(\vec{T_S}-\vec{C})\cdot(\vec{T_S}-\vec{C})-r^2.$$

Koeficijent a je jedan jer smo prethodno tražili da vektor $\vec{d}$ bude normiran; skalarni produkt tog vektora sa samim sobom jednak kvadratu norme, no to u tom slučaju upravo 1. Rješenja jednadžbe su:

$$\lambda_{1,2}=\frac{-b\pm\sqrt{b^2-4ac}}{2a}\quad\Rightarrow\quad\lambda_{1,2}=\frac{-b\pm\sqrt{b^2-4c}}{2}.$$

Sada imamo sljedeće mogućnosti.

- λ_1 i λ_2 su dva različita realna broja i oba su veća od 0. Neka je $\lambda_1<\lambda_2$ (ako ovo ne vrijedi, zamijenite vrijednosti). Postoje dva sjecišta sa sferom i oba su ispred promatrača. λ_1 određuje bliže sjecište a λ_2 dalje sjecište (ako je sfera neprozirna, to se sjecište onda ne vidi). Konkretne točke dobivamo uvrštavanjem λ_1 i λ_2 u parametarski oblik jednadžbe pravca.

- λ_1 i λ_2 su dva različita realna broja, jedan je pozitivan, drugi nije. Neka je $\lambda_1<=0$ i $\lambda_2>0$ (ako ovo ne vrijedi, zamijenite vrijednosti). Postoje dva sjecišta sa sferom. Jedno je iza promatrača (određeno s λ_1) a drugo je ispred promatrača (određeno s λ_2). Promatrač se, dakle, nalazi u sferi.

- $\lambda_1=\lambda_2=\lambda$ i to je realan broj. Pravac je tangenta na sferu i postoji samo jedna točka dodira. Ako je $\lambda>0$, točka je ispred promatrača, ako je $\lambda<0$ točka je iza promatrača.

- λ_1 i λ_2 su dva različita kompleksna broja. U tom slučaju pravac i sfera nemaju sjecišta.

- U slučaju da je neki od lambda jednak 0, radi se o posebnom slučaju gdje je početna točka upravo na plaštu kugle; takvi slučajevi nam nisu zanimljivi.

2.7.3 Probodište pravca i trokuta

Neka je trokut zadan u prostoru s tri točke A, B i C. Zanima nas probada li pravac taj trokut. Problem ćemo rastaviti na dva jednostavnija problema.

Prvi problem je pronalazak probodišta ravnine u kojoj leži trokut i pravca. Prisjetimo se, ako s $\vec{t}$ označimo točku ravnine, a s $\vec{n}$ normalu ravnine, implicitni oblik jednadžbe ravnine možemo pisati kao $\vec{n} \cdot \vec{t} + D = 0$. Normalu ćemo odrediti kao vektorski produkt između vektora koje razapinju točke A i B te A i C: $\vec{n} = (\vec{B} - \vec{A}) \times (\vec{C} - \vec{A})$. Jednom kada znamo $\vec{n}$, u implicitnom obliku jednadžbe pravca jedina je nepoznanica D, no za njega tada vrijedi: $D = -\vec{n} \cdot \vec{t}$. Kako i A i B i C leže u ravnini, možemo uzeti bilo koju od tih točaka kako bismo dobili D; npr. $D = -\vec{n} \cdot \vec{A}$.

Jednom kada znamo implicitni oblik jednadžbe ravnine, odredit ćemo sjecište (već smo opisali kako). Ako sjecište ne postoji, znamo da pravac ne probada trokut i gotovi smo. U suprotnom, neka je to sjecište točka t. Ostaje nam odrediti nalazi li se točka t unutar trokuta ili ne. Jedan od jednostavnijih načina jest izračunati baricentrične kooridnate točke t s obzirom na trokut A, B i C (i to smo već obradili – napravite to primjerice omjerom odgovarajućih površina trokuta koje ćete izračunati kao polovine normi odgovarajućih vektorskih produkata). Neka su baricentrične koordinate t_1, t_2 i t_3. Točka se nalazi unutar trokuta ako je $t_1 > 0$ i $t_2 > 0$ i $t_3 > 0$. Primjetite da zbog uvjeta $t_1 + t_2 + t_3 = 1$ slijedi da sve tri koordinate moraju biti iz intervala $(0, 1)$. U tom slučaju probodište trokuta i pravca postoji, i to je upravo točka t.

2.7.4 Probodište pravca i poligona

Neka je zadan poligon u prostoru; pretpostavit ćemo da su vrhovi poligona doista zadani tako da svi leže u istoj ravnini. Problem određivanja probodišta pravca i poligona rješava se u dva koraka.

1. Potrebno je odrediti ravninu u kojoj leži poligon i potom probodište pravca i te ravnine. Označimo to probodište s $\vec{T}$.

2. Ako probodište u prethodnom koraku postoji, potrebno je odrediti nalazi li se to probodište unutar poligona ili izvan poligona. Ako je izvan, probodište pravca i poligona ne postoji; u suprotnom, probodište je točka $\vec{T}$.

Ako je poligon zadan u 3 dimenzije, da bismo mogli primjeniti klasične algoritme za utvrđivanje odnosa točke i poligona, potrebno ga je projicirati u dvije dimenzije. Jedna je mogućnost naprosto odbaciti jednu od koordinata, čime se poligon projicira u ravninu koju razapinju preostale dvije koordinatne osi. U slučaju da je ravnina u kojoj leži poligon takva da u njoj leži i koordinatna os koju smo projekcijom htjeli odbaciti, poligon će degradirati u linijski segment

što će onemogućiti ispravan rad algoritma. Stoga je općenitije rješenje pronaći parametarski oblik ravnine u kojoj leži poligon (neka je to oblik koji koristi dva vektora $\vec{u}$ i $\vec{v}$), i potom sve točke poligona zapisati u obliku $\vec{T_i} = \vec{S} + \lambda\vec{u} + \mu\vec{v}$ gdje je $\vec{S}$ bilo koja točka koja leži u ravnini i od koje je razapet dvodimenzionalni koordinatni sustav određen osima $\vec{u}$ i $\vec{v}$. U tom koordinatnom sustavu točka $\vec{T_i}$ ima koordinate (λ, μ), i nad tako transformiranim točkama moguće je primijeniti uobičajene algoritme. Više o ovim algoritmima bit će riječi u poglavlju 4.

2.8 Ponavljanje

1. Kako se računa reflektirani vektor?
2. Napišite jednadžbu pravca u 3D prostoru u parametarskom obliku.
3. Je li u parametarskom obliku jednadžbe pravca u 3D prostoru vektor pravca jedinstven? Objasnite.
4. Kod parametarskog oblika jednadžbe pravca u 3D prostoru vektor pravca određen je razlikom krajnje i početne točke nekog segmenta. Objasnite što nam tada za neku promatranu točku govori parametar λ koji pripada toj točki?
5. Zašto se uvodi homogeni prostor? Kako se točke iz radnog preslikavaju u homogeni prostor? Kako se točke iz homogenog prostora preslikavaju u radni prostor? Je li zapis točke iz radnog prostora u homogenom prostoru jedinstven? Objasnite zašto.
6. Objasnite na koji se način definira odnos točke i pravca u 2D prostoru.
7. Napišite parametarski oblik jednadžbe ravnine u 3D prostoru, i to u vektorskom obliku i u matričnom obliku.
8. Kako se iz parmetarskog oblika jednadžbe ravnine dolazi do implicitnog oblika jednadžbe ravnine?
9. U kakvoj su vezi vektor normale ravnine te implicitni oblik jednadžbe ravnine?
10. Objasnite na koji se način definira odnos točke i ravnine u 3D prostoru.
11. Kod definiranja odnosa točke i ravnine u 3D prostoru, u kakvoj su vezi smjer normale ravnine i klasifikacija "točka je iznad" odnosno "točka je ispod" ravnine?
12. Kako se računa skalarni produkt dvaju vektora? Navedite svojstva skalarnog produkta.

13. Kako se računa vektorski produkt dvaju vektora? Koja je njegova interpretacija? Navedite svojstva vektorskog produkta.
14. Kako uporabom vektorskog produkta možemo izračunati površinu trokuta?
15. Kako možete provjeriti jesu li dva pravca u 3D prostoru okomiti?
16. Kako možete provjeriti jesu li dva pravca u 3D prostoru paralelni?
17. Kako možete provjeriti sijeku li se dva pravca u 3D prostoru ili su mimoilazni?
18. Objasnite kako se dolazi do baricentričnih koordinata.
19. Objasnite kako za neku točku možemo izračunati njezine baricentrične koordinate rješavanjem sustava jednadžbi.
20. Objasnite kako za neku točku možemo izračunati njezine baricentrične koordinate uporabom omjera površina trokuta.
21. Pokažite kako se računa probodište pravca i ravnine.
22. Pokažite kako se računa probodište pravca i sfere.
23. Pokažite kako se računa probodište pravca i trokuta.

Poglavlje 3

Interpolacije

3.1 Uvod

U računalnoj grafici često se koriste interpolacije. Primjerice, pri animaciji nekog objekta želimo brzinu kojom se objekt kreće smanjiti od početne do konačne u nekoliko koraka, ili pak želimo objekt pomaknuti od jedne pozicije do druge pozicije ali tako da se dobije dojam kontinuiranog kretanja i slično. U svim tim situacijama imamo zadanu početnu i konačnu vrijednost, a naš je zadatak pronaći niz međuvrijednosti.

Ovisno o potrebama, interpolaciju možemo raditi na različite načine. U ovom poglavlju malo detaljnije ćemo se pozabaviti najčešće korištenim načinima.

3.2 Linearna interpolacija

Zamislimo sljedeći scenarij: objekt se giba uzduž osi x. Njegova početna pozicija je $x = x_0 = 1$ a konačna $x = x_1 = 6$. U zadani raspon koordinata potrebno je "umetnuti" još nekoliko međutočaka (primjerice, još 4), kako bismo objekt mogli pomicati u manjim koracima. Jedan od načina kako to možemo napraviti je očit i bez previše računanja: dodat ćemo još pozicije 2, 3, 4 i 5. No što da smo htjeli dodati, recimo, 5 međutočaka, ili njih više?

Linearna interpolacija daje nam jedan mogući recept kako to napraviti. Uvedimo parametar t koji poprima vrijednosti iz intervala $[0, 1]$. Neka je s $x(t)$ označena interpolirana vrijednost: za $t = 0$ očekujemo $x(0) = x_0 = 1$ dok za $t = 1$ očekujemo $x(1) = x_1 = 6$. Za bilo koju vrijednost od t koja je veća od 0 i manja od 1 očekujemo da je interpolirana vrijednost negdje između x_0 i x_1. Nadalje, što je vrijednost od t bliža 0, to je interpolirana vrijednost bliža x_0 (odnosno utjecaj od x_1 iščezava), a što je vrijednost od t bliža 1, to je interpolirana vrijednost bliža x_1 (odnosno utjecaj od x_0 iščezava). Možemo pisati:

$$x(t) = x_0 + (x_1 - x_0) \cdot t. \tag{3.1}$$

Lako je vidjeti da je uz $t = 0$:

$$x(0) = x_0 + (x_1 - x_0) \cdot 0 = x_0$$

odnosno da je uz $t = 1$:

$$x(1) = x_0 + (x_1 - x_0) \cdot 1 = x_0 + x_1 - x_0 = x_1.$$

Također, za $t \in (0, 1)$ vrijednost od $x(t)$ će biti iz intervala (x_0, x_1). Primjerice, za $t = 0.5$ dobit ćemo aritmetičku sredinu:

$$x(0.5) = x_0 + (x_1 - x_0) \cdot 0.5 = x_0 + 0.5 \cdot x_1 - 0.5 \cdot x_0 = 0.5 \cdot x_0 + 0.5 \cdot x_1 = \frac{x_0 + x_1}{2}.$$

Izraz (3.1) uobičajenije se prikazuje na način koji separira utjecaje početne i konačne vrijednosti. Izraz možemo raspisati na sljedeći način:

$$x(t) = x_0 + (x_1 - x_0) \cdot t = x_0 + x_1 \cdot t - x_0 \cdot t$$

što grupiranjem daje konačni izraz:

$$x(t) = (1 - t) \cdot x_0 + t \cdot x_1. \tag{3.2}$$

Sada je odmah vidljivo zašto za $t = 0$ vrijedi $x(0) = x_0$ odnosno zašto za $t = 1$ vrijedi $x(1) = x_1$. Uočimo također sljedeće važno svojstvo linearne interpolacije: izrazi kojima se množe početna i konačna vrijednost u sumi daju 1:

$$(1 - t) + t = 1.$$

Osvrnimo se još jednom na ovo što smo upravo napravili. Interpolaciju između dviju vrijednosti zapisali smo u obliku:

$$x(t) = b_0(t) \cdot x_0 + b_1(t) \cdot x_1. \tag{3.3}$$

gdje su $b_0(t)$ i $b_1(t)$ težinske funkcije (vidi sliku 3.2). Kod linearne interpolacije vrijedi:

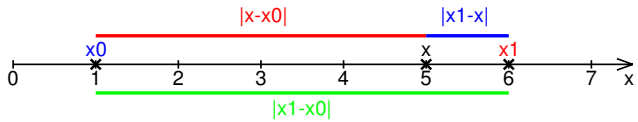
$$b_0(t) = 1 - t \qquad \text{te}$$

$$b_1(t) = t.$$

Pokazali smo i da vrijedi sljedeće svojstvo:

$$b_0(t) + b_1(t) = 1$$

odnosno da je suma baznih funkcija jednaka 1 (što se također jasno vidi sa slike 3.2). Ovaj kriterij osigurava da ako se prilikom interpolacije uzme primjerice $\frac{3}{4}$ vrijednosti od x_0, to će se uravnotežiti uzimanjem još $\frac{1}{4}$ vrijednosti od x_1.



Slika 3.1: Linearna interpolacija i baricentrične koordinate

S obzirom da je suma baznih funkcija upravo jednaka 1, uočimo da su $b_0(t)$ i $b_1(t)$ zapravo *baricentrične* koordinate točke $x(t)$. Neka je $x = b_0 \cdot x_0 + b_1 \cdot x_1$, uz $b_0 + b_1 = 1$. Tada se, u 1D slučaju kao što je ovaj naš, baricentrične koordinate mogu izračunati kao omjeri nasuprotne duljine i duljine čitavog segmenta – vidi sliku 3.1. Prisjetimo se, u 2D računali smo ih kao omjere nasuprotne površine i površine čitavog trokuta.

$$b_0 = \frac{\text{duljina}(x, x_1)}{\text{duljina}(x_0, x_1)} \qquad \text{te}$$

$$b_1 = \frac{\text{duljina}(x_0, x)}{\text{duljina}(x_0, x_1)}.$$

Primjerice, ako interpoliramo između 1 i 6, i promatramo točku 5 što je slučaj prikazan na slici 3.1, vrijedi:

$$\text{duljina}(x, x_1) = \text{duljina}(5, 6) = |6 - 5| = 1$$

$$\text{duljina}(x_0, x) = \text{duljina}(1, 5) = |5 - 1| = 4$$

$$\text{duljina}(x_0, x_1) = \text{duljina}(1, 6) = |6 - 1| = 5$$

pa je:

$$b_0 = \frac{\text{duljina}(x, x_1)}{\text{duljina}(x_0, x_1)} = \frac{1}{5} \qquad \text{te}$$

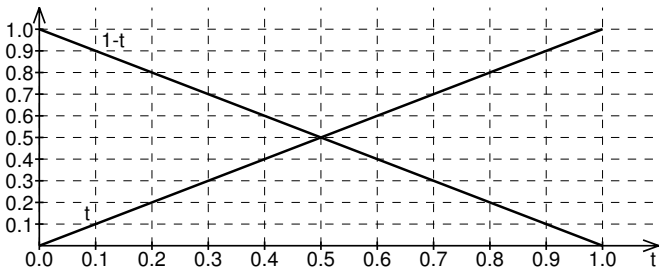
$$b_1 = \frac{\text{duljina}(x_0, x)}{\text{duljina}(x_0, x_1)} = \frac{4}{5} \qquad \text{odnosno}$$

$$5 = \frac{1}{5} \cdot 1 + \frac{4}{5} \cdot 6$$

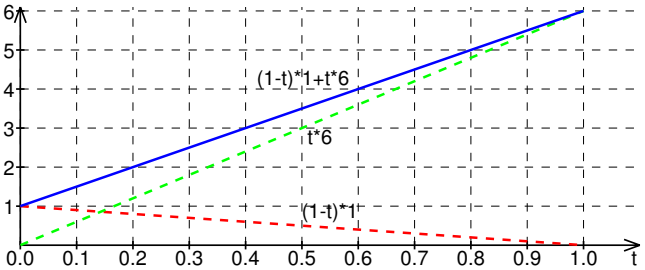
Primjer linearne interpolacije između vrijednosti 1 i 6 ilustriran je na slici 3.3, gdje je za svaki t prikazan udio komponenti $b_0(t) \cdot x_0$ i $b_1(t) \cdot x_1$ te njihova suma odnosno konačna interpolirana vrijednost.

Konačno, kod linearne interpolacije zadovoljeno je i sljedeće svojstvo: promjena parametra t za iznos Δt izazvat će proporcionalnu promjenu interpolirane vrijednosti:

$$x(t) = (1 - t) \cdot x_0 + t \cdot x_1$$



Slika 3.2: Težinske funkcije kod linearne interpolacije



Slika 3.3: Linearna interpolacija razložena na sastavne dijelove

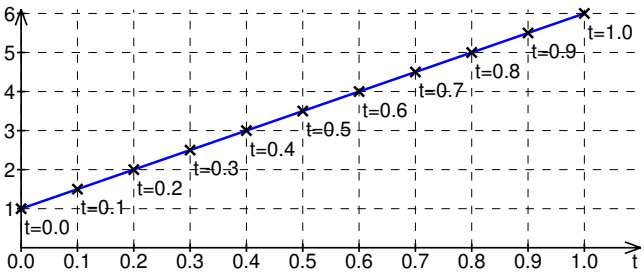
$$\begin{aligned} x(t+\Delta t) &= (1-(t+\Delta t))\cdot x_0 + (t+\Delta t)\cdot x_1 \\ &= (1-t-\Delta t)\cdot x_0 + (t+\Delta t)\cdot x_1 \end{aligned}$$

$$\begin{aligned} x(t+\Delta t) - x(t) &= (1-t-\Delta t)\cdot x_0 + (t+\Delta t)\cdot x_1 - ((1-t)\cdot x_0 + t\cdot x_1) \\ &= -\Delta t\cdot x_0 + \Delta t\cdot x_1 \\ &= \Delta t(x_1 - x_0) \\ &= \Delta t\cdot \Delta x \end{aligned}$$

pri čemu razlika krajnje i početne vrijednosti $x_1 - x_0$ predstavlja koeficijent proporcionalnosti. To je jasno ilustrirano na slici 3.4, gdje se vidi da za svaki $\Delta t = 0.1$ dolazi do promjene interpolirane vrijednosti iznosa 0.5, što upravo odgovara $\Delta t(x_1 - x_0) = 0.1 \cdot (6-1) = 0.1 \cdot 5 = 0.5$.

3.3 Interpolacija kubnim polinomima

Linearna interpolacija ne nudi nam nikakvu mogućnost upravljanja brzinom kojom se mijenjaju interpolirane vrijednosti. Zamislimo opet slučaj u kojem je



Slika 3.4: Razdioba vrijednosti kod linearne interpolacije

potrebno interpolirati pomicanje između dviju pozicija ali na način kako bismo to napravili da od početne pozicije do konačne pozicije trebamo doći vozilom. U početnoj poziciji, vozilo miruje i dodavanjem gasa vozilo će početi ubrzavati. S povećanjem brzine, za svaki novi Δt očekujemo da će pripadna promjena interpolirane vrijednosti rasti. To naravno ne smije ići u nedogled; ako vozilo ne počne usporavati na vrijeme, dogodit će se da prije no što t postane 1 već premašimo konačnu vrijednost. Stoga točno na pola puta kada smo postigli maksimalnu brzinu trebamo početi usporavati kako bi inkrementi interpoliranih vrijednosti postajali sve manji i manji, i konačno baš u trenutku kada interpolirana vrijednost postigne svoju konačnu vrijednost pali na nulu. Ovime smo opisali jedan primjer koji nam ilustrira potrebu da imamo više kontrole nad načinom kako se mijenjaju interpolirane vrijednosti. Za to ćemo trebati posegnuti za baznim funkcijama koje su polinomi višeg stupnja.

Pogledajmo stoga općenitiji slučaj interpolacije koji, općenito govoreći, spada u nelinearne interpolacije. Razmotrit ćemo interpolaciju kod koje su težinske funkcije kubni polinomi parametra t. I dalje razmatramo interpolaciju oblika:

$$x(t) = b_0(t) \cdot x_0 + b_1(t) \cdot x_1$$

pri čemu dopuštamo da su $b_0(t)$ i $b_1(t)$ kubni polinomi:

$$b_0(t) = a_0 \cdot t^3 + b_0 \cdot t^2 + c_0 \cdot t + d_0 \qquad \text{te}$$

$$b_1(t) = a_1 \cdot t^3 + b_1 \cdot t^2 + c_1 \cdot t + d_1.$$

Skrećemo pažnju čitatelju da oznake $b_0(t)$ i $b_1(t)$ predstavljaju funkcije (točnije kubne polinome) dok su b_0 i b_1 koeficijenti (skalarne vrijednosti) koji su u tim polinomima uz član t^2 – iz konteksta bi trebalo biti jasno o čemu se radi.

Postavit ćemo nekoliko uvjeta koje želimo imati zadovoljene, i iz njih potom odrediti koeficijente. Evo što želimo.

1. Suma težinskih funkcija mora biti 1:

$$b_0(t) + b_1(t) = 1.$$

2. Težina $b_1(t)$ u $t = 0$ mora biti jednaka 0 kako bi sav doprinos interpoliranoj vrijednosti bio od x_0 a u $t = 1$ mora biti jednaka 1 kako bi sav doprinos interpoliranoj vrijednosti bio od x_1.

3. Zahtijevamo simetričnost derivacije težine $b_1(t)$ s obzirom na krajeve intervala, tj. $\frac{db_1(t)}{dt} = \frac{db_1(1-t)}{dt}$. Derivacija težinske funkcije $b_1(t)$ s obzirom na parametar t predstavlja nagib odnosno brzinu kojom se mijenja ta težinska funkcija.

4. Zahtijevamo centralnu simetričnost oko točke $(\frac{1}{2}, \frac{1}{2})$ grafa funkcije $b_1(t)$, odnosno ako je za vrijednost t $b_1(t) = T$, tada vrijednost težinske funkcije u $1 - t$ mora biti jednaka $1 - T$.

Krenimo redom. Iz zahtjeva (2) slijedi:

$$\begin{aligned} b_1(0) = 0 \quad &\Rightarrow \quad a_1 \cdot 0^3 + b_1 \cdot 0^2 + c_1 \cdot 0 + d_1 = 0 \\ &\Rightarrow \quad d_1 = 0 \end{aligned}$$

$$\begin{aligned} b_1(1) = 1 \quad &\Rightarrow \quad a_1 \cdot 1^3 + b_1 \cdot 1^2 + c_1 \cdot 1 + d_1 = 1 \\ &\Rightarrow \quad a_1 + b_1 + c_1 + d_1 = 1 \qquad \text{no kako je} \quad d_1 = 0 \quad \text{slijedi} \\ &\Rightarrow \quad a_1 + b_1 + c_1 = 1 \end{aligned}$$

Kako bismo razriješili zahtjev (3), trebamo izračunati derivaciju:

$$\frac{db_1(t)}{dt} = 3a_1 \cdot t^2 + 2b_1 \cdot t + c_1. \tag{3.4}$$

Iz zahtjeva (3) sada slijedi:

$$3a_1 \cdot t^2 + 2b_1 \cdot t + c_1 = 3a_1 \cdot (1-t)^2 + 2b_1 \cdot (1-t) + c_1$$

iz čega dobivamo:

$$3a_1 + 2b_1 = 0.$$

Prema zahtjevu (4) imamo:

$$b_1(1-t) = 1 - b_1(t)$$

što uz već izvedenu jednakost $d_1 = 0$ znači:

$$a_1 \cdot (1-t)^3 + b_1 \cdot (1-t)^2 + c_1 \cdot (1-t) = 1 - \left(a_1 \cdot t^3 + b_1 \cdot t^2 + c_1 \cdot t\right)$$

odakle slijedi:

$$a_1 + b_1 + c_1 = 1.$$

Da bismo u potpunosti odredili koeficijente od $b_1(t)$ trebamo 4 uvjeta. Za sada smo utvrdili da mora vrijediti sljedeće:

- $d_1 = 0$,
- $3a_1 + 2b_1 = 0$ te
- $a_1 + b_1 + c_1 = 1$.

Nedostaje nam još jedan uvjet. Jedna od mogućnosti jest postaviti zahtjev na nagib težinske funkcije $b_1(t)$ za neku vrijednost parametra t. Ono što ćemo razmotriti jest nagib (odnosno iznos derivacije) u $t = 0$. Derivaciju smo već odredili – vidi izraz (3.4). Uvrštavanjem $t = 0$ u taj izraz slijedi da je vrijednost derivacije za $t = 0$ upravo jednaka koeficijentu c_1. Sada možemo riješiti prethodno definirani sustav jednadžbi za nekoliko zanimljivih slučajeva.

3.3.1 Derivacija jednaka 0

Ako tražimo da je za $t = 0$ vrijednost derivacije jednaka 0, imamo sustav jednadžbi:

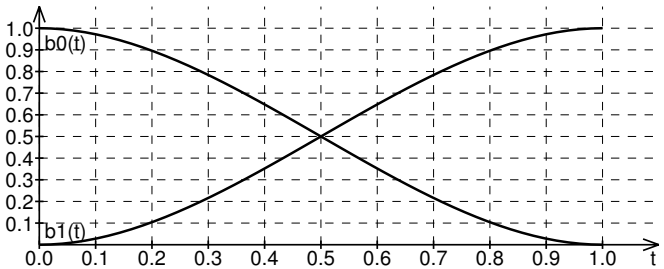
- $d_1 = 0$,
- $3a_1 + 2b_1 = 0$,
- $a_1 + b_1 + c_1 = 1$ te
- $c_1 = 0$.

Rješenje ovog sustava je $a_1 = -2$, $b_1 = 3$, $c_1 = 0$, $d_1 = 0$. Time je:

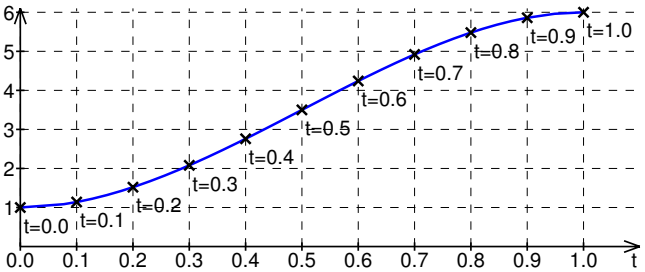
$$b_1(t) = -2t^3 + 3t^2 \quad \text{te}$$

$$b_0(t) = 1 - b_1(t) = 1 + 2t^3 - 3t^2.$$

Dobivene težinske funkcije prikazane su na slici 3.5a dok je rezultat interpolacije prikazan na slici 3.5b. Uočite kako se sada za jednaku promjenu parametra t interpolirana vrijednost više ne mijenja proporcionalno promjeni parametra – interpolacija nije linearna. Međutim, sa slike se lijepo vidi da se na početku interpolirane vrijednosti mijenjaju sporo, pa zatim se promjena ubrzava sve do $t = 0.5$ nakon čega počinje usporavanje promjene da bi na samom kraju promjena pala na 0. Razlog ovome je činjenica da smo tražili da nagib za $t = 0$ bude 0, što automatski zbog tražene simetričnosti na derivaciju znači i da isti takav nagib mora biti na kraju.



(a) Težinske funkcije dobivene uz zahtjev da je $\frac{db_1(t)}{dt}$ u $t = 0$ jednaka 0.



(b) Interpolacija vrijednosti uz težinske funkcije dobivene uz zahtjev da je $\frac{db_1(t)}{dt}$ u $t = 0$ jednaka 0.

Slika 3.5: Interpolacija kubnim polinomima uz zahtjev da je nagib u početnoj točki jednak 0.

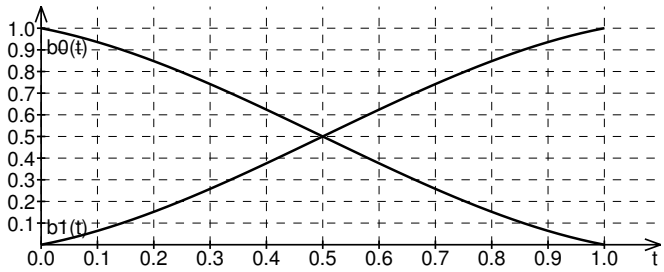
3.3.2 Derivacija jednaka $\frac{1}{2}$

Ako tražimo da je za $t = 0$ vrijednost derivacije jednaka $\frac{1}{2}$, imamo sustav jednadžbi:

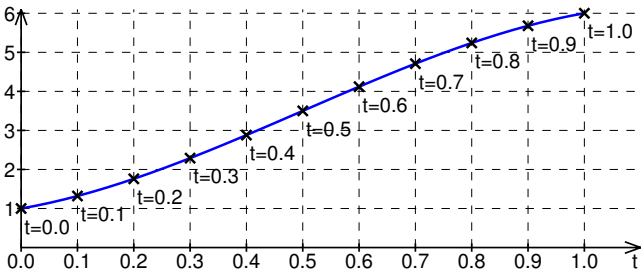
- $d_1 = 0$,
- $3a_1 + 2b_1 = 0$,
- $a_1 + b_1 + c_1 = 1$ te
- $c_1 = \frac{1}{2}$.

Rješenje ovog sustava je $a_1 = -1$, $b_1 = \frac{3}{2}$, $c_1 = \frac{1}{2}$, $d_1 = 0$. Time je:

$$b_1(t) = -t^3 + \frac{3}{2}t^2 + \frac{1}{2}t \quad \text{te}$$



(a) Težinske funkcije dobivene uz zahtjev da je $\frac{db_1(t)}{dt}$ u $t = 0$ jednaka $\frac{1}{2}$.



(b) Interpolacija vrijednosti uz težinske funkcije dobivene uz zahtjev da je $\frac{db_1(t)}{dt}$ u $t = 0$ jednaka $\frac{1}{2}$.

Slika 3.6: Interpolacija kubnim polinomima uz zahtjev da je nagib u početnoj točki jednak $\frac{1}{2}$.

$$b_0(t) = 1 - b_1(t) = 1 + t^3 - \frac{3}{2}t^2 - \frac{1}{2}t.$$

Dobivene težinske funkcije prikazane su na slici 3.6a dok je rezultat interpolacije prikazan na slici 3.6b. Uočite kako sada nagib u početnoj i krajnjoj točki više nije 0. Promjena interpolirane vrijednosti sada ide drugačijom dinamikom no što je to bio slučaju izveden uz zahtjev da je nagib u $t = 0$ jednak 0.

3.3.3 Derivacija jednaka 1

Ako tražimo da je za $t = 0$ vrijednost derivacije jednaka 1, imamo sustav jednadžbi:

- $d_1 = 0$,
- $3a_1 + 2b_1 = 0$,

- $a_1 + b_1 + c_1 = 1$ te
- $c_1 = 1$.

Rješenje ovog sustava je $a_1 = 0$, $b_1 = 0$, $c_1 = 1$, $d_1 = 0$. Time je:

$$b_1(t) = t \quad \text{te}$$

$$b_0(t) = 1 - b_1(t) = 1 - t.$$

Uočite što se dogodilo. Kubni polinom degradirao je u linearni, i dobili smo upravo izraze koji odgovaraju linearnoj interpolaciji. Naime, ovo je intuitivno lagano objasniti. Uz početni nagib od 0, dobivene težinske funkcije su morale biti takve da počnu povećavati brzinu promjene interpolirane vrijednosti kako bi se interpolirana vrijednost stigla promijeniti od početne do konačne. Kako početni nagib raste (ali mora biti manji od 1), sve je manji pritisak na porast nagiba. Granični slučaj nastupa kada je početna vrijednost nagiba upravo jednaka 1 – ona je upravo takva da ako se t promijeni od 0 do 1, interpolirana će se vrijednost promijeniti od početne do konačne – brzinu promjene ne treba niti ubrzavati niti usporavati, i kubni polinom degradira u linearni. Daljnjim porastom početnog nagiba interpolirane vrijednosti bi se prebrzo promijenile i interpolirana vrijednost za $t = 1$ bi uz taj nagib premašila zadanu konačnu vrijednost – stoga će u tom slučaju brzinu promjene trebati gušiti kako t bude rastao od 0 do 0.5, i to ćemo upravo vidjeti za sljedeći slučaj.

3.3.4 Derivacija jednaka 2

Ako tražimo da je za $t = 0$ vrijednost derivacije jednaka 2, imamo sustav jednadžbi:

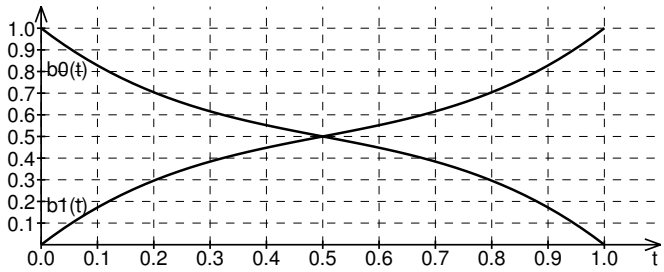
- $d_1 = 0$,
- $3a_1 + 2b_1 = 0$,
- $a_1 + b_1 + c_1 = 1$ te
- $c_1 = 2$.

Rješenje ovog sustava je $a_1 = 2$, $b_1 = -3$, $c_1 = 2$, $d_1 = 0$. Time je:

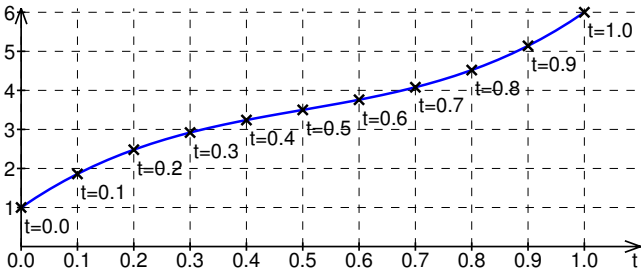
$$b_1(t) = 2t^3 - 3t^2 + 2t \quad \text{te}$$

$$b_0(t) = 1 - b_1(t) = 1 - 2t^3 + 3t^2 - 2t.$$

Dobivene težinske funkcije prikazane su na slici 3.7a dok je rezultat interpolacije prikazan na slici 3.7b. Uočite kako u ovom slučaju, s obzirom da je početni nagib prevelik (tj. veći od 1), s porastom parametra t nagib odnosno brzina porasta interpolirane vrijednosti pada do $t = 0.5$ i nakon toga zbog tražene simetričnosti derivacije opet počinje rasti.



(a) Težinske funkcije dobivene uz zahtjev da je $\frac{db_1(t)}{dt}$ u $t = 0$ jednaka 2.



(b) Interpolacija vrijednosti uz težinske funkcije dobivene uz zahtjev da je $\frac{db_1(t)}{dt}$ u $t = 0$ jednaka 2.

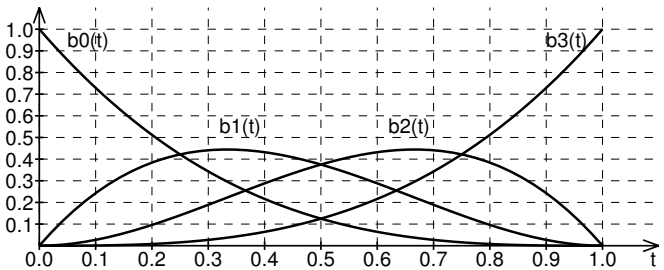
Slika 3.7: Interpolacija kubnim polinomima uz zahtjev da je nagib u početnoj točki jednak 2.

3.4 Drugi primjer interpolacije kubnim polinomima

Razmotrimo još jedan vrlo ilustrativan primjer interpolacije kubnim polinomima. U prethodnom poglavlju unaprijed smo definirali dva kubna polinoma (jedan za početnu vrijednost, drugi za konačnu vrijednost) i njihove koeficijente smo odredili postavljajući niz uvjeta na svojstva koja ti polinomi moraju zadovoljiti. Sada ćemo pogledati drugačiji primjer.

Krenut ćemo od zahtjeva da suma baznih funkcija mora biti jednaka 1. Ako je t parametar, tada vrijedi sljedeća jednakost:

$$1 = (1 - t) + t.$$



Slika 3.8: Dobivene kubne bazne funkcije.

Čitav izraz sada možemo dignuti na n-tu potenciju – primjenom binomnog poučka slijedi:

$$\begin{aligned} 1^n &= ((1-t)+t)^n \\ &= \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k \end{aligned}$$

gdje je $\binom{n}{k}$ binomni koeficijent (odnosno "n povrh k"). Za slučaj da je $n = 3$ tako imamo:

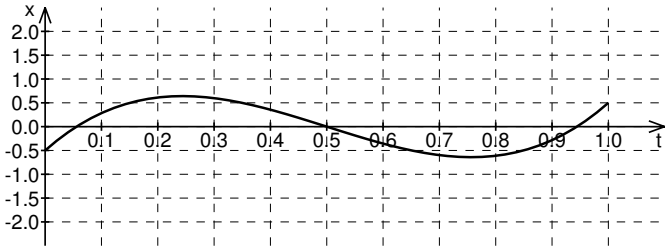
$$\begin{aligned} 1^3 &= ((1-t)+t)^3 \\ &= (1-t)^3 + 3 \cdot (1-t)^2 \cdot t + 3 \cdot (1-t) \cdot t^2 + t^3 \end{aligned}$$

čime smo dobili četiri bazne funkcije:

$$\begin{aligned} b_0(t) &= (1-t)^3 \\ b_1(t) &= 3 \cdot (1-t)^2 \cdot t \\ b_2(t) &= 3 \cdot (1-t) \cdot t^2 \\ b_3(t) &= t^3. \end{aligned}$$

Suma ovako izvedenih baznih funkcija je doista 1 – to je trivijalno za pokazati s obzirom da su nastale upravo ekspanzijom jedinice.

Ovako izvedene bazne funkcije mogu se iskoristiti za interpolaciju između dviju vrijednosti x_0 i x_1. Pogledajmo najprije grafički prikaz baznih funkcije na slici 3.8. S obzirom da je samo $b_0(t)$ u $t = 0$ jednaka 1, a znamo da interpolirana vrijednost u $t = 0$ mora biti jednaka x_0, slijedi da $b_0(t)$ mora množiti x_0. Isto tako, s obzirom da je samo $b_3(t)$ u $t = 1$ jednaka 1, a znamo da interpolirana vrijednost u $t = 1$ mora biti jednaka x_1, slijedi da $b_3(t)$ mora množiti x_1. Pogledamo li $b_1(t)$ i $b_2(t)$, uočit ćemo da njihov utjecaj u $t = 0$ i $t = 1$ iščezava – ti



Slika 3.9: Interpolacija kubnim baznim funkcijama: dobivena dva stupnja slobode.

polinomi ni na koji način ne utječu na početnu i konačnu vrijednost interpolirane vrijednosti. Postavlja se pitanje: što onda treba množiti bazne funkcije $b_1(t)$ i $b_2(t)$? Odgovor je: bilo što! Bazne funkcije $b_1(t)$ i $b_2(t)$ nam omogućavaju da odredimo što će se događati u međukoracima prilikom interpolacije između x_0 i x_1 – te dvije funkcije nam daju dva dodatna stupnja kontrole nad interpoliranim vrijednostima. Uvedimo stoga dvije pomoćne vrijednosti: x_A koja će množiti $b_1(t)$ i x_B koja će množiti $b_2(t)$. Tada izraz za interpoliranu vrijednost postaje:

$$n(t) = x_0 \cdot b_0(t) + x_A \cdot b_1(t) + x_B \cdot b_2(t) + x_1 \cdot b_3(t).$$

Prilikom interpolacije, kako t počinje rasti od vrijednosti 0, interpolirana vrijednost će krenuti od vrijednosti x_0 prema vrijednosti x_A da bi nakon toga bila privučena prema vrijednosti x_B i da bi potom konačno završila u vrijednosti x_1. Međutočke x_A i x_B možemo zamisliti kao da na interpoliranu vrijednost djeluju privlačnom silom: interpoliranu vrijednost će privući prema sebi, ali interpolirana vrijednost nikada neće baš poprimiti te vrijednosti. Jedan primjer ovakve interpolacije prikazan je na slici 3.9 gdje se interpolira između vrijednosti $x_0 = -0.5$ i $x_1 = 0.5$ pri čemu je $x_A = 3$ a $x_B = -3$.

Što možemo zaključiti iz opisanoga? Dizanjem izraza $(1-t)+t$ na n-tu potenciju nastat će $n+1$ bazna funkcija reda n. Dvije od tih baznih funkcija koriste se za množenje početne i konačne vrijednosti dok preostalih $n-1$ baznih funkcija nudi upravo dodatnih $n-1$ stupnjeva kontrole nad interpoliranim vrijednostima. Tako smo za slučaj kubnih polinoma $n = 3$ dobili još $n - 1 = 3 - 1 = 2$ stupnja kontrole. Dizanjem izraza na drugu potenciju dobili bismo bazne funkcije $b_0(t) = (1-t)^2$, $b_1(t) = 2 \cdot (1-t) \cdot t$ i $b_2(t) = t^2$; imali bismo jedan dodatni stupanj kontrole nad interpoliranim vrijednostima, zahvaljujući baznoj funkciji $b_1(t)$.

Ovdje opisan postupak izvođenja baznih funkcija za interpolaciju dizanjem izraza $(1-t)+t$ na n-tu potenciju čini osnovu za izvođenje *Bézierovih* krivulja

koje ćemo obraditi nešto kasnije; stoga neka Vas ne začudi pojava binomnih koeficijenata u izrazu za Bézierovu krivulju – vratite se samo na ovo poglavlje za pojašnjenje izvoda.

3.5 Bilinearna interpolacija

Prethodno opisana linearna interpolacija koristi se za interpolaciju između dvije vrijednosti koje smo označavali s x_0 i x_1. U praksi se, međutim, često javlja problem interpolacije između četiri vrijednosti, odnosno preciznije bi bilo reći između dva para vrijednosti.

Ilustrirajmo ovo najprije na jednostavnom primjeru. Radimo program za renderiranje 3D modela s potporom za ljepljenje tekstura na objekte. Kao teksturu smo uzeli sliku `pepsi.png` - to je slika dimenzija 100×400 koja prikazuje limenku *pepsi-cole* u 256 nijansi sive boje. Naša scena se sastoji od jednog valjka na koji je potrebno zalijepiti ovu teksturu kako konačni rezultat ne bi bio jednobojni valjak već valjak koji izgleda kao limenka *pepsi-cole*. Potom smo pokrenuli generiranje visokorezolucijskog prikaza naše scene. Prilikom renderiranja, program je utvrdio da slikovnom elementu ekrana na poziciji $(352, 692)$ odgovara element teksture s koordinatama $(29.75, 140.25)$. Postavlja se pitanje – koju ćemo vrijednost slikovnog elementa iz slike teksture uzeti za popunjavanje odabranog slikovnog elementa zaslona? Da su kojim slučajem koordinate u teksturi ispale cjelobrojne, primjerice $(29, 140)$, mogli bismo bez razmišljanja uzeti vrijednost tog slikovnog elemenata. Međutim, to se nije dogodilo. Jedno moguće rješenje problema je postupiti na sljedeći način: zaokružimo dobivene koordinate $(29.75, 140.25) \rightarrow (30, 140)$ i očitajmo boju slikovnog elementa na tim koordinatama u teksturi. Nažalost, kako zaokruživanje tipično vodi do nazubljenog prikaza i niza drugih problema, bolje je rješenje boju konstruirati interpolacijom. Kako je x koordinata s kojom ulazimo u teksturu 29.75, nalazimo se između slikovnog elementa koji ima $x = 29$ i slikovnog elementa koji ima $x = 30$; isto tako, kako je y koordinata s kojom ulazimo u teksturu 140.25, nalazimo se između slikovnog elementa koji ima $y = 140$ i slikovnog elementa koji ima $y = 141$. Oko pogođene lokacije imamo dakle 4 slikovna elementa: $(29, 140)$, $(30, 140)$, $(29, 141)$ i $(30, 141)$. Za svaki od tih slikovnih elemenata u teksturi možemo pročitati boju (odnosno u našem slučaju gdje radimo s 256 nijansi sive, možemo pročitati pridruženi intenzitet sive).

Ideja bilinearne interpolacije je sljedeća. Pogledajmo intenzitete lijevog i desnog slikovnog elementa za $y = 140$ i izračunajmo linearnu interpolaciju tog intenziteta. Označimo parametar kojim radimo interpolaciju po horizontali s u. Pri tome $u = 0$ odgovara intenzitetu lijevog slikovnog elementa, a $u = 1$ odgovara intenzitetu desnog slikovnog elementa. Konkretna vrijednost parametra u u primjeru je 0.75 (na tri smo četvrtine puta od 29. do 30. slikovnog elementa).

Osim lijevog i desnog para slikovnih elemenata na $y = 140$, u teksturi imamo i lijevi i desni slikovni element na $y = 141$. Napravimo stoga linearnu interpolaciju intenziteta lijevog i desnog slikovnog elementa koje imamo za $y = 141$. Time smo dobili dvije interpolirane vrijednosti: jednu za $y = 140$ a drugu za $y = 141$.

Da bismo dobili konačnu vrijednosti, sada trebamo napraviti linearnu interpolaciju između tih dviju vrijednosti promatrajući naš položaj na y-osi. Označimo s v parametar koji predstavlja interpolaciju po vertikali. Neka $v = 0$ odgovara interpoliranoj vrijednosti koju smo dobili za $y = 140$ te neka $v = 1$ odgovara interpoliranoj vrijednosti koju smo dobili za $y = 141$. U promatranom primjeru, $v = 0.25$ jer se nalazimo na jednoj četvrtini puta od $y = 140$ do $y = 141$. Konačna vrijednost je dakle linearna interpolacija između rezultata dviju linearnih interpolacija – stoga i naziv: *bilinearna interpolacija.*

3.5.1 Formalna definicija

Označimo vrijednost intenziteta koja pripada donjem lijevom slikovnom elementu s x_{00}, vrijednost intenziteta koja pripada donjem desnom slikovnom elementu s x_{10}, vrijednost intenziteta koja pripada gornjem lijevom slikovnom elementu s x_{01} te vrijednost intenziteta koja pripada gornjem desnom slikovnom elementu s x_{11} (vidi sliku 3.10). Potrebno je izračunati bilinearnu interpolaciju $x(u, v)$ gdje je u parametar koji predstavlja vodoravnu interpolaciju a v okomitu interpolaciju. Uz ove oznake vrijedi sljedeće:

- $x(0, 0) = x_{00}$,
- $x(1, 0) = x_{10}$,
- $x(0, 1) = x_{01}$ te
- $x(1, 1) = x_{11}$.

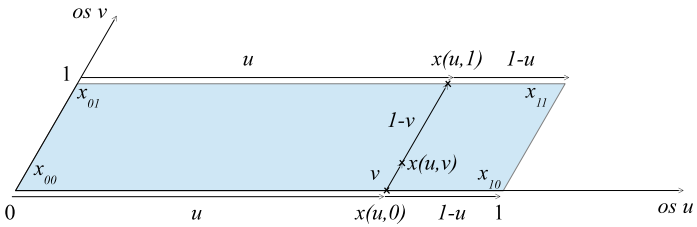
Prvi korak pri bilinearnoj interpolaciji jest provedba horizontalne linearne interpolacije za donje vrijednosti (gdje je $v = 0$) i za gornje vrijednosti (gdje je $v = 1$), kako je prikazano izrazima (3.5) i (3.6). Slika 3.10 ilustrira potrebne korake.

$$x(u, 0) = (1 - u) \cdot x_{00} + u \cdot x_{10} \tag{3.5}$$

$$x(u, 1) = (1 - u) \cdot x_{01} + u \cdot x_{11} \tag{3.6}$$

Drugi korak je napraviti linearnu interpolaciju po vertikali, između upravo dobivenih vrijednosti $x(u, 0)$ i $x(u, 1)$. Kombiniranjem izraza 3.5 i 3.6 možemo pisati:

$$\begin{aligned} x(u, v) &= (1 - v) \cdot x(u, 0) + v \cdot x(u, 1) \\ &= (1 - v) \cdot \{(1 - u) \cdot x_{00} + u \cdot x_{10}\} + v \cdot \{(1 - u) \cdot x_{01} + u \cdot x_{11}\} \end{aligned}$$



Slika 3.10: Izvod bilinearne interpolacije.

što vodi na konačni izraz za bilinearnu interpolaciju koji je dan izrazom (3.7).

$$x(u,v) = (1-u)\cdot(1-v)\cdot x_{00} + u\cdot(1-v)\cdot x_{10} + (1-u)\cdot v\cdot x_{01} + u\cdot v\cdot x_{11} \quad (3.7)$$

Ovaj izvod motivirali smo potrebom za određivanje intenziteta slikovnog elemenata koji se u sliku teksture ne preslikava točno u jedan slikovni element. No to nije jedina primjena bilinearne interpolacije. Bilinearna interpolacija općenito je primjenjiva na svaki problem interpolacije između dva para zadanih vrijednosti.

Trilinearna interpolacija prirodno je proširenje bilinearne interpolacije na tri dimenzije pri čemu se vrijednost interpolira na temelju zadanih 8 vrijednosti. Pri tome se linearno interpolira između dviju bilinearno interpoliranih vrijednosti. Za dobivanje intuicije o ovom postupku pogledajte ponovno sliku 3.10 ali sada zamislite da smo dodali i treću os odnosno još četiri točke koje su smještene iznad donje četiri točke. U tom slučaju trebamo još jedan parametar te možemo zamisliti da smo proveli bilinearnu interpolaciju na temelju donje četiri točke, potom bilinearnu interpolaciju na temelju gornje četiri točke, i zatim uporabom trećeg parametra linearno interpolirali između tih dviju dobivenih vrijednosti.

3.6 Interpolacija vektora

Do sada smo razmatrali interpolaciju između skalarnih vrijednosti. U računalnoj grafici zna se javiti i potreba za interpolacijom između dvaju vektora. Stoga ćemo u nastavku razmotriti kako možemo provesti ovu vrstu interpolacije.

Pretpostavimo da je s $\vec{v}_0$ označen početni vektor a s $\vec{v}_1$ označen konačni vektor. Neka je $\vec{v}(t)$ interpolirana vrijednost vektora ovisna o parametru $t \in [0,1]$ uz uobičajene pretpostavke da je $\vec{v}(0) = \vec{v}_0$ odnosno $\vec{v}(1) = \vec{v}_1$.

3.6.1 Linearna interpolacija vektora

Linearna interpolacija vektora (*LERP*, od engleskog naziva *Linear interpolation*) je postupak kod kojeg se linearno interpolira svaka komponenta vektora zasebno.

Izraz (3.8) daje matematičku formulaciju.

$$\vec{v}(t) = (1-t) \cdot \vec{v}_0 + t \cdot \vec{v}_1 \tag{3.8}$$

Primjerice, u 2D prostoru vektore $\vec{v}_0$ i $\vec{v}_1$ možemo zapisati kao:

$$\vec{v}_0 = v_{0,x}\vec{i} + v_{0,y}\vec{j}$$

$$\vec{v}_1 = v_{1,x}\vec{i} + v_{1,y}\vec{j}$$

čime se izraz (3.8) raspada u dva izraza:

$$v_x(t) = (1-t) \cdot v_{0,x} + t \cdot v_{1,x}$$

$$v_y(t) = (1-t) \cdot v_{0,y} + t \cdot v_{1,y};$$

dakako, $\vec{v}(t) = v_x(t)\vec{i} + v_y(t)\vec{j}$.

Ovako izvedena interpolacija ima sljedeća svojstva:

1. kut interpoliranog vektora se postupno mijenja od početnog prema konačnom, no promjena kuta nije linearna;
2. norma interpoliranog vektora može biti neočekivana – primjerice, interpolacijom između vektora norme 1 i vektora norme 2 moguće je dobiti vektor norme 0.1 te
3. računska složenost je vrlo niska, što je veliki bonus.

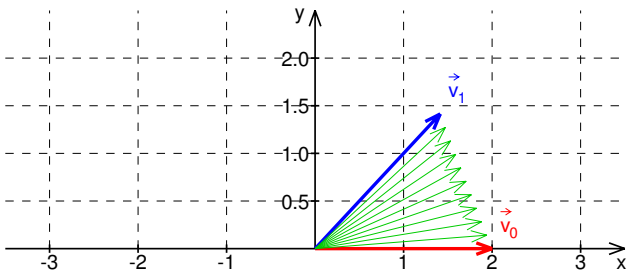
Tri primjera linearne interpolacije vektora prikazana su na slici 3.11.

Nešto složenija interpolacija opisana u nastavku ide u smjeru interpolacije smjera i norme interpoliranog vektora.

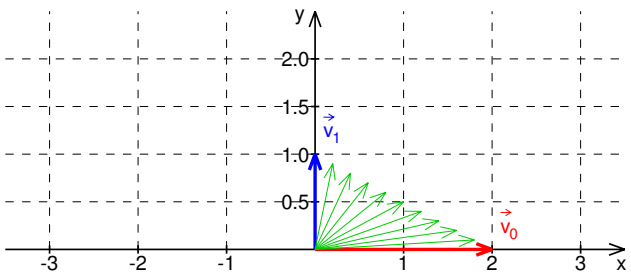
3.6.2 Sferna linearna interpolacija

Sferna linearna interpolacija vektora (*SLERP*, od engleskog naziva *Spherical Linear interpolation*) je postupak kod kojeg se nad zadanim vektorima konstruira minimalna hiperkugla i potom se kao interpolirana vrijednost vektora uzima vektor koji najkraćim putem putuje po dijelu površine hiperkugle. Dakako, postavlja se zahtjev da su početni vektor $\vec{v}_0$ i završni vektor $\vec{v}_1$ jedinični vektori (tj. vektori norme 1).

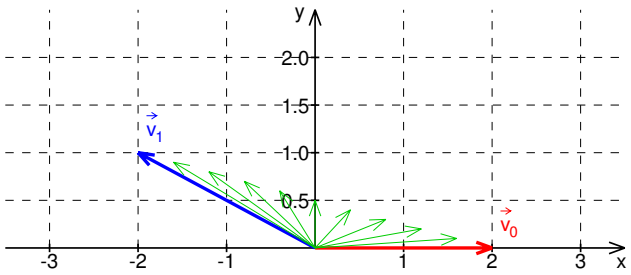
Da bismo došli do izraza za *SLERP*, promotrimo sliku 3.12 koja ovo, bez gubitka općenitosti, ilustrira u dvodimenzionalnom prostoru u kojem je hipersfera zapravo kružnica. Na slici su prikazani početni vektor $\vec{v}_0$ i završni vektor $\vec{v}_1$ koji s početnim vektorom $\vec{v}_0$ zatvara kut Ω. Vektor $\vec{v}$ je interpolirani vektor dobiven za kut $\Theta = t \cdot \Omega$. Pri tome se interpolira po parametru t, te očekujemo da je za $t = 0$ interpolirani vektor $\vec{v}(0) = \vec{v}_0$ a za $t = 1$ interpolirani vektor $\vec{v}(1) = \vec{v}_1$. Za t iz intervala $[0, 1]$ kut Θ kretat će se od 0 do Ω. Kao pomoć u izračunu prikazan je i vektor $\perp \vec{v}_0$ koji je okomit na vektor $\vec{v}_0$.



(a) Interpolacija između vektora $(2, 0)$ i $(\sqrt{2}, \sqrt{2})$

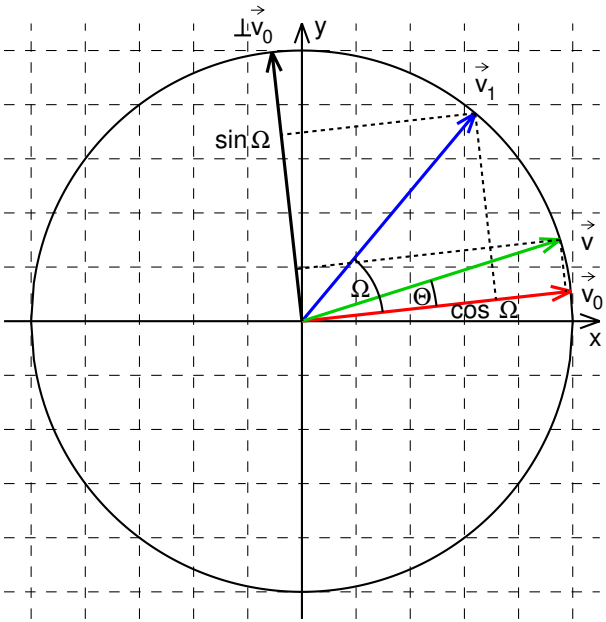


(b) Interpolacija između vektora $(2, 0)$ i $(0, 1)$



(c) Interpolacija između vektora $(2, 0)$ i $(-2, 1)$

Slika 3.11: Primjeri linearne interpolacije vektora.



Slika 3.12: Izvod interpolacije *SLERP*.

Uočimo da su vektori $\vec{v}_0$ i $\perp \vec{v}_0$ dva međusobno okomita jedinična vektora – stoga oni čine ortonormiranu bazu. U toj bazi vektor $\vec{v}_1$ možemo zapisati kako slijedi:

$$\vec{v}_1 = \cos(\Omega) \cdot \vec{v}_0 + \sin(\Omega) \cdot \perp \vec{v}_0$$

iz čega dalje slijedi:

$$\sin(\Omega) \cdot \perp \vec{v}_0 = \vec{v}_1 - \cos(\Omega) \cdot \vec{v}_0$$

$$\perp \vec{v}_0 = \frac{1}{\sin(\Omega)} \cdot \vec{v}_1 - \frac{\cos(\Omega)}{\sin(\Omega)} \cdot \vec{v}_0.$$

Vektor $\vec{v}_1$ u istoj je bazi moguće zapisati preko njegovih projekcija na $\vec{v}_0$ i $\perp \vec{v}_0$, pa imamo:

$$\begin{aligned}
\vec{v} &= \cos(\Theta) \cdot \vec{v}_0 + \sin(\Theta) \cdot \perp \vec{v}_0 \\
&= \cos(\Theta) \cdot \vec{v}_0 + \sin(\Theta) \cdot \left(\frac{1}{\sin(\Omega)} \cdot \vec{v}_1 - \frac{\cos(\Omega)}{\sin(\Omega)} \cdot \vec{v}_0 \right) \\
&= \cos(\Theta) \cdot \vec{v}_0 + \frac{\sin(\Theta)}{\sin(\Omega)} \cdot \vec{v}_1 - \frac{\sin(\Theta)\cos(\Omega)}{\sin(\Omega)} \cdot \vec{v}_0 \\
&= \frac{\cos(\Theta)\sin(\Omega) - \cos(\Omega)\sin(\Theta)}{\sin(\Omega)} \cdot \vec{v}_0 + \frac{\sin(\Theta)}{\sin(\Omega)} \cdot \vec{v}_1 \\
&= \frac{\sin(\Omega - \Theta)}{\sin(\Omega)} \cdot \vec{v}_0 + \frac{\sin(\Theta)}{\sin(\Omega)} \cdot \vec{v}_1
\end{aligned}$$

Uvažavanjem izraza $\Theta = t \cdot \Omega$ i činjenice da je $\vec{v}$ zapravo funkcija parametra t možemo pisati:

$$\vec{v}(t) = \frac{\sin(\Omega - t \cdot \Omega)}{\sin(\Omega)} \cdot \vec{v}_0 + \frac{\sin(t \cdot \Omega)}{\sin(\Omega)} \cdot \vec{v}_1$$

što daje konačni izraz:

$$\vec{v}(t) = \frac{\sin\left((1-t) \cdot \Omega\right)}{\sin(\Omega)} \cdot \vec{v}_0 + \frac{\sin(t \cdot \Omega)}{\sin(\Omega)} \cdot \vec{v}_1 \tag{3.9}$$

Primjeri *SLERP* interpolacije vektora prikazani su na slici 3.13. U određenim slučajevima *SLERP* može raditi dobro čak i za slučaj da početni i konačni vektori nisu jedinični, što se vidi na slikama 3.13b i 3.13c. Također, treba pripaziti da se izraz ne primjenjuje za kut $\Omega = k \cdot 180$ř (gdje je $k \in Z$ jer u tim slučajevima u danom izrazu imamo dijeljenje s nulom).

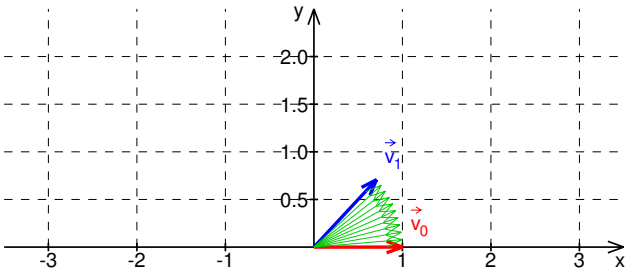
Ovako izvedena interpolacija ima dva svojstva:

1. kut interpoliranog vektora se linearno mijenja od početnog prema konačnom te se norma interpoliranog vektora također čuva (barem u slučaju da su početni i konačni vektor normirani);

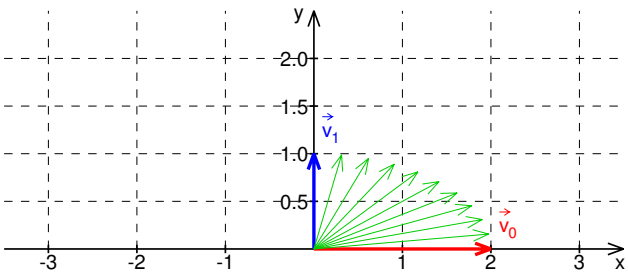
2. računska složenost je, međutim, veća no što je to kod *LERP*-a.

Također, ova interpolacija ne radi ako je kut između dvaju vektora 0° ili 180°, jer u tom slučaju najkraći put po površini hipersfere nije jedinstven.

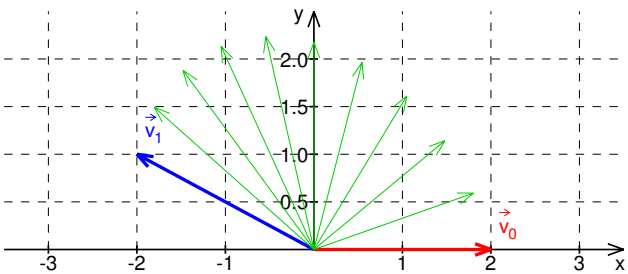
Kut između vektora $\vec{v}_0$ i $\vec{v}_1$ može se izračunati preko kosinusa tog kuta koji je određen skalarnim produktom ta dva vektora podijeljenim umnoškom njihovih normi. Vektorski oblik izvedene formule za interpolaciju primjenjiv je na proizvoljno dimenzionalne vektore.



(a) Interpolacija između vektora $(1, 0)$ i $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$



(b) Interpolacija između vektora $(2, 0)$ i $(0, 1)$



(c) Interpolacija između vektora $(2, 0)$ i $(-2, 1)$

Slika 3.13: Primjeri sferne linearne interpolacije vektora.

3.6.3 Modificirana sferna linearna interpolacija

Interpolacija *SLERP* radi dobro za interpolaciju između jediničnih vektora. Uporabom te metode može se napraviti metoda interpolacije koja je računski još složenija, ali nudi jednoliku interpolaciju nagiba interpoliranog vektora (kao što to nudi *SLERP*) te linearnu interpolaciju norme interpoliranog vektora.

Ideja je sljedeća: najprije od vektora $\vec{v}_0$ i $\vec{v}_1$ izračunamo jedinične vektore; potom izračunamo pomoćni interpolirani vektor (koji će tada isto biti jedinični) i zatim mu linearno skaliramo normu. Izrazi su sljedeći:

$$\vec{\tilde{v}}_0 = \frac{\vec{v}_0}{\|\vec{v}_0\|}$$

$$\vec{\tilde{v}}_1 = \frac{\vec{v}_1}{\|\vec{v}_1\|}$$

$$\vec{w}(t) = \mathrm{SLERP}(\vec{\tilde{v}}_0, \vec{\tilde{v}}_1, t)$$

$$\vec{v}(t) = \left((1-t)\cdot\|\vec{v}_0\| + t\cdot\|\vec{v}_1\|\right)\vec{w}(t)$$

Primjeri ovako izvedene interpolacije prikazani su na slici 3.14. Treba uočiti da je ovako izvedena interpolacija računski najzahtjevnija od svih prethodno prikazanih interpolacija.

S obzirom da se kut između vektora $\vec{v}_0$ i $\vec{v}_1$ ne mijenja ako vektore normiramo, lako je pokazati (napravite za vježbu) da prethodno opisani postupak generira interpolaciju prema sljedećem izrazu:

$$\begin{aligned}\vec{v}(t) = \quad & \left\{1 + t\cdot\left(\frac{\|\vec{v}_1\|}{\|\vec{v}_0\|} - 1\right)\right\}\frac{\sin((1-t)\cdot\Omega)}{\sin(\Omega)}\cdot\vec{v}_0 \\ & + \left\{\frac{\|\vec{v}_0\|}{\|\vec{v}_1\|} - t\cdot\left(\frac{\|\vec{v}_0\|}{\|\vec{v}_1\|} - 1\right)\right\}\frac{\sin(t\cdot\Omega)}{\sin(\Omega)}\cdot\vec{v}_1.\end{aligned} \tag{3.10}$$

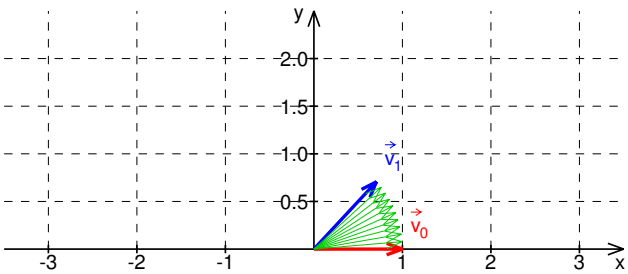
3.7 Ponavljanje

1. Opišite na koji se način provodi linearna interpolacija između dvije zadane vrijednosti.

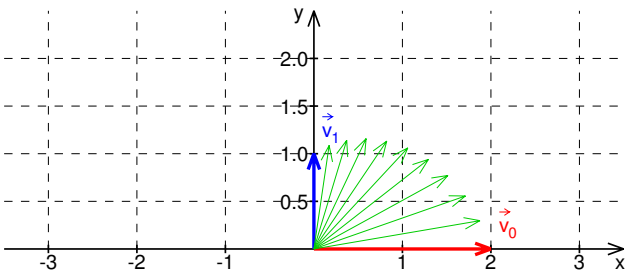
2. Na koji se način linearna interpolacija obavlja preko baricentričnih koordinata? Koja je u tom slučaju interpretacija baricentričnih koordinata?

3. Ako interpolaciju radimo polinomima koji nastaju ekspanzijom broja 1 (sekcija 3.4), koliko baznih funkcija imamo ako je stupanj polinoma n, koji su to polinomi i kako se radi interpolacija između dviju zadanih vrijednosti? Jesu li nam u tom slučaju dovoljne samo te dvije vrijednosti?

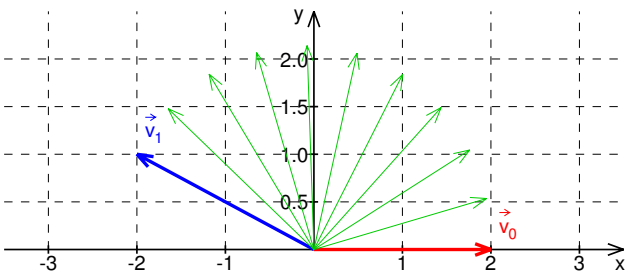
4. Kako glasi formalna definicija bilinearne interpolacije? Između koliko vrijednosti se tada interpolira? S koliko je parametara zadana ova interpolacija?



(a) Interpolacija između vektora $(1, 0)$ i $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$



(b) Interpolacija između vektora $(2, 0)$ i $(0, 1)$



(c) Interpolacija između vektora $(2, 0)$ i $(-2, 1)$

Slika 3.14: Modifikacija sferne linearne interpolacije vektora.

5. Kako glasi formalna definicija trilinearne interpolacije? Između koliko vrijednosti se tada interpolira? S koliko je parametara zadana ova interpolacija?

6. Kako se radi linearna interpolacija vektora? Koji je njezin osnovni problem?

7. Kako se radi sferna linearna interpolacija vektora? Koje je njezino ograničenje?

Poglavlje 4

Crtanje linija i poligona na rasterskim prikaznim jedinicama

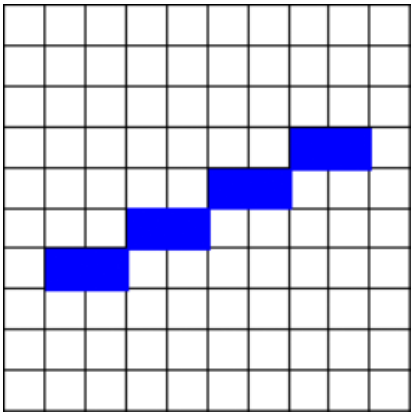
4.1 Uvod

Kada na monitoru želimo nacrtati liniju, za to obično pozivamo ugrađenu funkciju jezika u kojem pišemo program. Pri tome zadajemo koordinate početne točke i završne točke, i očekujemo da će funkcija u što kraćem vremenu nacrtati tu liniju. No na koji način se izvodi to brzo crtanje linije? Slično pitanje vrijedi i za složenije likove – poligone, kod kojih pri tome možemo iscrtavati samo obrub, ili ih možemo i popunjavati. Oba ova zadatka pri tome rezultiraju osvjetljavanjem određenog broja slikovnih elemenata na zaslonu, odnosno na rasterskoj prikaznoj jedinici. Naime, slika koju prikazuje monitor sastoji od niza elementarnih djelića slike - slikovnih elemenata (engl. *pixel – picture element*) čiji su položaji određeni cjelobrojnim koordinatama pa je slika koju prikazuje monitor diskretna. Moguće je osvijetliti slikovni element na poziciji npr. (0,0), no nije moguće osvijetliti slikovni element na poziciji (1.13,2.52) jer takav ne postoji. Posljedica ovakve diskretizacije slike je da prilikom crtanja linija dolazi do nazubljenosti istih (slika 4.1 ovo jasno ilustrira). Više o ovoj pojavi reći ćemo u nastavku. Pa krenimo najprije s metodom crtanja linije.

4.1.1 Bresenhamov postupak crtanja linije

Postupak kojim se danas uobičajeno crtaju linije jest implementacija Bresenhamovog postupka. Pa pogledajmo o čemu se tu radi.

Prilikom crtanja linije poznate su nam koordinate početne i završne točke. Isto tako znamo da liniju crtamo u ravnini. Zaboravimo na tren da liniju crtamo



Slika 4.1: Prikaz linije na rasterskoj prikaznoj jedinici

na zaslonu. Zamislimo da imamo idealnu prikaznu jedinicu koja može prikazati sve što poželimo s beskonačno finom preciznošću. Prva stvar koju moramo napraviti jest izračunati koje sve točke moramo osvijetliti. Da bismo si olakšali razmatranje, zadajmo točke na slijedeći način:

- Početna točka T_S ima cjelobrojne koordinate (x_s, y_s).
- Završna točka T_E ima cjelobrojne koordinate (x_e, y_e).
- Vrijedi: $x_s < x_e$, $y_s < y_e$.
- Pravac je pod kutom manjim ili jednakim 45°.

Kasnije ćemo se osloboditi ovih ograničenja no za početak neka ograničenja vrijede. Koordinate točaka T_S i T_E u ovom slučaju nisu označene standardnim oznakama $(T_{S,1}, T_{S,2})$ i $(T_{E,1}, T_{E,2})$ već su iskorištene oznake (x_s, y_s) i (x_e, y_e) budući da ćemo cijeli postupak izvoditi imajući na umu računala gdje su zasloni dvodimenzionalni sa standardnim oznakama x i y za pojedine osi.

Jednadžba pravca kroz dvije točke može se napisati prema relaciji u obliku:

$$y - y_s = \frac{y_e - y_s}{x_e - x_s}(x - x_s)$$

ili nakon sređivanja i uvođenja oznake a za koeficijent smjera pravca (odnosno tangens kuta) te oznake b za odsječak na osi y:

$$y = a \cdot x + b$$

pri čemu je

$$a = \frac{y_e - y_s}{x_e - x_s}, \qquad b = -a \cdot x_s + y_s.$$

a i b su izdvojeni kao zasebne konstante jer se mogu izračunati samo jednom na početku, s obzirom da ovise samo o konstantama. Umjesto oznaka a i b često su u uporabi i oznake k za koeficijent smjera i l za odsječak na osi y.

Uz ovu posljednju formulu pravac se može nacrtati sljedećim trivijalnim algoritmom:

```
void nacrtaj(int xs, int ys, int xe, int ye) {
    double a = (ye-ys)/(double)(xe-xs);
    double b = -a*xs+ys;

    for(int x=xs; x<=xe; x++) {
        int y = zaokruzi(a*x + b);
        osvijetli_pixel(x,y);
    }
}
```

S obzirom na diskretiziranost monitora potrebno je za svaki x pronaći odgovarajući y i tu točku osvijetliti. No ovaj algoritam, iako radi, ima jedan veliki nedostatak: sporost. Osnovni problem ovog algoritma je množenje u petlji. Za svaki x odgovarajući y računa se množenjem i to u aritmetici pomičnog zareza. Algoritam vapi za poboljšanjem!

Pogledamo još jednom što prethodno opisani algoritam radi. Varijabla x mijenja se u petlji od vrijednosti x_s do vrijednosti x_e po jedan i za svaki x se računa odgovarajući y uz uporabu operacije množenja. Pogledajmo malo bolje što to program zapravo računa:

$$\begin{aligned}
y|_{x=x_s} &= a \cdot x_s + b = y_s \\
y|_{x=x_s+1} &= a \cdot (x_s + 1) + b = a \cdot x_s + b + a = y|_{x=x_s} + a \\
y|_{x=x_s+2} &= a \cdot (x_s + 2) + b = a \cdot x_s + b + 2a = y|_{x=x_s+1} + a \\
y|_{x=x_s+3} &= a \cdot (x_s + 3) + b = a \cdot x_s + b + 3a = y|_{x=x_s+2} + a
\end{aligned}$$

Pogledamo li desne strane u svakom retku, možemo vidjeti da se svaki redak može dobiti tako da se prethodnom doda vrijednost varijable a. To je izvrstan rezultat jer nam ukazuje na to da nam više ne treba vremenski zahtjevno množenje. Možda bi bio problem s prvim retkom, jer po našoj filozofiji on nema prethodnika pa bismo tu trebali koristiti množenje, no nije tako. Vrijednost prvog retka već nam je zadana i iznosi y_s. Pogledajmo sada što smo dobili.

```
void nacrtaj(int xs, int ys, int xe, int ye) {
    int x,y_pom;
    double a,y;
```

```
a = (ye-ys)/(double)(xe-xs);

y=ys;
for(x=xs; x<=xe; x++) {
    y_pom = zaokruzi(y);
    osvijetli_pixel(x,y_pom);
    y = y + a;
}
```

Poboljšanje je već značajno. Izvan petlje vrijednost varijable y postavlja se na početnu vrijednost. U petlji je uvedena pomoćna varijabla *y_pom* koja pamti vrijednost varijable y nakon zaokruživanja jer varijablu y ne smijemo dirati budući da nam treba i u nastavku.

Pogledajmo što još ne valja. Prečesto zovemo funkciju zaokruzi(y). Bilo bi jako zgodno kada bismo mogli i bez zaokruživanja znati koja je zaokružena vrijednost y koordinate, ili ako ništa drugo, pokušati funkciju zvati rjeđe. Naravno, i to se može. Evo kako.

4.1.2 Izvod Bresenhamovog algoritma s decimalnim brojevima

Ideja je sljedeća. Početnu zaokruženu vrijednost y koordinate znamo: to je y_s, tj. za $x = x_s$ imamo $y = y_s$. Trebamo dakle osvijetliti slikovni element na poziciji (x_s, y_s). Kada se pomaknemo za jedan slikovni element udesno, vrijednost y koordinate poveća se za koeficijent a. Kako smo postavili ograničenje da je pravac pod kutom manjim ili jednakim 45°, to znači da se vrijednost varijable a kreće između 0 i 1. Uzmimo za primjer da a iznosi 0.2. Tada nakon jednog pomaka u desno y se je povećao za 0.2 te iznosi $y = y_s + 0.2$. Zaokruživanjem ove vrijednosti dolazi se opet do vrijednosti y_s te se y koordinata točke koju trebamo osvijetliti ne razlikuje od prethodnog koraka. Znači, trebamo osvijetliti slikovni element na poziciji $(x_s + 1, y_s)$. Pomaknimo se za još jedan slikovni element u desno. y-koordinatu opet treba uvećati za vrijednost varijable a. Sada to iznosi $y = y_s + 0.2 + 0.2 = y_s + 0.4$. Zaokruživanjem se opet dolazi do vrijednosti $y = y_s$, pa osvjetljavamo slikovni element na poziciji $(x_s + 2, y_s)$. Idemo za još jedan slikovni element u desno: $y = y_s + 0.4 + 0.2 = y_s + 0.6$. Konačno, zaokruživanjem ove vrijednosti penjemo se za jedan slikovni element prema gore: $y = y_s + 1$, i osvijetljavamo slikovni element na poziciji $(x_s + 2, y_s + 1)$.

Označimo cjelobrojnu vrijednost koordinate y s y_c, a pridodani decimalni dio s y_f. Tada gornji postupak možemo opisati ovako: osvjetljavamo točke na poziciji $(x_s + k, y_c)$. Na početku je $y_c = y_s$ a $y_f = 0$. Svakim pomakom u desno y_f uvećavamo za koeficijent a. Onog trenutka kada y_f pređe (ili dođe na) 0.5, y_c uvećavamo za jedan.

U ovom trenutku možemo nastaviti na dva načina.

1. Možemo nastaviti prilikom pomaka u desno s dodavanjem vrijednosti varijable a varijabli y_f. U tom slučaju sljedeće uvećavanje y_c-a za jedan biti će kada y_f prekorači 1.5, pa sljedeće kada prekorači 2.5 itd. No ova ideja i nije baš najbolja jer opet dozvoljava da y_f postane veći od jedan pa moramo voditi računa da svaki puta cjelobrojni dio "zaboravimo" i promatramo samo decimalni ostatak, što je opet vremenski zahtjevno.

2. Možemo od y_f oduzeti 1 i time poništiti nagomilanu pogrešku te y_f zadržati u opsegu od -0.5 do 0.5 čime nam je usporedba znatno olakšana.

Budući da je drugi postupak bolji, odlučit ćemo se za njega. No u tom slučaju treba još pojasniti zašto se od y_f oduzima baš 1. Razmislimo malo što nam predstavlja y_f. Crtajmo liniju iz početne točke s y koordinatom jednakom 0 i ponovimo ukratko gore opisani postupak. Prvo smo bili u $y = 0$, odnosno $y_c = 0$, $y_f = 0$. Zatim smo došli u $y = 0.2$, tj. $y_c = 0$, $y_f = 0.2$. Zatim $y = 0.4$, tj. $y_c = 0$, $y_f = 0.4$, i konačno $y = 0.6$, tj. $y_c = 1$, $y_f = 0.6$. Vrijednost pohranjena u y_f nam zapravo govori koliko smo pobjegli od cjelobrojne koordinate. Npr. za $y = 0.4$, tj. $y_c = 0$, $y_f = 0.4$, osvijetlit ćemo slikovni element s y-koordinatom 0, dok smo realno gledajući već 0.4 slikovnog elementa iznad. Onog trena kada od slikovnog elementa pobjegnemo za 0.5 ili više (npr. za $y_f = 0.6$), prema dogovoru uvećavamo y_c za jedan. No tada više nismo u točki s y-koordinatom 0, već s y-koordinatom 1. A to za pogrešku y_f znači da više nismo 0.6 slikovnog elementa iznad, nego 0.4 slikovnog elementa ispod trenutnog slikovnog elementa određenog s y_c, što znači da pogreška od $y_f = 0.6$ prelazi u $y_f = 0.6 - 1 = -0.4$.

Implementacija ovog algoritma dana je u nastavku.

```
void nacrtaj(int xs, int ys, int xe, int ye) {
    int x,yc;
    double a,yf;

    a = (ye-ys)/(double)(xe-xs);

    yc=ys; yf=0.;
    for(x=xs; x<=xe; x++) {
        osvijetli_pixel(x,yc);
        yf=yf+a;
        if(yf>=0.5) {
            yf=yf-1.0;
            yc=yc+1;
        }
    }
}
```

Uvedemo li još samo jednu minornu modifikaciju, a to je da inicijalno za y_f uzmemo vrijednost -0.5, te poslije usporedbu radimo s $0.5 - 0.5 = 0$ (dakle nulom) dobili smo Bresenhamov algoritam!

```
void bresenham_nacrtaj(int xs, int ys, int xe, int ye) {
    int x,yc;
    double a,yf;

    a = (ye-ys)/(double)(xe-xs);

    yc=ys; yf=-0.5;
    for( x = xs; x <= xe; x++ ) {
        osvijetli_pixel(x,yc);
        yf=yf+a;
        if(yf>=0.) {
            yf=yf-1.0;
            yc=yc+1;
        }
    }
}
```

Može li bolje? Naravno...

4.1.3 Izvod Bresenhamovog algoritma s cijelim brojevima

Do osnovnog Bresenhamovog postupka došli smo krenuvši iz samo jednog zahtjeva – brzine. Pogledom na sam algoritam postavlja se pitanje može li još bolje? Trn u oku u opisanom algoritmu svakako su brojevi s pomičnim zarezom. Pokazuje se da se cijeli algoritam može prebaciti u cjelobrojnu domenu, a opće je poznato da su operacije s cijelim brojevima daleko brže od aritmetike u pomičnom zarezu. Stoga ćemo se u nastavku pozabaviti prilagodbom osnovnog Bresenhamovog algoritma tako da proradi s cijelim brojevima.

Osnovni element koji je uveo decimalne brojeve u igru bio je koeficijent smjera koji smo računali prema formuli:

$$a = \frac{y_e - y_s}{x_e - x_s}.$$

Ovaj koeficijent koristili smo prilikom izračuna pogreške (varijabla y_f). Pri tome smo, nakon svakog pomaka po x-u za jedan, pogrešku računali prema formuli:

$$y_f = y_f + a.$$

Idemo to malo raspisati. Uvrštavanjem izraza za a dobiva se:

$$y_f = y_f + \frac{y_e - y_s}{x_e - x_s} = \frac{y_f \cdot (x_e - x_s) + y_e - y_s}{x_e - x_s}.$$

Množenjem čitave jednadžbe nazivnikom razlomka s desne strane dobivamo:

$$y_f \cdot (x_e - x_s) = y_f \cdot (x_e - x_s) + y_e - y_s.$$

Uvođenjem $y_f^{'} = y_f \cdot (x_e - x_s)$ izraz prelazi u:

$$y_f^{'} = y_f^{'} + y_e - y_s.$$

Ovaj posljednji redak nam govori da umjesto dosadašnje pogreške možemo pamtiti pogrešku pomnoženu s koeficijentom $x_e - x_s$. Kako su to sve cijeli brojevi, ovdje smo se riješili sporih decimalnih brojeva. Sljedeći korak je rješavanje inicijalnih uvjeta. Naime, inicijalno vrijednost pogreške postavljamo na $-\frac{1}{2}$. To opet možemo raspisati:

$$y_f = -\frac{1}{2} \quad | \cdot (x_e - x_s) \qquad \Rightarrow \qquad y_f \cdot (x_e - x_s) = -\frac{x_e - x_s}{2}$$

$$y_f^{'} = -\frac{x_e - x_s}{2}$$

Skoro pa dobro. Još da se riješimo dvojke u nazivniku i svi naši problemi su riješeni. Dvojke ćemo se jednostavno riješiti tako da sve pomnožimo s 2. No tada na lijevoj strani umjesto nove pogreške stoji njezina dvostruka vrijednost. To će nas još prisiliti da kod izvoda nove pogreške sve pomnožimo s dvojkom, te će se dobiti redom izrazi opisani u nastavku.

- Za pogrešku ćemo koristiti izraz:

$$y_f = y_f + \frac{y_e - y_s}{x_e - x_s} = \frac{y_f \cdot (x_e - x_s) + y_e - y_s}{x_e - x_s} | \cdot 2(x_e - x_s)$$

$$2 \cdot y_f \cdot (x_e - x_s) = 2 \cdot y_f \cdot (x_e - x_s) + 2 \cdot (y_e - y_s).$$

  Uvođenjem $y_f^{'} = 2 \cdot y_f \cdot (x_e - x_s)$ izraz prelazi u:

$$y_f^{'} = y_f^{'} + 2 \cdot (y_e - y_s).$$

- Inicijalno dodjeljivanje tada prelazi u

$$y_f = -\frac{1}{2} \quad |\cdot 2 \cdot (x_e - x_s) \qquad \Rightarrow \qquad 2 \cdot y_f \cdot (x_e - x_s) = -2 \cdot \frac{x_e - x_s}{2} = -(x_e - x_s)$$

$$y_f^{'} = -(x_e - x_s).$$

- uz ove oznake oduzimanje jedinice od pogreške može se zapisati kao:

$$y_f = y_f - 1 \quad |\cdot 2 \cdot (x_e - x_s) \qquad \Rightarrow \qquad 2 \cdot y_f \cdot (x_e - x_s) = 2 \cdot y_f \cdot (x_e - x_s) - 2(x_e - x_s)$$

$$y_f^{'} = y_f^{'} - 2(x_e - x_s).$$

Imajući u vidu ove izmjene, može se napisati Bresenhamov algoritam s cijelim brojevima. Pri tome će se umjesto oznake y_f' koristiti standardna oznaka y_f podrazumijevajući da se pri tome govori o novo-definiranoj pogreški. Tako napisana varijanta Bresenhamovog algoritma sa cijelim brojevima prikazana je u nastavku.

```
void bresenham_nacrtaj_cjelobrojni1(int xs, int ys, int xe, int ye) {
    int x,yc,korekcija;
    int a,yf;

    a = 2*(ye-ys);

    yc=ys; yf=-(xe-xs); korekcija=-2*(xe-xs);
    for( x = xs; x <= xe; x++ ) {
        osvijetli_pixel(x,yc);
        yf=yf+a;
        if(yf>=0) {
            yf=yf+korekcija;
            yc=yc+1;
        }
    }
}
```

4.1.4 Kutevi od 0° do 90°

Sada kada smo se riješili decimalnih brojeva, vrijeme da se riješimo i ograničenja vezanih uz kutove. Pa idemo postupno. Razmatranje ćemo raditi za postupak s cijelim brojevima, ali ćemo imati stalno u vidu kako smo do tog postupka zapravo došli. Pravac pod kutom od 90° za algoritam s decimalnim brojevima (osnovna izvedba) bio je neizvediv. Naime, tangens kuta je tada beskonačno i imamo dijeljenje s nulom (i vjerojatno nasilni prekid programa). No, algoritam s cijelim brojevima nema dijeljenja pa ovo više nije problem.

Za kuteve od 0° do 45° algoritam smo već izveli. No zašto smo se ograničili na to područje? Odgovor opet treba tražiti u iznosu tangensa kuta. Naime, na tom intervalu on se kreće u granicama od nula do jedan, *osiguravajući pri tome da ćemo se pomakom za jedan slikovni element u desno pomaknuti maksimalno za jedan slikovni element prema gore*. A što ako dopustimo da tangens postane veći od 1? To znači da bismo se jednim pomakom u desno mogli pomaknuti i za više slikovnih elemenata prema gore (vodeći računa o tome da ih sve treba osvijetliti). Međutim, pojavljuje se problem u određivanju koje sve slikovne elemente pri tom penjanju treba osvijetliti. Da ne ulazimo dublje u prazne rasprave o ovome, rješenje ćemo potražiti na drugi način.

Ako je kut između pravca i apscise veći od 45°, tada je očito kut između pravca i ordinate manji od 45°. Znači, ako sada jednostavno zamijenimo osi prilikom postupka crtanja, umjesto problematičnog pomaka za više slikovnih elemenata

prema gore sa svakim pomakom u desno dobivamo maksimalno pomak za jedan slikovni element u desno sa svakim pomakom prema gore. I to je rješenje. Sve što treba napraviti jest ispitati je li tangens kuta veći od 1, tj. je li kut veći od 45°. Ako nije, koristimo već poznati algoritam; ako je mijenjamo uloge osima i kopiramo algoritam. Evo implementacije:

```
void bresenham_nacrtaj_cjelobrojni2(int xs, int ys, int xe, int ye) {
    int x,yc,korekcija;
    int a,yf;

    if(ye-ys <= xe-xs) {
        a = 2*(ye-ys);
        yc=ys; yf=-(xe-xs); korekcija=-2*(xe-xs);
        for( x = xs; x <= xe; x++ ) {
            osvijetli_pixel(x,yc);
            yf=yf+a;
            if(yf>=0) {
                yf=yf+korekcija;
                yc=yc+1;
            }
        }
    } else {
        // zamijeni x i y koordinate
        x=xe; xe=ye; ye=x;
        x=xs; xs=ys; ys=x;
        a = 2*(ye-ys);
        yc=ys; yf=-(xe-xs); korekcija=-2*(xe-xs);
        for( x = xs; x <= xe; x++ ) {
            osvijetli_pixel(yc,x);
            yf=yf+a;
            if(yf>=0) {
                yf=yf+korekcija;
                yc=yc+1;
            }
        }
    }
}
```

Funkcija kreće s ispitivanjem je li razlika po y-u manja od razlike po x-u. Ako je, pravac je pod kutom manjim od 45° i koristi se već izvedeni algoritam. Ako je razlika po y-u veća od razlike po x-u, koristeći pomoćnu varijablu x zamjenjuju se x i y koordinate točaka i nastavlja se s crtanjem. U petlji se u ovom slučaju može uočiti još jedna razlika: funkciji osvijetli_piksel predajemo na prvi pogled zamijenjene koordinate. No imajući u vidu da smo već prethodno napravili jednu zamjenu, ono što se mijenja u varijabli y_c zapravo je vrijednost x komponente točke i obrnuto.

4.1.5 Kutevi od 0° do -90°

Osnovna razlika od prethodnog slučaja je okomito gibanje koje sada ide prema dolje, umjesto prema gore. Zbog toga y_c treba umanjivati, a ne uvećavati za jedan. Isto tako, postupak pri računanju pogreške se ponešto modificira, no modifikacije su čisto kozmetičke prirode. Evo algoritma:

```
void bresenham_nacrtaj_cjelobrojni3(int xs, int ys, int xe, int ye) {
    int x,yc,korekcija;
    int a,yf;

    if(-(ye-ys) <= xe-xs) {
        a = 2*(ye-ys);
        yc=ys; yf=(xe-xs); korekcija=2*(xe-xs);
        for( x = xs; x <= xe; x++ ) {
            osvijetli_pixel(x,yc);
            yf=yf+a;
            if(yf<=0) {
                yf=yf+korekcija;
                yc=yc-1;
            }
        }
    } else {
        x=xe; xe=ys; ys=x;
        x=xs; xs=ye; ye=x;
        a = 2*(ye-ys);
        yc=ys; yf=(xe-xs); korekcija=2*(xe-xs);
        for( x = xs; x <= xe; x++ ) {
            osvijetli_pixel(yc,x);
            yf=yf+a;
            if(yf<=0) {
                yf=yf+korekcija;
                yc=yc-1;
            }
        }
    }
}
```

Od izmjena u odnosu na kod koji crta pravce pod kutovima od 0° do 90° mogu se navesti sljedeće:

- inicijalna vrijednost pogreške promijenila je predznak,
- y-komponenta se ne uvećava za jedan već se umanjuje za jedan te
- kako je $y_s > y_e$ ispitivanje $(y_e - y_s) <= (x_e - x_s)$ pretvoreno je u $-(y_e - y_s) <= (x_e - x_s)$ da bi se poništio negativan predznak rezultata na lijevoj strani.

4.1.6 Konačan kod za sve kuteve

Do sada smo obradili slučajeve kada smo se pri crtanju pravca gibali od lijeva u desno. Ostalo nam je obraditi crtanje pravca kojeg bismo trebali crtati s desna u lijevo. No za ovo nam ne trebaju posebni algoritmi: ukoliko je pravac zadan tako da se crta s desna u lijevo, zamjenom početne i završne koordinate dobivamo pravac koji se crta s lijeva u desno.

```
void bresenham_nacrtaj_cjelobrojni(int xs, int ys, int xe, int ye) {
    if( xs <= xe ) {
        if( ys <= ye ) {
            bresenham_nacrtaj_cjelobrojni2(xs,ys,xe,ye);
        } else {
            bresenham_nacrtaj_cjelobrojni3(xs,ys,xe,ye);
        }
    } else {
        if( ys >= ye ) {
            bresenham_nacrtaj_cjelobrojni2(xe,ye,xs,ys);
        } else {
            bresenham_nacrtaj_cjelobrojni3(xe,ye,xs,ys);
        }
    }
}
```

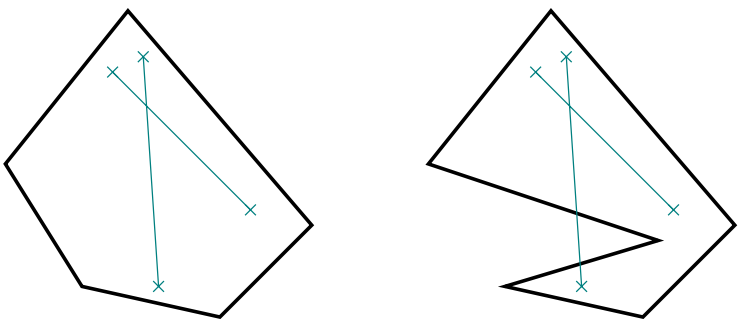
Funkcija ovisno o predanim točkama poziva odgovarajuću funkciju i pri tome po potrebi zamijenjuje početnu i krajnju točku. Prisjetimo se što smo već implementirali:

- funkcija bresenham_nacrtaj_cjelobrojni2 crta linije pod kutovima od 0° do 90°, uz $x_s < x_e$,
- funkcija bresenham_nacrtaj_cjelobrojni3 crta linije pod kutovima od 0° do -90°, uz $x_s < x_e$ te
- funkcija bresenham_nacrtaj_cjelobrojni kombinirajući prethodne dvije crta linije pod svim kutovima, uz proizvoljan odnos koordinata x_s i x_e.

4.2 Konveksni poligon

Poligonalna linija određena točkama T_0, T_1, ..., T_n je unija dužina $\overline{T_0T_1}$, $\overline{T_1T_2}$, ..., $\overline{T_{n-1}T_n}$. Točke T_0, ..., T_n zovu se vrhovi poligonalne linije. Zatvorena poligonalna linija je ona poligonalna linija za koju vrijedi $T_0 = T_n$. Jednostavna poligonalna linija je ona poligonalna linija kod koje se samo susjedne dužine sijeku i to u svojim krajevima; drugim riječima, jednostavna poligonalna linija ne siječe samu sebe. Jednostavan poligon (u daljnjem tekstu poligon) je dio ravnine koji se nalazi unutar zatvorene jednostavne poligonalne linije. Vrhovi poligona

su ujedno i vrhovi poligonalne linije koja ga zatvara. Dužina koja povezuje dva susjedna vrha zove se brid poligona. Treba napomenuti da je u matematici uobičajen naziv stranica poligona, no mi ćemo koristiti pojam brid zbog proširenja na trodimenzionalna tijela. Bridove ćemo označavati oznakom b_i, gdje je i indeks brida. Može se uočiti da je broj bridova uvijek jednak broju vrhova poligona. Vrhove ćemo zadavati u radnom prostoru i redoslijed zadavanja bit će točno definiran, jer taj redoslijed određuje kako ćemo povezivati vrhove poligona u bridove. Poligon je konveksan ako sadrži spojnicu svakog para točaka tog poligona. Jasnije tumačenje daje slika 4.2. Poligon koji nije konveksan je konkavan.



(a) Konveksni poligon (b) Konkavni poligon

Slika 4.2: Razlika između konveksnog i konkavnog poligona

4.2.1 Matematički opis poligona

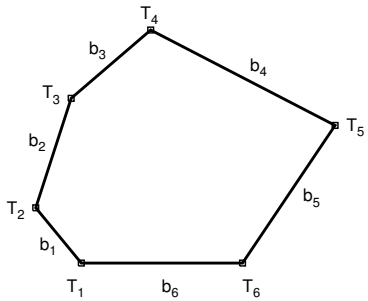
Osnovno što će nas kod poligona interesirati, i pomoću čega ćemo donositi razne zaključke jesu jednadžbe bridova poligona. U ovom razmatranju ograničit ćemo se na poligone u dvije dimenzije. Prema slici 4.3, brid b_i određen je vrhovima:

$$b_i \ldots \begin{cases} T_i, T_{i+1} & 0 < i < n \\ T_i, T_1 & i = n \end{cases} \tag{4.1}$$

Pri tome ćemo bridove prikazivati u homogenom prostoru. Sve koordinate vrhova poligona prije uporabe proširit ćemo homogenim parametrom 1, budući da vrhove poligona zadajemo u radnom prostoru.

Parametarska jednadžba pravca (u radnom prostoru) glasi:

$$T_b = \begin{cases} (T_{i+1} - T_i) \cdot \lambda + T_i & 0 < i < n \\ (T_1 - T_i) \cdot \lambda + T_i & i = n \end{cases} \tag{4.2}$$



Slika 4.3: Poligon

gdje je T_b proizvoljna točka pravca. Točke koje pripadaju bridu dobivaju se za vrijednosti $\lambda \in [0,1]$. Za vrijednosti $\lambda < 0$ ili $\lambda > 1$ točke koje se dobivaju više ne pripadaju bridu.

Jednadžba pravca u homogenom prostoru može se dobiti kao vektorski produkt vrhova pravca:

$$T_i \times T_{i+1} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \\ T_{i,1} & T_{i,2} & 1 \\ T_{i+1,1} & T_{i+1,2} & 1 \end{bmatrix} \tag{4.3}$$

$$= \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \begin{bmatrix} T_{i,2} - T_{i+1,2} \\ -(T_{i,1} - T_{i+1,1}) \\ T_{i,1}T_{i+1,2} - T_{i,2}T_{i+1,1} \end{bmatrix} \tag{4.4}$$

$$= \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot B_i \quad \text{za} \quad 0 < i < n \tag{4.5}$$

$$T_i \times T_1 = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \\ T_{i,1} & T_{i,2} & 1 \\ T_{1,1} & T_{1,2} & 1 \end{bmatrix} \tag{4.6}$$

$$= \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \begin{bmatrix} T_{i,2} - T_{1,2} \\ -(T_{i,1} - T_{1,1}) \\ T_{i,1}T_{1,2} - T_{i,2}T_{1,1} \end{bmatrix} \tag{4.7}$$

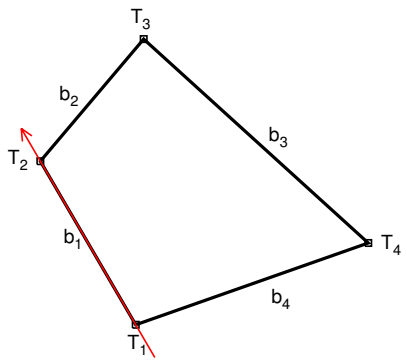
$$= \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot B_i \quad \text{za} \quad i = n \tag{4.8}$$

pri čemu je B_i matrica koeficijenata. U 2D prostoru to bi bila matrica $[\ a \quad b \quad c\ ]^T$; pripadna jednadžba pravca u tom bi slučaju glasila: $ax + by + ch = 0$.

Oznaka $T_{i,k}$ predstavlja k-tu komponentu točke T_i. Kako smo se ogradili na 2D prostor, umjesto $T_{i,1}$ mogli smo pisati $T_{i,x}$ a umjesto $T_{i,2}$ mogli smo pisati $T_{i,y}$. Iz razloga koji će kasnije biti objašnjeni, posebno je važno poštivati redosljed vrhova, odnosno prilikom izračuna jednadžbi bridova vrhove uzimati uvijek na isti način (u smjeru kazaljke na satu ili obrnuto, ali konzistentno).

4.2.2 Orijentacija vrhova konveksnog poligona

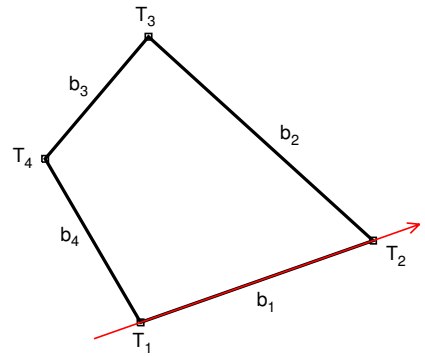
Konveksan poligon može imati vrhove zadane tako da se njihovim obilaskom gibamo u smjeru kazaljke na satu ili u smjeru suprotnom od smjera kazaljke na satu; ovaj smjer zovemo orijentacijom vrhova poligona. Orijentacija vrhova bit će nam bitna u daljnjim razmatranjima, pa je nužno objasniti kako se ista može ispitati. U prethodnom potpoglavlju izračunali smo jednadžbe bridova. Na temelju tih jednadžbi može se ustanoviti u kojem su odnosu neka promatrana točka i zadani brid, odnosno pravac na kojem brid leži. Mogući odgovori su pri tome: točka se nalazi iznad pravca, točka se nalazi na pravcu te točka se nalazi ispod pravca. Pametnim odabirom ispitne točke jednostavno ćemo doći do spoznaje o orijentaciji vrhova. Pogledajmo sliku 4.4.



Slika 4.4: Orijentacija vrhova poligona – u smjeru kazaljke na satu. Vrh T_3 je ispod brida b_1.

Brid b_1 određen je vrhovima T_1 i T_2. U kakvom je odnosu taj brid s točkom T_3? Točka T_3 nalazi se ispod brida. Isto vrijedi i za brid b_2 određen vrhovima T_2 i T_3 i točku T_4. Točka T_4 je ispod brida b_2. Obiđemo li tako sve bridove u krug ispitujući odnos tog brida i prvog sljedećeg vrha poligona, rezultat je uvijek isti. Pogledamo li kako su zadani vrhovi našeg poligona, vidimo da su zadani u smjeru kazaljke na satu. Ova spoznaja daje naslutiti kriterij koji bi se mogao koristiti, no prije nego što proglasimo kriterij ispravnim, pogledajmo i drugi slučaj. Na slici 4.5 prikazan je poligon s vrhovima zadanim u smjeru suprotnom od smjera kazaljke na satu.

Pogledamo sada odnos brida b_1 određenog vrhovima T_1 i T_2 i točke T_3. Točka T_3 nalazi se iznad brida b_1. Na sličan način opet se može vidjeti da ovo vrijedi za svaki brid i prvi sljedeći vrh poligona. Dakle, kriterij je ispravan. Evo načina kako provjeriti orijentaciju vrhova poligona.



Slika 4.5: Orijentacija vrhova poligona – u smjeru suprotnom od smjera kazaljke na satu. Vrh T_3 je iznad brida b_1.

- Vrhovi poligona zadani su u smjeru kazaljke na satu ako vrijedi:

$$(\forall i) T_j \cdot B_i \leq 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \leq i \leq n \end{cases} .$$

- Vrhovi poligona zadani su u smjeru suprotnom od smjera kazaljke na satu ako vrijedi:

$$(\forall i) T_j \cdot B_i \geq 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \leq i \leq n \end{cases} .$$

Ako ustanovimo da je poligon zadan uz jednu orijentaciju vrhova, a nama treba suprotna orijentacija vrhova, tada se jednostavno može zamijeniti redoslijed vrhova poligona i ponovno preračunati jednadžbe bridova.

Imajući u vidu da radimo s konveksnim poligonom kod kojeg su vrhovi zadani nekim konzistentnim redosljedom, može se jednostavno pokazati sljedeće:

- ako vrijedi da

$$(\exists i) T_j \cdot B_i < 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \leq i \leq n \end{cases}$$

dakle da postoji bar jedan vrh (prvi iza dva koja određuju brid) takav da se nalazi ispod brida, tada mora vrijediti:

$$(\forall i) T_j \cdot B_i \leq 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \leq i \leq n \end{cases}$$

odnosno da to vrijedi za svaki vrh. Ovo proizlazi iz činjenice da je poligon konveksan. Isto tako se može pokazati i da

- ako vrijedi

$$(\exists i) T_j \cdot B_i > 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \leq i \leq n \end{cases}$$

dakle da postoji bar jedan vrh (prvi iza dva koja određuju brid) takav da se nalazi iznad brida, tada mora vrijediti:

$$(\forall i) T_j \cdot B_i \geq 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \leq i \leq n \end{cases}$$

odnosno da to vrijedi za svaki vrh.

Prethodna dva uvjeta mogu se složiti u *kriterij koji određuje tip poligona*: konveksan ili konkavan. Možemo reći ovako. Poligon je konveksan ako vrijedi:

$$(\forall i) T_j \cdot B_i \leq 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \leq i \leq n \end{cases}$$

ili

$$(\forall i) T_j \cdot B_i \geq 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \leq i \leq n \end{cases} .$$

Konveksnost zahtijeva da ukoliko za neki vrh T_{i+2} utvrdimo da je ispod (iznad) brida b_i (određenog vrhovima T_i i T_{i+1}) tada i svi vrhovi T_{j+2} moraju biti ispod (iznad) svih bridova b_j poligona. Ukoliko pak utvrdimo da su neki vrhovi iznad a neki ispod odgovarajućeg brida, tada je poligon sigurno konkavan.

Ovdje izrečeni kriterij možemo iskoristiti i na drugi način. Ako sigurno znamo da je poligon konveksan, tada se određivanje orijentacije vrhova svodi na jedno jedino ispitivanje. Npr. uzmemo točku T_3 i brid b_1 (određen točkama T_1 i T_2). Ako je T_3 ispod b_1, orijentacija je u smjeru kazaljke na satu; ako je T_3 iznad b_1, orijentacija je u smjeru suprotnom od smjera kazaljke na satu. Jedino što se ovdje nepredviđenoga može dogoditi jest da je poligon zadan "čudno" pa da vrh T_3 leži na bridu b_1. No tada se jednostavno pomaknemo na sljedeći brid i sljedeću točku.

4.2.3 Odnos točke i poligona

Želimo utvrditi u kakvom je odnosu proizvoljna točka T i konveksan poligon – je li točka unutar poligona ili je izvan. To možemo napraviti sličnim postupkom kao i kod provjere orijentacije bridova. Potrebno je pogledati odnos svakog brida i zadane točke T. Isto tako je potrebno poznavati orijentaciju bridova poligona. Prema slikama 4.6a, 4.6b i 4.6c, kriterij glasi:

- Točka T je unutar poligona ako su vrhovi poligona zadani u smjeru kazaljke na satu i ako vrijedi:

$$(\forall i) T \cdot B_i \leq 0 \quad \text{za} \quad 1 \leq i \leq n.$$

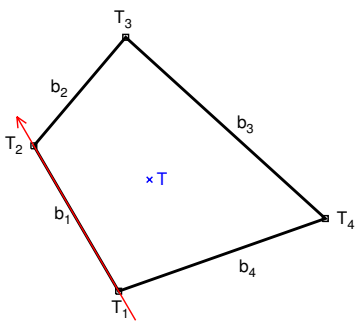
  Dakle, točka je unutar poligona ako su vrhovi poligona zadani u smjeru kazaljke na satu i ako točka je ispod *svakog* brida.

- Točka T je unutar poligona ako su vrhovi poligona zadani u smjeru suprotnom od smjera kazaljke na satu i ako vrijedi:

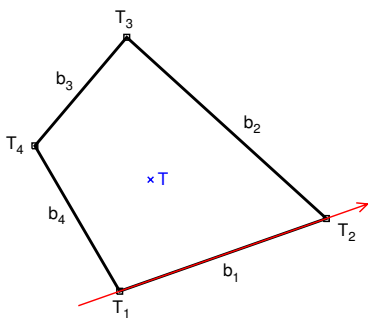
$$(\forall i) T \cdot B_i \geq 0 \quad \text{za} \quad 1 \leq i \leq n.$$

  Znači, točka je unutar poligona ako su vrhovi poligona zadani u smjeru suprotnom od smjera kazaljke na satu i točka je iznad *svakog* brida.

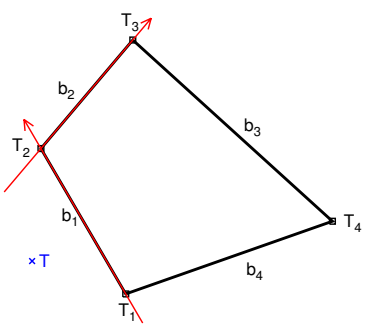
- I matematički "pametno" za kraj: točka T je izvan poligona ako nije unutra.



(a) Slučaj 1



(b) Slučaj 2



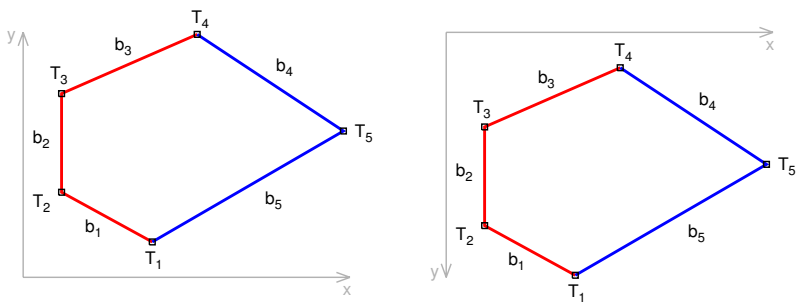
(c) Slučaj 3

Slika 4.6: Odnos točke i poligona

4.2.4 Bojanje konveksnog poligona

Postoji nekoliko načina za bojanje poligona tj. ispunjavanje poligona bojom. U nastavku ćemo opisati postupak s putujućom vodoravnom zrakom (engl. *scan-line*). Ideja je sljedeća: za svaki y pustit ćemo vodoravnu zraku i pratiti gdje se ona siječe s poligonom (tj. njegovim bridovima). Uz pretpostavku da je poligon konveksan, dobit ćemo niti jedno, jedno ili dva sjecišta. Ako postoje dva sjecišta, spojit ćemo ih vodoravnom linijom zadane boje. Da bismo osigurali da za svaku zraku sjecišta uvijek postoje, prije bojanja proći ćemo kroz sve vrhove poligona i zapamtiti najveću i najmanju y-koordinatu. S ovim podatkom vodoravnu zraku ćemo puštati za sve y-e od y_{min} do y_{max}. Budući da je zraka zapravo pravac $y = \text{konst}$, traženjem sjecišta sa svim zrakama čiji se y kreće od y_{min} do y_{max} proći ćemo cijeli poligon. Prilikom prolaska kroz točke korisno će biti zapamtiti i najmanju i najveću x-koordinatu.

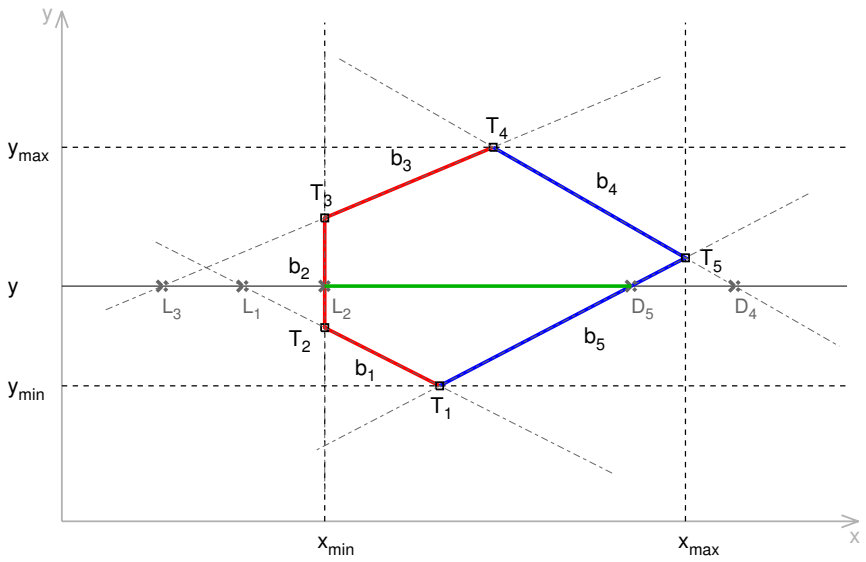
Opisana ideja čini se vrlo jednostavnom, no računski je dosta zahtjevna. Stoga ćemo uvesti dodatne olakšice i iskoristiti činjenicu da bojimo konveksan poligon. U tom slučaju ideja je sljedeća. Poligon možemo podijeliti na lijevi dio (gdje vrhovi rastu prema većim y vrijednostima) i desni dio (gdje vrhovi padaju prema nižim y vrijednostima) u slučaju matematičkog koordinatnog sustava (gdje y-os gleda prema gore, kao na slici 4.7a) te uz orijentaciju vrhova u smjeru kazaljke na satu. Slično vrijedi za slučaj ekranskog koordinatnog sustava (vidi sliku 4.7b) gdje lijevi dio čine bridovi kod kojih vrhovi padaju prema nižim y vrijednostima a desni dio bridovi kod kojih vrhovi rastu prema višim y vrijednostima.



(a) Slučaj kada os y gleda prema gore. (b) Slučaj kada os y gleda prema dolje.

Slika 4.7: Podjela bridova na lijeve i desne kod konveksnog poligona. Crvenom bojom su prikazani lijevi bridovi a plavom desni.

Tražit ćemo sjecišta s pravcima koje definiraju bridovi poligona, kako je to prikazano na slici 4.8, i pamtiti njihove x koordinate. y-koordinate već znamo – sve su iste i odgovaraju y-vrijednosti za koju promatramo vodoravnu zraku. Uočimo pri tome da, formalno govoreći, doista ne tražimo sjecišta zrake i bridova



Slika 4.8: Sjecišta vodoravne zrake i pravaca određenih bridovima konveksnog poligona

(na slici 4.8 brid b_3 i zraka nemaju sjecišta) već tražimo sjecišta pravaca na kojima bridovi leže i zrake (na slici 4.8 pravac koji odgovara bridu b_3 i zraka imaju sjecišta; to je L_3). Uočimo da, ako brid nije vodoravan, odgovarajući pravac i zraka uvijek će imati sjecište. Međutim, promatramo li samo sjecišta koja odgovaraju pravcima lijevih bridova, te ako analiziramo samo zrake $y_{min} \leq y \leq y_{max}$, tada najdesnije sjecište (tj. sjecište koje ima maksimalnu x-koordinatu) sigurno odgovara sjecištu poligona i zrake. Isto tako, promatramo li samo sjecišta koja odgovaraju pravcima desnih bridova, te ako analiziramo samo zrake $y_{min} \leq y \leq y_{max}$, tada najlijevije sjecište (tj. sjecište koje ima minimalnu x-koordinatu) sigurno odgovara sjecištu poligona i zrake. Razmislite zašto je tome tako.

Iz prethodnog opisa sada je jasno što treba pamtiti:

- za lijeve bridove pamtit ćemo ono sjecište koje ima najveću x-koordinatu i taj x označit ćemo s L, a
- za desne bridove pamtit ćemo ono sjecište koje ima najmanju x-koordinatu, i taj x označit ćemo s D.

Nakon što nađemo sjecište sa svim bridovima, iscrtat ćemo liniju između L i D koordinata na visini y.

Formalno bridove možemo podijeliti na lijeve i desne na sljedeći način. Neka je koordinatni sustav takav da os y gleda prema gore (vidi sliku 4.7a). Tada vrijedi:

- brid b_i određen vrhovima T_i i T_{i+1} je *lijevi* ako je $T_{i,y} < T_{i+1,y}$, a
- brid b_i određen vrhovima T_i i T_{i+1} je *desni* ako je $T_{i,y} > T_{i+1,y}$.

U slučaju da je koordinatni sustav takav da os y gleda prema dolje (vidi sliku 4.7b), lijevi bridovi su i dalje oni koji su i vizualno s lijeve strane. Ono što se mijenja jest odnos y-koordinata susjednih vrhova pa kriterij glasi:

- brid b_i određen vrhovima T_i i T_{i+1} je *lijevi* ako je $T_{i,y} > T_{i+1,y}$, a
- brid b_i određen vrhovima T_i i T_{i+1} je *desni* ako je $T_{i,y} < T_{i+1,y}$.

Primijetite također da u postupku nije nužno za svaku y-koordinatu tražiti sjecišta sa svim bridovima. Kod konveksnog poligona, za neki konkretan y postojat će samo jedan lijevi brid i jedan desni brid koji doista predstavljaju sjecišta sa promatranom vodoravnom zrakom. Nazovimo te bridove *aktivnim* bridovima. Sortiramo li lijeve i desne bridove u dvije liste po y-koordinatama, tada će slijedne vodoravne zrake koje dobivamo za y pa $y+1$, pa $y+2$ itd. najprije imati sjecišta s prvim lijevim bridom iz liste pa će nakon određenog broja koraka preći na drugi lijevi brid iz liste i tako redom. Štoviše, kako točno znamo y-koordinate početne i završne točke brida, trivijalno je napisati algoritam koji će u svakom trenutku točno znati koji mu je aktivni lijevi brid a koji desni i koji će sjecište tražiti samo s tim bridovima. Dodatno ubrzanje algoritma može se postići ako se uoči da su bridovi segmenti pravaca: stoga je trivijalno razraditi postupak koji sjecišta neće računati za svaki slijedni y već će ih inkrementalno generirati (kao što radi Bresenhamov algoritam prilikom rasterizacije linije). Parametri inkrementalnog generiranja mijenjat će se tek pri promjenama aktivnih bridova.

Struktura podataka za pamćenje konveksnog poligona

Pri opisu poligona rekli smo da poligon ima onoliko bridova koliko ima vrhova. Stoga za elementarni dio strukture koja pamti podatke o poligonu možemo uzeti pamćenje jednog vrha i jednog brida. Poligon će se tada sastojati od n ovakvih struktura. Kako vrhove poligona zadajemo u slikovnim elementima, za pamćenje koordinata mogu poslužiti cijeli brojevi. Isto tako s obzirom da jednadžbe bridova računamo bez dijeljenja, svi koeficijenti mogu biti cjelobrojni.

Za pamćenje točke u 2D s cjelobrojnim koordinatama koristit ćemo sljedeću strukturu.

```
typedef struct {
 int x;
 int y;
} iTocka2D;
```

Za pamćenje koeficijenata brida u 2D s cjelobrojnim koeficijentima koristit ćemo sljedeću strukturu.

```
typedef struct {
 int a;
 int b;
 int c;
} iBrid2D;
```

Za pamćenje elementarnog dijela poligona koristit ćemo sljedeću strukturu.

```
typedef struct {
 iTocka2D Vrh;
 iBrid2D Brid;
 int lijevi;
} iPolyElem;
```

Varijabla lijevi u strukturi iPolyElem služi za pohranu informacije je li brid lijevi. Kako je prikazan primjer iz programskog jezika C, elementu lijevi pridružen je tip **int** u nedostatku prikladnijeg tipa. Tip koji bi najbolje odgovarao bio bi boolean koji u C-u ne postoji. U našem slučaju stoga će varijabla imati vrijednost 0 ako je brid desni a različitu od 0 ako je lijevi.

4.2.5 Funkcije za rad s poligonima

Krenimo s najjednostavnijom funkcijom čiji je zadatak nacrtati poligon na zaslonu. Za crtanje bridova koristit ćemo prethodno napisanu funkciju koja taj postupak radi uporabom Bresenhamovog algoritma.

```
void CrtajPoligonKonv(iPolyElem *polel, int n) {
    int i,i0;

    i0 = n-1;
    for( i = 0; i < n; i++ ) {
        bresenham_nacrtaj_cjelobrojni(
            polel[i0].Vrh.x, polel[i0].Vrh.y,
            polel[i].Vrh.x,  polel[i].Vrh.y);
        i0 = i;
    }
}
```

Funkcija jednostavno spaja dva po dva vrha u smjeru kako su zadani. Spaja se i završni i početni vrh kako bi se poligon zatvorio. Argumenti funkcije su polje elemenata poligona te broj elemenata polja.

Funkcija koja računa koeficijente poligona prikazana je u nastavku.

```
void RacunajKoefPoligonKonv(iPolyElem *polel, int n) {
    int i,i0;

    i0 = n-1;
    for( i = 0; i < n; i++ ) {
        polel[i0].Brid.a = polel[i0].Vrh.y-polel[i].Vrh.y;
        polel[i0].Brid.b = -(polel[i0].Vrh.x-polel[i].Vrh.x);
        polel[i0].Brid.c = polel[i0].Vrh.x*polel[i].Vrh.y
                         - polel[i0].Vrh.y*polel[i].Vrh.x;
        polel[i0].lijevi = polel[i0].Vrh.y < polel[i].Vrh.y;
        i0 = i;
    }
}
```

Funkcija računa koeficijente bridova radeći pri tome vektorski produkt dva po dva vrha. Jednadžba se dobije u homogenom prostoru, a točke se u homogene pretvaraju proširivanjem s homogenim parametrom iznosa 1. Dodatno se brid klasificira kao lijevi ako je zadan vrhovima od kojih je prvi ispod drugoga. Ova klasifikacija bit će ispravna samo ako je poligon zadan tako da su mu vrhovi u smjeru kazaljke na satu. Ako su vrhovi zadani suprotno, tada će postavljena zastavica lijevi zapravo označavati da je brid desni.

Funkcija čiji je zadatak obojati konveksni poligon prikazana je u nastavku.

```
void PopuniPoligonKonv(iPolyElem *polel, int n) {
    int i,i0,y;
    int xmin, xmax, ymin, ymax;
    double L,D,x;

    /* Trazenje minimalnih i maksimalnih koordinata */
    xmin = xmax = polel[0].Vrh.x;
    ymin = ymax = polel[0].Vrh.y;
    for( i = 1; i < n; i++ ) {
        if( xmin > polel[i].Vrh.x ) xmin = polel[i].Vrh.x;
        if( xmax < polel[i].Vrh.x ) xmax = polel[i].Vrh.x;
        if( ymin > polel[i].Vrh.y ) ymin = polel[i].Vrh.y;
        if( ymax < polel[i].Vrh.y ) ymax = polel[i].Vrh.y;
    }

    /* Bojanje poligona: za svaki y izmedu ymin i ymax radi... */
    for( y = ymin; y<=ymax; y++ ) {
        /* Pronadi najvece lijevo i najmanje desno sjeciste... */
        L = xmin; D = xmax;
        i0=n-1;
        /* i0 je pocetak brida, i je kraj brida */
        for( i = 0; i < n; i0=i++ ) {
            /* ako je brid vodoravan*/
            if(polel[i0].Brid.a==0.) {
                if(polel[i0].Vrh.y == y) {
```

```
                    if(polel[i0].Vrh.x<polel[i].Vrh.x) {
                        L = polel[i0].Vrh.x;
                        D = polel[i].Vrh.x;
                    } else {
                        L = polel[i].Vrh.x;
                        D = polel[i0].Vrh.x;
                    }
                    break;
                }
            } else { /* inace je regularan brid, nadi sjeciste */
                x = (-polel[i0].Brid.b*y-polel[i0].Brid.c) /
                    (double)polel[i0].Brid.a;
                if(polel[i0].lijevi) {
                    if( L < x ) L = x;
                } else {
                    if( D > x ) D = x;
                }
            }
        }
        bresenham_nacrtaj_cjelobrojni(zaokruzi(L),y,zaokruzi(D),y);
    }
}
```

Funkcija je implementacija prethodno opisanog postupka. Funkcija očekuje da su koeficijenti bridova već izračunati. Pogledajmo trenutak kako se računaju sjecišta s bridovima. Postoje dva slučaja opisana u nastavku.

- Brid je vodoravan (i zraka je vodoravna) te postoji opasnost da se brid i zraka sijeku u puno točaka. Brid je vodoravan ako mu je koeficijent a jednak nuli. Tada opet imamo dva slučaja: ili se brid i zraka uopće ne sijeku jer su na različitim y-ima ili se sijeku u svim točkama jer su na istim y-ima. Ako se ne sijeku, tada jednostavno preskačemo daljnju analizu i nastavljamo s obradom sljedećeg brida. Ako se sijeku, tada kao lijevo sjecište uzimamo x-koordinatu onog vrha kod kojeg je ona manja, a kao desno sjecište uzimamo x-koordinatu onog vrha kod kojeg je ona veća. Zatim prestajemo s danjom analizom sjecišta (jer kod konveksnog poligona ona ionako ne postoje) i crtamo liniju određenu pronađenim lijevim i desnim sjecištem.

- Brid nije vodoravan te postoji točno jedno sjecište. Tada pronalazimo to sjecište i ovisno o tome je li brid lijevi ili desni, pamtimo to sjecište kao lijevo ili desno (odnosno pamtimo ga samo ako je lijevo i veće od trenutno zapamćenog lijevog, odnosno ako je desno i manje od trenutno zapamćenog desnog).

Evo za kraj još i funkcije čiji je zadatak utvrditi je li poligon konveksan te koja je orijentacija vrhova poligona.

```
void ProvjeriPoligonKonv(
                iPolyElem *polel, int n, int *konv, int *orij) {
    int i,i0,r;
    int iznad, ispod, na;

    ispod = iznad = na = 0;
    i0 = n-2;
    for( i = 0; i < n; i++,i0++ ) {
        if(i0>=n) i0=0;
        r = polel[i0].Brid.a*polel[i].Vrh.x +
                    polel[i0].Brid.b*polel[i].Vrh.y +
                    polel[i0].Brid.c;
        if( r == 0 ) na++;
        else if( r > 0 ) iznad++;
        else ispod++;
    }
    *konv = 0; *orij = 0;
    if( ispod == 0 ) {
        *konv = 1;
    } else if( iznad == 0 ) {
        *konv = 1; *orij = 1;
    }
}
```

Funkcija se zasniva na brojanju koliko vrhova leži iznad odgovarajućih bridova, koliko ispod a koliko na njima. Podatak koliko vrhova leži na odgovarajućim bridova redundantan je i trebao bi biti nula. Zaključivanje je sljedeće:

- ispod = 0
  Poligon je konveksan jer su tada svi vrhovi iznad odgovarajućih bridova. Orijentacija je u smjeru suprotnom od smjera kazaljke na satu.

- iznad = 0
  Poligon je konveksan jer su tada svi vrhovi ispod odgovarajućih bridova. Orijentacija je u smjeru kazaljke na satu.

- iznad != 0 && ispod != 0
  Neki su vrhovi iznad a neki ispod odgovarajućih bridova. Poligon je konkavan i informaciju o orijentaciji u varijabli orij treba ignorirati.

4.3 Ponavljanje

1. Izvedite osnovni Bresenhamov algoritam koji radi s decimalnim brojevima i koji je ograničen na crtanje linija čiji je kut od 0° do 45°.

2. Kakav će prikaz generirati prethodni Bresenhamov algoritam ako mu se zada linijski segment koji je pod kutem:

   (a) između 45° i 90°,
   (b) između 90° i 270°,
   (c) između 0° i -45°?

   Zadajte si neki konkretan primjer pa pokušajte nacrtati rezultantnu sliku.

3. Kako se Bresenhamov algoritam koji radi s decimalnim brojevima prevodi u inačicu koja radi s cijelim brojevima?

4. Kako se na temelju osnovnog Bresenhamovog algoritma koji radi za 0° do 45° dobiva algoritam koji korektno crta sve linije?

5. Definirajte što je to konveksni poligon?

6. Kako možemo ispitati orijentaciju vrhova konveksnog poligona?

7. Kako možemo odrediti je li neka zadana točna izvan poligona, na rubu poligona ili pak unutar poligona?

8. Na koji se način radi bojanje konveksnog poligona?

9. Razmislite što će biti rezultat rada prethodnog algoritma ako mu kao ulaz date konkavni poligon. Zadajte si neki konkretan primjer konkavnog poligona pa utvrdite koji bi bio rezultat njegova bojanja.

Poglavlje 5

Osnovne geometrijske transformacije

Matrični račun temeljni je alat linearne algebre – velikog područja matematike koje je svoju primjenu našlo i u računalnoj grafici. Prilikom izrade kompleksnih scena, često smo u situaciji da neko tijelo želimo pomaknuti u prostoru, promijeniti mu veličinu ili ga zarotirati oko neke točke. Uporabom matričnog računa i homogenih koordinata svaka se od spomenutih operacija može prikazati kao jedna matrica. Izvođenje takve operacije svodi se na matrično množenje – točku koju želimo pomaknuti u prostoru jednostavno pomnožimo matricom translacije, točku koju želimo rotirati pomnožimo matricom rotacije i slično. Iz ovog razloga matrični je prikaz ovih operacija izuzetno pogodan u grafičkim aplikacijama, i toliko je čest da zapravo čini temelj izvođenja grafičkih operacija i jezgru svih popularnih biblioteka za rad s grafikom, uključivo *OpenGL* i *DirectX*.

U ovom poglavlju najprije ćemo pogledati kako se definiraju spomenute operacije u 2D prostoru, nakon čega ćemo rezultate poopćiti na 3D prostor. Pri tome ćemo čitavo vrijeme raditi s homogenim koordinatama točaka. U 2D slučaju to znači da će koordinate imati 3 komponente, i pripadne matrice bit će kvadratne, dimenzija 3×3. U 3D slučaju koordinate će imati 4 komponente, i pripadne matrice bit će kvadratne, dimenzija 4×4.

Djelovanje nekog operatora na točku T_h rezultirat će točkom $T_h^{'}$. Ovo djelovanje možemo iskazati na dva jednakovrijedna načina. Prvi način jest množenjem jednoretčane matrice točke i matrice operatora $\mathbf{\Psi}$, kako je prikazano izrazom (5.1).

$$T_h^{'} = T_h \cdot \mathbf{\Psi}. \tag{5.1}$$

Drugi način jest množenjem matrice operatora $\mathbf{\Omega}$ i jednostupčane matrice promatrane točke, kao što to prikazuje izraz (5.2).

$$T_h^{'} = \mathbf{\Omega} \cdot T_h. \tag{5.2}$$

Ovisno o konvenciji s kojom radimo (množimo li točku s matricom operatora ili matricu operatora s točkom), matrice operatora bit će različite, iako se poznavanjem jedne može doći do druge. Naime, vrijedit će:

$$\mathbf{\Omega}^{\mathrm{T}} = \mathbf{\Psi}. \tag{5.3}$$

No, u računalnoj grafici je izuzetno rijetka situacija da koristimo samo jednu transformaciju. Puno je češća situacija primjene slijeda transformacija koji generira konačnu poziciju točke. Primjerice, točku A najprije želimo translatirati čime dobivamo točku B. Potom, tu točku želimo rotirati, čime dobivamo točku C. Potom tako dobivenu točku želimo još jednom translatirati, čime ćemo dobiti točku D, i konačno točku još želimo skalirati čime ćemo dobiti konačnu točku E. Ovaj postupak preslikavanja točke A u točku E skiciran je u nastavku.

$$A \xrightarrow{\text{oper}_1} B \xrightarrow{\text{oper}_2} C \xrightarrow{\text{oper}_3} D \xrightarrow{\text{oper}_4} E$$

Koristimo li konvenciju prikazanu izrazom (5.1), možemo pisati:

$$\begin{aligned}
B &= A \cdot \mathbf{\Psi}_1 \\
C &= B \cdot \mathbf{\Psi}_2 = A \cdot \mathbf{\Psi}_1 \cdot \mathbf{\Psi}_2 \\
D &= C \cdot \mathbf{\Psi}_3 = A \cdot \mathbf{\Psi}_1 \cdot \mathbf{\Psi}_2 \cdot \mathbf{\Psi}_3 \\
E &= D \cdot \mathbf{\Psi}_4 = A \cdot \mathbf{\Psi}_1 \cdot \mathbf{\Psi}_2 \cdot \mathbf{\Psi}_3 \cdot \mathbf{\Psi}_4
\end{aligned}$$

Zbirni operator koji obavlja sve navedene transformacije tada možemo prikazati matricom $\mathbf{\Psi}$ koja je naprosto umnožak matrica koje obavljaju željene transformacije:

$$\mathbf{\Psi} = \mathbf{\Psi}_1 \cdot \mathbf{\Psi}_2 \cdot \mathbf{\Psi}_3 \cdot \mathbf{\Psi}_4.$$

U slučaju da se odlučimo za konvenciju definiranu izrazom (5.2), možemo pisati:

$$\begin{aligned}
B &= \mathbf{\Omega}_1 \cdot A \\
C &= \mathbf{\Omega}_2 \cdot B = \mathbf{\Omega}_2 \cdot \mathbf{\Omega}_1 \cdot A \\
D &= \mathbf{\Omega}_3 \cdot C = \mathbf{\Omega}_3 \cdot \mathbf{\Omega}_2 \cdot \mathbf{\Omega}_1 \cdot A \\
E &= \mathbf{\Omega}_4 \cdot D = \mathbf{\Omega}_4 \cdot \mathbf{\Omega}_3 \cdot \mathbf{\Omega}_2 \cdot \mathbf{\Omega}_1 \cdot A
\end{aligned}$$

Zbirni operator koji obavlja sve navedene transformacije tada možemo prikazati matricom $\mathbf{\Omega}$ koja je umnožak matrica koje obavljaju željene transformacije, ali obrnutim redoslijedom. Dakle,

$$\mathbf{\Omega} = \mathbf{\Omega}_4 \cdot \mathbf{\Omega}_3 \cdot \mathbf{\Omega}_2 \cdot \mathbf{\Omega}_1.$$

Uočite sada i kako, temeljem izraza (5.3) vrijedi:

$$\begin{aligned}\boldsymbol{\Psi} &= \boldsymbol{\Omega}^{\mathrm{T}} \\ &= (\boldsymbol{\Omega}_4 \cdot \boldsymbol{\Omega}_3 \cdot \boldsymbol{\Omega}_2 \cdot \boldsymbol{\Omega}_1)^{\mathrm{T}} \\ &= \boldsymbol{\Omega}_1^{\mathrm{T}} \cdot \boldsymbol{\Omega}_2^{\mathrm{T}} \cdot \boldsymbol{\Omega}_3^{\mathrm{T}} \cdot \boldsymbol{\Omega}_4^{\mathrm{T}} \\ &= \boldsymbol{\Psi}_1 \cdot \boldsymbol{\Psi}_2 \cdot \boldsymbol{\Psi}_3 \cdot \boldsymbol{\Psi}_4.\end{aligned}$$

Pogledajmo sada jednostavan primjer. Neka nam je na raspolaganju biblioteka koja za svaku od ovih transformacija nudi prikladnu naredbu koja modificira trenutnu matricu na način da ju pomnoži željenom matricom (s desna) i rezultat postavi kao novu trenutnu matricu. Stavimo li se u ulogu programera, pogledajmo čime će rezultirati program u kojem naredbe zadajemo redoslijedom kojim želimo da se obave opisane transformacije.

Naredba	Djelovanje
`identity();`	`M = I`
`translate(x0,y0);`	`M *= M1 ==> M = M1`
`rotate(angle);`	`M *= M2 ==> M = M1*M2`
`translate(x1,y1);`	`M *= M3 ==> M = M1*M2*M3`
`scale(k1,k2);`	`M *= M4 ==> M = M1*M2*M3*M4`

Ako koristimo konvenciju množenja točke s matricom, rezultat će biti upravo u skladu s očekivanim. Naime, predanu točku najprije će množiti matrica M_1, potom će rezultat pomnožiti matrica M_2, taj rezultat pomnožit će matrica M_3 i na kraju će sve pomnožiti matrica M_4. Koristimo li međutim konvenciju množenja matrice s točkom, rezultat više neće biti korektan. Naime, u tom slučaju prvo se s točkom množi matrica M_4; potom matrica M_3 množi tako dobiveni rezultat, i tako sve do matrice M_1. Koji je zaključak? Ako se koristi konvencija množenja matrice s točkom, prilikom pisanja programa naredbe koje obavljaju željene transformacije moramo pisati obrnutim redosljedom od onoga kojim želimo da se te transformacije doista i obave.

U nastavku ovog poglavlja (a i knjige) koristit ćemo upravo konvenciju množenja točke matricom, kako je definirano izrazom (5.1). Međutim, biblioteka *OpenGL* koja nudi programsko sučelje koje radi baš kao u prethodnom jednostavnom primjeru koristi konvenciju definiranu izrazom (5.2) – matricu operatora množi točkom. Kod početnika u *OpenGL*-u ovo može rezultirati problemima i frustracijom, jer početnici često zaborave da naredbe koje obavljaju željene transformacije moraju pisati obrnutim redosljedom. U tom smislu, program koji obavlja navedene transformacije morao bi biti napisan kako slijedi u nastavku.

Naredba	Djelovanje
`identity();`	`M = I`
`scale(k1,k2);`	`M *= M4 ==> M = M4`
`translate(x1,y1);`	`M *= M3 ==> M = M4*M3`
`rotate(angle);`	`M *= M2 ==> M = M4*M3*M2`
`translate(x0,y0);`	`M *= M1 ==> M = M4*M3*M2*M1`

Osvrnimo se i na svojstva kompozicije transformacija. Budući da se djelovanje operatora opisuje matričnim množenjem, lako se može doći do sljedećeg zaključka: ako na jednu točku djeluje više transformacija, tada je bitan redoslijed djelovanja transformacija. Dokaz ove tvrdnje je trivijalan. Naime, matrično množenje nije komutativno pa se za različite redoslijede množenja matrica dobivaju različiti rezultati.

Djelovanje više transformacija može se zapisati:

$$T_h^{'} = T_h \cdot \mathbf{\Psi}_1 \cdot \mathbf{\Psi}_2 \cdots \mathbf{\Psi}_n = T_h \cdot \mathbf{\Psi}^{'}$$

$$\mathbf{\Psi}^{'} = \mathbf{\Psi}_1 \cdot \mathbf{\Psi}_2 \cdots \mathbf{\Psi}_n$$

pri čemu na točku prvo djeluje operator $\mathbf{\Psi}_1$, pa $\mathbf{\Psi}_2$ itd.

Postoji i iznimka gornjem pravilu: unutar jedne vrste transformacija, redoslijed djelovanja pojedine transformacije nije bitan. Ovo se može i matematički dokazati, no ostanimo na logičkom dokazu: rotiramo li točku za kut α pa za kut β, dobit ćemo isti rezultat kao i da rotiramo točku najprije za kut β pa onda za kut α.

Pojasnimo još nekoliko pojmova koji se vežu uz transformacije, kao što su Euklidske transformacije, afine transformacije te projektivne transformacije.

5.1 Vrste transformacija

Kako su različite vrste transformacija od velikog značaja za računalnu grafiku, u ovom odjeljku upoznat ćemo se s podjelom transformacija. Rigorozan matematički tretman transformacija spada u područje linearne algebre; stoga ćemo se ovdje pokušati zadržati na manje formalnoj razini.

U računalnoj grafici jedan od osnovnih pojmova je točka, koju tipično poistovjećujemo s pripadnim radij-vektorom. Pri tome točke promatramo ili u dvodimenzionalnom radnom prostoru ili u trodimenzionalnom radnom prostoru, pa govorimo o dvo- ili trokomponentnim vektorima. Nad tim vektorima radit ćemo niz transformacija poput translacije, rotacije, povećanja, sažimanja, smika itd. Djelovanje same transformacije opisivat ćemo pripadnom kvadratnom matricom, kako smo već opisali u ovom poglavlju. Pa krenimo redom.

Promatrajmo n-dimenzionalan radni prostor $\mathbf{V}$. *Linearna transformacija* T je transformacija koja zadovoljava sljedeća svojstva.

1. Čuvanje zbrajanja vektora. Vrijedi: $T(v_1+v_2) = T(v_1)+T(v_2)$ i to $\forall v_1, v_2 \in \mathbf{V}$.

2. Čuvanje množenja sa skalarom. Vrijedi: $T(\alpha \cdot v) = \alpha \cdot T(v)$ i to $\forall v \in \mathbf{V}, \forall \alpha \in \mathbf{R}$.

Označimo li s $\mathbf{M}$ kvadratnu matricu ranga n koja predstavlja promatranu transformaciju, možemo pisati:

- $(v_1 + v_2) \cdot \mathbf{M} = v_1 \cdot \mathbf{M} + v_2 \cdot \mathbf{M}$, te
- $(\alpha \cdot v) \cdot \mathbf{M} = \alpha \cdot (v \cdot \mathbf{M})$.

Transformacije rotacije, skaliranja i smika pripadaju u linearne tranformacije. Translacija se, međutim, ne može zapisati kao linearna tranformacija u radnom prostoru. Naime, iz prethodna dva zahtjeva slijedi da se nul-vektor (ishodište) uvijek preslikava u nul-vektor, tj $T(0) = 0$ (provjerite). Stoga transformacija koja vektor pomiče za konstantan pomak ne može biti linearna.

Afine transformacije su transformacije koje čuvaju kolinearnost i omjere udaljenosti, i čine općenitiju kategoriju od linearnih transformacija. Ovo prvo (čuvanje kolinearnosti) ima za posljedicu da će tri različite točke A, B i C koje su u originalnom prostoru kolinearne i nakon transformacije u točke A', B' i C' ostati kolinearne (ležat će na istom pravcu). Drugo svojstvo nam govori i da će omjeri udaljenosti tih točaka ostati nepromijenjeni, tj: $d(A,B)/d(A,C) = d(A',B')/d(A',C')$, gdje je $d(X,Y)$ funkcija koja vraća udaljenost točaka X i Y. Afine transformacije čuvaju i paralelnost. Linije koje su u originalnom prostoru paralelne ostat će paralelne i nakon transformacije. Međutim, afine transformacije ne čuvaju kuteve niti stvarne duljine. Naime, afinom transformacijom bilo koji trokut moguće je preslikati u bilo koji drugi trokut – posljedica je da se kutevi u trokutu kao niti duljine stranica ne moraju očuvati.

Svaka se afina transformacija može zapisati u obliku:

$$T(v) = L(v) + p$$

gdje je L linearna transformacija a p vektor pomaka. Kompozicija afinih transformacija opet se može prikazati kao jedna afina transformacija. Afine transformacije obuhvaćaju sve linearne transformacije kao i neke druge, primjerice pomak.

Euklidske transformacije su posebna vrsta afinih transformacija. Naime, Euklidska transformacija je afina transformacija oblika $E(v) = L(v) + p$, gdje je L ortogonalna linearna transformacija. Euklidska transformacija kao posebna vrsta afine transformacije ima sva svojstva afine transformacije. Dodatno, kako je $E(u) - E(v) = L(u - v)$ i kako je L ortogonalna linearna transformacija, E čuva duljine segmenata i kuteve između dva linijska segmenta. Drugim riječima,

pravokutni trokut duljina stranica 9, 12 i 15 i nakon Euklidske transformacije ostat će pravokutan, s upravo tim duljinama stranice. Kako bi ova svojstva bila zadovoljena, na opći oblik afine transformacije već smo postavili jedno ograničene – L mora biti ortogonalna linearna transformacija. Postavljanjem daljnjih ograničenja na L i p dolazi se do tri moguće vrste Euklidske transformacije.

1. Translacija – dobije se ako je $L = I$ (tj. kada je matrica linearne translacije jedinična matrica). Tada ostaje $T(v) = v + p$, što odgovara pomaku točke za fiksni iznos.

2. Rotacija – dobije se ako je $p = 0$ i ako je dodatno $det(L) = 1$. Svaka ortogonalna matrica čija je determinanta jednaka 1 predstavlja transformaciju rotacije.

3. Refleksija – dobije se ako je $p = 0$ a $det(L) = -1$. Primjerice, u 2D prostoru matrica
$$\begin{bmatrix} \cos(\phi) & \sin(\phi) \\ \sin(\phi) & -\cos(\phi) \end{bmatrix}$$
zrcali svaki vektor oko pravca određenog jednadžbom $y = x \cdot \mathrm{tg}\,(\frac{\phi}{2})$.

Kako smo se ovdje oslonili na pojam ortogonalne (tj. ortonormirana) transformacije, recimo da je to transformacija čija je matrica ortogonalna, a to znači da za takvu matricu A vrijedi:

- $A \cdot A^T = I$,
- $A^{-1} = A^T$,
- ortogonalne matrice čuvaju skalarni produkt: $v \cdot w = Av \cdot Aw$,
- determinanta od A je 1 ili -1 te
- retci matrice su ortonormirani vektori, tj. jedinični su (norme 1) i međusobno su okomiti (skalarni produkt im je 0); isto vrijedi i za stupce.

Konačno, s obzirom da je izvorni zapis afine transformacije:

$$T(v) = L \cdot v + p$$

nepraktičan za direktnu primjenu na računalu, sve transformacije najčešće se provode u *homogenom* prostoru. U tom slučaju čitava se afina transformacija može prikazati kao jedna kvadratna matrica. Uz gore prikazanu konvenciju množenja matrice i točke gdje je matrica L kvadratna ranga n i gdje je točka v zapravo $n + 1$-komponentni vektor (jednostupčana matrica) koji u homogenom prostoru predstavlja točku radnog n-dimenzionalnog prostora, odgovarajuća matrica afine transformacije je oblika:

$$\begin{bmatrix} L & p \\ 0\cdots 0 & 1 \end{bmatrix}.$$

U slučaju da se koristi konvencija množenja točke s matricom, $T(v) = v \cdot L + p$, tj. ako je točka v zapravo $n+1$-komponentni vektor (jednoretčana matrica) koji u homogenom prostoru predstavlja točku radnog n-dimenzionalnog prostora, odgovarajuća matrica afine transformacije je oblika:

$$\begin{bmatrix} L & 0 \\ p & 1 \end{bmatrix}.$$

Konačno, osvrnimo se i na još općenitiji razred transformacija – *projektivne transformacije*. Projektivne transformacije opisujemo matricama u homogenom prostoru, pa je tako opći oblik projektivne matrice koja radi projekciju u 3D radnom prostoru opisan kvadratnom matricom ranga 4:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix}.$$

Ovo nije afina transformacije jer nema ograničenja koje smo postavili nad afine matrice. U nastavku ovog poglavlja obradit ćemo najčešće korištene afine transformacije, a u sljedećem poglavlju govorit ćemo o projekcijama.

5.2 2D transformacije

5.2.1 Uvod

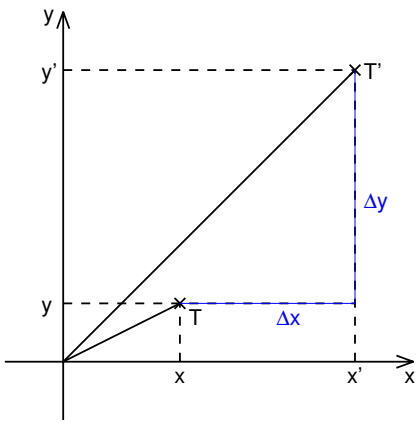
Postoji nekoliko elementarnih transformacija koje djeluju nad točkom. Kada se ta točka nalazi u 2D prostoru, govorimo o 2D-transformacijama. U nastavku ćemo obraditi sljedeće elementarne transformacije:

- translaciju,
- rotaciju,
- skaliranje te
- smik.

Koristit ćemo točke u homogenom prostoru, a transformacije ćemo prikazivati u matričnom obliku. Svaka elementarna transformacija bit će predstavljena kao jedan operator, kome će odgovarati kvadratna matrica. U 2D prostoru ovi operatori imaju kvadratne matrice reda 3, budući da se točke zapisuju kao jednoretčane matrice s tri stupca $[\ x \quad y \quad z\ ]$.

5.2.2 Translacija

Translacija je transformacija koja svakoj komponenti točke u radnom prostoru dodaje određeni pomak. Primjer translacije prikazan je na slici 5.1. Točka T translatira se u točku T'.



Slika 5.1: Translacija točke

Postupak translacije može se opisati sljedećim jednadžbama u radnom prostoru:

$$T_x^{'} = T_x + \Delta_x$$
$$T_y^{'} = T_y + \Delta_y$$

gdje su T_x i T_y komponente točke T a $T_x^{'}$ i $T_y^{'}$ komponente točke $T^{'}$. Iskoristimo li vezu između radnih i homogenih koordinata:

$$T_x = \frac{T_{h,x}}{T_{h,h}} \qquad T_y = \frac{T_{h,y}}{T_{h,h}}$$

dobiva se:

$$T_{h,x}^{'} = T_{h,x} + \Delta_x \cdot T_{h,h}$$
$$T_{h,y}^{'} = T_{h,y} + \Delta_y \cdot T_{h,h}$$

pri čemu su $T_{h,x}$ i $T_{h,y}$ komponente homogenog zapisa točke T (odnosno tada T_h), dok je $T_{h,h}$ homogeni parametar.

Prethodne relacije pokazuju da se točka $T_h^{'}$ može dobiti matričnim množenjem točke T_h operatorom translacije:

$$\begin{bmatrix} T'_{h,x} & T'_{h,y} & T'_{h,h} \end{bmatrix} = \begin{bmatrix} T_{h,x} & T_{h,y} & T_{h,h} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta_x & \Delta_y & 1 \end{bmatrix}$$

pa se operator translacije matrično zapisuje kao:

$$\mathbf{\Psi}_{tr} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta_x & \Delta_y & 1 \end{bmatrix}.$$

Inverzni operator ovom operatoru je operator translacije za negativan pomak:

$$\mathbf{\Psi}_{tr}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\Delta_x & -\Delta_y & 1 \end{bmatrix},$$

jer vrijedi:

$$\mathbf{\Psi}_{tr} \cdot \mathbf{\Psi}_{tr}^{-1} = \mathbf{\Psi}_{tr}^{-1} \cdot \mathbf{\Psi}_{tr} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

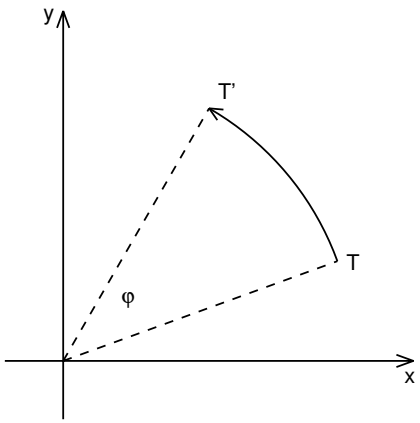
5.2.3 Rotacija

Rotacija je transformacija koja točku rotira oko ishodišta za zadani kut ϕ. Smjer rotacije može biti podudaran sa smjerom kazaljke na satu, ili suprotan od smjera kazaljke na satu. Kako je matematički "pozitivan" smjer rotacije definiran kao smjer suprotan od smjera kazaljke na satu, tako će u nastavku biti izveden operator koji rotira točku u smjeru suprotnom od kazaljke na satu. Ako se želi rotirati u smjeru kazaljke na satu, dovoljno je umjesto kuta ϕ rotirati za smjer $-\phi$. Primjer rotacije prikazan je na slici 5.2.

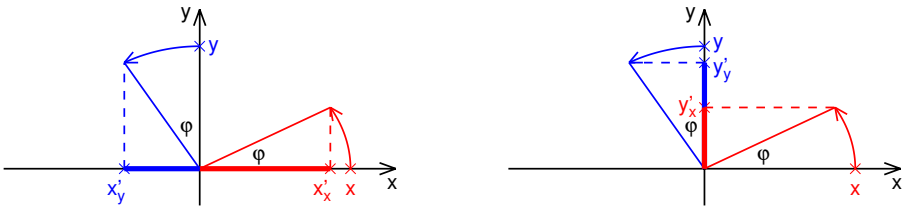
Da bismo izveli matricu operatora rotacije, potrebno se prisjetiti linearne algebre, gdje smo naučili da se djelovanje linearnog operatora na vektor (točku) može zapisati kao zbroj djelovanja operatora na svaku komponentu tog vektora. Budući da izvodimo rotaciju u ravnini, imamo dvije komponente: komponentu u smjeru osi x, i komponentu u smjeru osi y. Djelovanje operatora na te dvije komponente prikazano je na slici 5.3.

Pogledajmo najprije sliku 5.3.a. Ako točku koja leži na osi x zarotiramo za kut ϕ, dobiti ćemo točku čija je x'_x koordinata jednaka $x'_x = x \cdot \cos(\phi)$. Ako točku koja leži na osi y zarotiramo za kut ϕ, dobiti ćemo točku čija je x'_y koordinata jednaka: $x'_y = -y \cdot \sin(\phi)$. Tada djelovanje operatora na točku daje x' komponentu jednaku sumi ova dva djelovanja:

$$x' = x \cdot \cos(\phi) - y \cdot \sin(\phi).$$



Slika 5.2: Rotacija točke



(a) Utjecaj rotacije na komponentu x

(b) Utjecaj rotacije na komponentu y

Slika 5.3: Rotacija točke: utjecaj na pojedine komponente

Da bismo utvrdili kako se tvori $y^{'}$ komponenta točke, pogledajmo djelovanje operatora opet na točke $(x, 0)$ i $(0, y)$. Međutim, sada moramo pratiti što se događa s y-projekcijama nastalih točaka, kao što je prikazano na slici 5.3.b. Djelovanjem na $(x, 0)$ dobiva se $y^{'}_{x} = x \cdot \sin\phi$ a djelovanjem na $(0, y)$ dobiva se $y^{'}_{y} = y \cdot \cos(\phi)$. Ukupno djelovanje daje:

$$y^{'} = x \cdot \sin(\phi) + y \cdot \cos(\phi).$$

Zapisano u matričnom obliku u homogenim koordinatama dobiva se:

$$\begin{bmatrix} T'_{h,x} & T'_{h,y} & T'_{h,h} \end{bmatrix} = \begin{bmatrix} T_{h,x} & T_{h,y} & T_{h,h} \end{bmatrix} \cdot \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

pa se operator rotacije zapisuje:

$$\mathbf{\Psi}_{rot} = \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Ako želimo operator koji rotira u smjeru kazaljke na satu, dovoljno je umjesto kuta ϕ rotirati za kut $-\phi$, pa se uvrštavanjem u matricu operatora i uzimanjem u obzir parnosti funkcije cos i neparnosti funkcije sin dobiva operator:

$$\mathbf{\Psi}'_{rot} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Korištenjem ovog operatora uz pozitivne vrijednosti kuta ϕ dobiti ćemo rotaciju u smjeru kazaljke na satu.

Odmah se može uočiti da se operatori $\mathbf{\Psi}_{rot}$ i $\mathbf{\Psi}'_{rot}$ međusobno poništavaju u djelovanju te su inverzi jedan drugome. Naime, ako pogledamo operator koji opisuje djelovanje oba operatora, dobiva se:

$$\mathbf{\Psi}_{rot} \cdot \mathbf{\Psi}'_{rot} = \mathbf{\Psi}'_{rot} \cdot \mathbf{\Psi}_{rot} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

iz čega je vidljivo da slijedno djelovanje jednog pa drugog operatora vraća točku u prvobitni položaj.

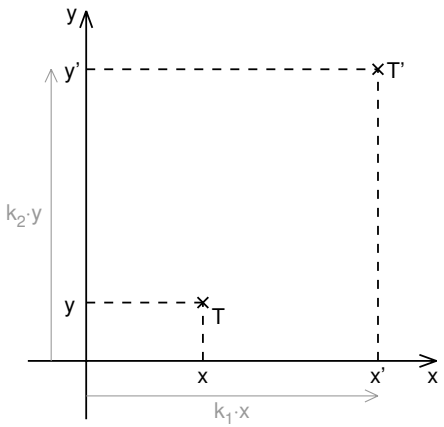
5.2.4 Skaliranje

Skaliranje je transformacija koja "skalira" (rasteže ili steže) svaku komponentu točke. Pri tome je skaliranje svake komponente određeno faktorom skaliranja, pri čemu faktori ne moraju biti isti za sve komponente. Ako je to slučaj, tada govorimo o neproporcionalnom skaliranju. Ako su faktori skaliranja jednaki za sve komponente, tada govorimo o proporcionalnom skaliranju. Primjer skaliranja prikazan je na slici 5.4.

Djelovanje operatora po komponentama prikazano je na slici 5.5.

Vrijedi:

$$x' = k_1 \cdot x$$



Slika 5.4: Skaliranje točke

$$y' = k_2 \cdot y$$

odnosno prelaskom na homogene koordinate:

$$\begin{bmatrix} T'_{h,x} & T'_{h,y} & T'_{h,h} \end{bmatrix} = \begin{bmatrix} T_{h,x} & T_{h,y} & T_{h,h} \end{bmatrix} \cdot \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

pa se operator skaliranja zapisuje:

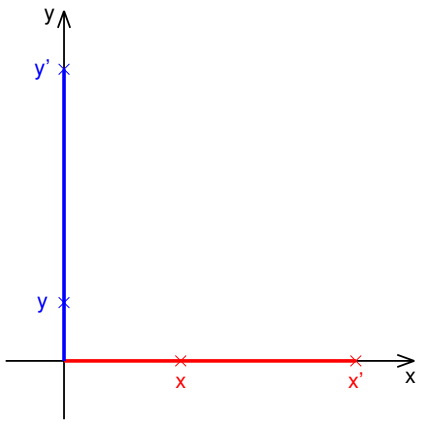
$$\mathbf{\Psi}_{skal} = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Ako želimo proporcionalno skaliranje, tada će k_1 biti jednak k_2 što možemo nazvati k, pa operator poprima oblik:

$$\mathbf{\Psi}_{pskal} = \begin{bmatrix} k & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

što se još može zapisati i u obliku:

$$\mathbf{\Psi}_{pskal} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{k} \end{bmatrix}.$$



Slika 5.5: Skaliranje točke: djelovanje po komponentama

Inverzni operator operatoru skaliranja je skaliranje recipročnim koeficijentima, što za neproporcionalno skaliranje daje:

$$\mathbf{\Psi}_{skal}^{-1} = \begin{bmatrix} \frac{1}{k_1} & 0 & 0 \\ 0 & \frac{1}{k_2} & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

a za proporcionalno skaliranje:

$$\mathbf{\Psi}_{pskal}^{-1} = \begin{bmatrix} \frac{1}{k} & 0 & 0 \\ 0 & \frac{1}{k} & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

ili

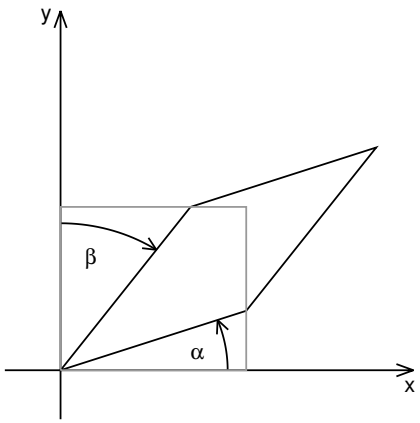
$$\mathbf{\Psi}_{pskal}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & k \end{bmatrix},$$

ovisno o matrici kojom je vršeno proporcionalno skaliranje.

5.2.5 Smik

Smik je "uzdužna" deformacija čije se djelovanje najbolje vidi sa slike 5.6; slika prikazuje rezultat djelovanja smika na kvadrat. Djelovanje smika sastoji se od deformacije uzduž osi x i deformacije uzduž osi y te se opisuje kutovima α i β.

Djelovanje operatora na pojedine osi prikazano je na slici 5.7.



Slika 5.6: Smik. Slika prikazuje djelovanje smika na zasivljeni kvadrat. Rezultat je prikazani paralelogram.

Za pojedine komponente proizlazi:

$$x' = x'_x + x'_y = x + y \cdot \mathrm{tg}\,(\beta)$$
$$y' = y'_y + y'_x = y + x \cdot \mathrm{tg}\,(\alpha)$$

pa se prelaskom na homogene koordinate dobiva:

$$\left[\begin{array}{ccc} T'_{h,x} & T'_{h,y} & T'_{h,h} \end{array}\right] = \left[\begin{array}{ccc} T_{h,x} & T_{h,y} & T_{h,h} \end{array}\right] \cdot \left[\begin{array}{ccc} 1 & \mathrm{tg}\,(\alpha) & 0 \\ \mathrm{tg}\,(\beta) & 1 & 0 \\ 0 & 0 & 1 \end{array}\right]$$

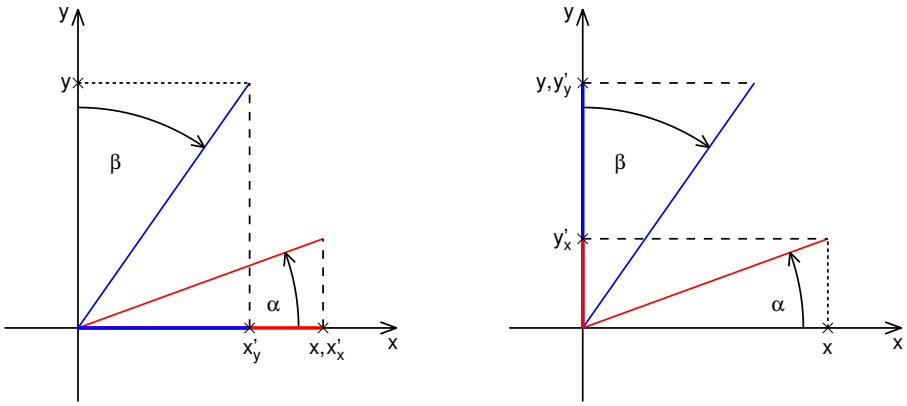
pa se operator smika zapisuje:

$$\mathbf{\Psi}_{smik} = \left[\begin{array}{ccc} 1 & \mathrm{tg}\,(\alpha) & 0 \\ \mathrm{tg}\,(\beta) & 1 & 0 \\ 0 & 0 & 1 \end{array}\right].$$

Potražimo li inverz ovog operatora iz uvjeta da umnožak operatora smika i inverznog operatora daje jediničnu matricu dobiva se:

$$\mathbf{\Psi}_{smik} = \left[\begin{array}{ccc} \frac{1}{1-\mathrm{tg}\,(\alpha)\cdot\mathrm{tg}\,(\beta)} & \frac{-\mathrm{tg}\,(\alpha)}{1-\mathrm{tg}\,(\alpha)\cdot\mathrm{tg}\,(\beta)} & 0 \\ \frac{-\mathrm{tg}\,(\beta)}{1-\mathrm{tg}\,(\alpha)\cdot\mathrm{tg}\,(\beta)} & \frac{1}{1-\mathrm{tg}\,(\alpha)\cdot\mathrm{tg}\,(\beta)} & 0 \\ 0 & 0 & 1 \end{array}\right].$$

što nije definirano ukoliko je umnožak tangensa jednak 1.



(a) Utjecaj smika na komponentu x (b) Utjecaj smika na komponentu y

Slika 5.7: Smik: djelovanje po komponentama

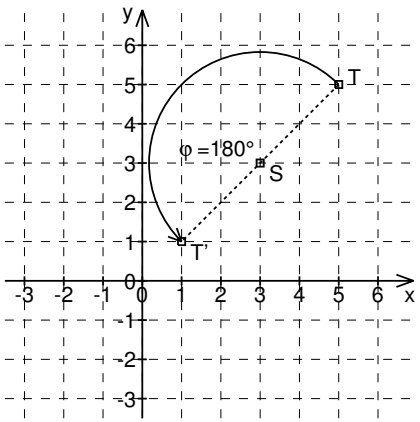
5.2.6 Primjer

U nastavku ćemo pokazati jednostavan primjer na kojem se mogu uočiti interesantni detalji. Potrebno je rotirati točku T(5,5) u smjeru suprotnom od smjera kazaljke na satu za kut ϕ oko točke S(3,3). Slika 5.8. pokazuje što želimo učiniti. Na slici je prikazan primjer rotacije za 180°.

Pokušamo li direktno primijeniti operator rotacije, rezultat neće dati očekivani rezultat (zapravo, ovisi što ste očekivali). Naime, operator rotacije izveden je tako da rotira točku oko ishodišta. Želimo li rotirati točku oko nekog drugog središta, morat ćemo primijeniti kompoziciju operatora translacije, rotacije i ponovno translacije. Evo koraka koje treba napraviti.

1. Točku T potrebno je translatirati istom translacijom koja bi točku S (željeno središte rotacije) dovela u ishodište koordinatnog sustava. To će biti translacija za $\Delta_x = -S_x$ i $\Delta_y = -S_y$, uz pretpostavku da je točka S dana komponentama $(S_x, S_y, 1)$. Nazovimo ovu tranformaciju $\mathbf{\Psi_1}$:

$$\mathbf{\Psi}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -S_x & -S_y & 1 \end{bmatrix}.$$



Slika 5.8: Rotacija oko zadane točke

2. Na ovako dobivenu točku možemo primijeniti operator rotacije jer se sada središte rotacije nalazi u ishodištu. Tada je $\mathbf{\Psi_2}$:

$$\mathbf{\Psi}_2 = \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

3. Konačno, nakon što smo točku zarotirali, moramo je natrag translatirati inverznom translacijom od one iz prvog koraka:

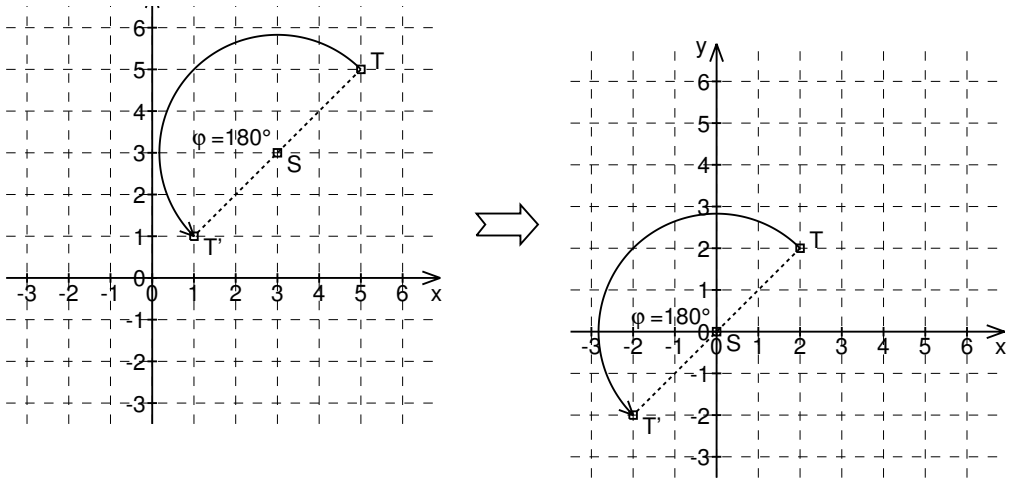
$$\mathbf{\Psi}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ S_x & S_y & 1 \end{bmatrix}.$$

Što se dogodilo nakon prvog koraka, opisuje slika 5.9.

U svakom koraku dobili smo po jedan operator. Zbirni operator $\mathbf{\Psi}$ koji će napraviti zadanu operaciju dobije se kao umnožak sva tri operatora i to upravo redoslijedom kojim su djelovali:

$$\mathbf{\Psi} = \mathbf{\Psi}_1 \cdot \mathbf{\Psi}_2 \cdot \mathbf{\Psi}_3$$

Da je ovo ispravno, možemo se lako uvjeriti pokusom. Ako rotiramo točku T za 180° oko točke S, trebali bismo dobiti točku $T'(1, 1)$. Ako izračunamo vrijednost operatora uz zadani kut dobit ćemo:



Slika 5.9: Rotacija oko zadane točke: nakon translacije

$$\mathbf{\Psi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & -3 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 3 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 6 & 6 & 1 \end{bmatrix}$$

Primijenimo li operator na točku $T(5,5)$ dobivamo:

$$\begin{aligned} \begin{bmatrix} T'_{h,x} & T'_{h,y} & T'_{h,h} \end{bmatrix} &= \begin{bmatrix} T_{h,x} & T_{h,y} & T_{h,h} \end{bmatrix} \cdot \mathbf{\Psi} \\ &= \begin{bmatrix} 5 & 5 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 6 & 6 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \end{aligned}$$

Dakle, dobili smo točku koju smo i očekivali.

Sličan postupak koji je naveden u prethodna tri koraka često se provodi jer izvedene elementarne transformacije djeluju obzirom na ishodište, pa treba dobro paziti što se želi postići, a što transformacije zapravo daju.

5.3 3D transformacije

5.3.1 Uvod

3D transformacije su transformacije nad točkom u 3D prostoru. U nastavku ćemo obraditi iste transformacije koje smo obradili u 2D prostoru. Jedina novost koju donosi 3D prostor su tri operatora rotacije umjesto jednog u 2D prostoru. Tako ćemo obraditi:

- translaciju,
- rotaciju oko osi x,
- rotaciju oko osi y,
- rotaciju oko osi z,
- skaliranje te
- smik.

Podsjetimo se još jednom: redoslijed djelovanja transformacija bitan je za krajnji rezultat, osim ako se ne radi o uzastopnom djelovanju iste transformacije kada je redoslijed nebitan. Naravno, rotacija oko osi x i rotacija oko osi y nisu iste transformacije.

U nastavku će biti dani izrazi za ove transformacije, budući da se oni izvode identično kao i izrazi za 2D transformacije koje smo već obradili u prethodnom potpoglavlju.

5.3.2 Translacija

Translacija pomiče točku tako da svakoj koordinati točke doda određeni pomak. Vrijedi:

$$T_x^{'} = T_x + \Delta_x$$
$$T_y^{'} = T_y + \Delta_y$$
$$T_z^{'} = T_z + \Delta_z$$

ili nakon prelaska u homogene koordinate:

$$\begin{bmatrix} T_{h,x}^{'} & T_{h,y}^{'} & T_{h,z}^{'} & T_{h,h}^{'} \end{bmatrix} = \begin{bmatrix} T_{h,x} & T_{h,y} & T_{h,z} & T_{h,h} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta_x & \Delta_y & \Delta_z & 1 \end{bmatrix}$$

što daje operator translacije u 3D:

$$\mathbf{\Psi}_{tr} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta_x & \Delta_y & \Delta_z & 1 \end{bmatrix}.$$

Inverz je translacija za negativne pomake:

$$\mathbf{\Psi}_{tr}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\Delta_x & -\Delta_y & -\Delta_z & 1 \end{bmatrix}.$$

5.3.3 Rotacija

Razlikujemo tri rotacije: rotaciju oko osi x, rotaciju oko osi y te rotaciju oko osi z.

Rotacija oko osi x

Rotacija oko osi x, gledano iz pozitivnog smjera osi u ishodište koordinatnog sustava, rotira točku u y-z ravnini pri čemu x-koordinata točke ostaje nepromijenjena.

Operator rotacije oko osi x u smjeru suprotnom od smjera kazaljke na satu (engl. *CCW*) glasi:

$$\mathbf{\Psi}_{rotx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) & 0 \\ 0 & -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Njemu inverzni operator je operator rotacije oko osi x u smjeru kazaljke na satu (engl. *CW*):

$$\mathbf{\Psi}'_{rotx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Rotacija oko osi y

Rotacija oko osi y, gledano iz pozitivnog smjera osi u ishodište koordinatnog sustava, rotira točku u x-z ravnini, pri čemu y-koordinata točke ostaje nepromijenjena.

Operator rotacije oko osi y u smjeru suprotnom od smjera kazaljke na satu glasi:

$$\mathbf{\Psi}_{roty} = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Njemu inverzni operator je operator rotacije oko osi y u smjeru kazaljke na satu:

$$\mathbf{\Psi}^{-1}_{roty} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Rotacija oko osi z

Rotacija oko osi z, gledano iz pozitivnog smjera osi u ishodište koordinatnog sustava, rotira točku u x-y ravnini, pri čemu z-koordinata točke ostaje nepromijenjena.

Operator rotacije oko osi z u smjeru suprotnom od smjera kazaljke na satu glasi:

$$\mathbf{\Psi}_{rotz} = \begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Njemu inverzni operator je operator rotacije oko osi z u smjeru kazaljke na satu:

$$\mathbf{\Psi}^{-1}_{rotz} = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Usporedimo li ovu matricu s matricom dobivenom za 2D rotaciju, vidimo da je matrica građena identično. Ovo je posljedica činjenice da smo rotaciju u 2D upravo izveli kao rotaciju u x-y ravnini gdje smo rekli da točku rotiramo oko ishodišta; ovo bi se slobodno moglo proširiti pa reći da rotaciju izvodimo oko osi okomite na x-y ravninu koja prolazi ishodištem $\Rightarrow$ z-osi!

U sljedećem poglavlju, prilikom izvođenja transformacije pogleda koja se temelji na uporabi translacije i nekoliko rotacija, rotaciju ćemo raditi u smjeru kazaljke na satu što znači u, matematički gledano, negativnom smjeru. Stoga ćemo tada kao operator rotacije u x-y ravnini koristiti operator $\mathbf{\Psi}_{rotz}$ uz negativan kut što odgovara operatoru $\mathbf{\Psi}_{rotz}^{-1}$ uz pozitivan iznos kuta rotacije.

5.3.4 Skaliranje

Skaliranje svaku koordinatu "skalira" (rasteže ili steže) množeći je sa odgovarajućim koeficijentom. Kako imamo tri koordinate radnog prostora, imati ćemo i tri koeficijenta. Ako su ti koeficijenti međusobno različiti, govorimo o neproporcionalnom skaliranju; inače govorimo o proporcionalnom skaliranju.

$$\mathbf{\Psi}_{sk} = \begin{bmatrix} k_1 & 0 & 0 & 0 \\ 0 & k_2 & 0 & 0 \\ 0 & 0 & k_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Inverz je skaliranje s recipročnim vrijednostima:

$$\mathbf{\Psi}_{sk}^{-1} = \begin{bmatrix} \frac{1}{k_1} & 0 & 0 & 0 \\ 0 & \frac{1}{k_2} & 0 & 0 \\ 0 & 0 & \frac{1}{k_3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Proporcionalno skaliranje dobije se za $k_1 = k_2 = k_3 = k$, no tada se operator proporcionalnog skaliranja može zapisati na još jedan način (osim već prikazanog općeg, pa uvrštavanjem k za sve koeficijente):

$$\mathbf{\Psi}_{psk} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{k} \end{bmatrix}.$$

U tom slučaju inverz ovom operatoru je operator:

$$\mathbf{\Psi}_{psk}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & k \end{bmatrix}.$$

5.3.5 Smik

Smik je uzdužna deformacija koja deformira položaj točke i opisuje se kutom deformacije prema svakoj koordinatnoj osi. U 3D prostoru imamo 3 koordinatne osi i tri kuta: α, β i γ.

Operator smika u 3D glasi:

$$\mathbf{\Psi}_{sm} = \begin{bmatrix} 1 & \mathrm{tg}\,(\alpha) & \mathrm{tg}\,(\alpha) & 0 \\ \mathrm{tg}\,(\beta) & 1 & \mathrm{tg}\,(\beta) & 0 \\ \mathrm{tg}\,(\gamma) & \mathrm{tg}\,(\gamma) & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Izvođenje inverznog operatora i pronalaženje uvjeta njegove egzistencije ostavlja se čitateljima za zabavu.

5.3.6 Primjer

U nastavku knjige trebat će nam rotacija oko zadanog središta, pa ćemo se ovdje još jednom podsjetiti na primjer koji smo već pokazali kod 2D transformacija. Potrebno je točku T rotirati oko točke S. Naravno, kako u 3D prostoru imamo 3 vrste rotacija, potrebno je zadati i koju rotaciju želimo primijeniti. Jednom kada imamo sve podatke, postupak je sljedeći:

1. translatirati točku S u ishodište koordinatnog sustava,
2. izvršiti rotaciju te
3. primijeniti inverznu translaciju od translacije iz prvog koraka.

5.4 Veza koordinatnih sustava

Određivanje vrijednosti koordinata točaka koje su zadane u jednom koordinatnom sustavu gledano iz drugog koordinatnog sustava iznimno je važno. Uzmimo za primjer učionicu čiji je globalni koordinatni sustav u jednom uglu te učionice. Neka se u učionici nalazi niz stolova i stolaca. Stolac kao pojedini objekt zadaje se u svom lokalnom koordinatnom sustavu a zatim se stvaraju primjerci tih stolaca na potrebnim mjestima u učionici. Tako je i sa stolovima i drugim sličnim objektima. Za stolac tada kažemo da se nalazi u svom lokalnom koordinatnom sustavu te je potrebno odrediti njegove koordinate u globalnom koordinatnom sustavu.

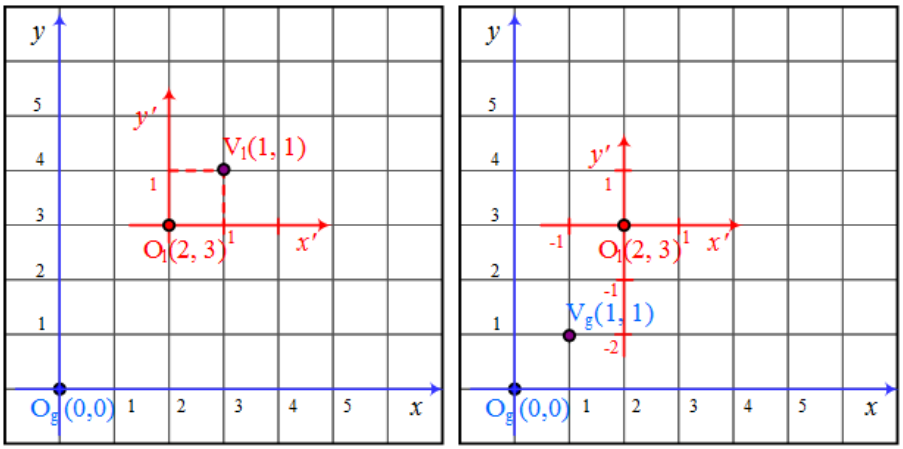
Ako uzmemo za primjer Sunčev sustav možemo odabrati koordinatni sustav Sunca kao globalni sustav. U njemu se nalazi Zemlja sa svojim lokalnim koordinatnim sustavom u kojem je lokalni sustav Mjeseca. No gledano sa Zemlje nas će zanimati i koordinate Sunca i koordinate Mjeseca promatrano iz našeg koordinatnog sustava.

Sličan primjer dobit ćemo i promatranjem čovjeka. Koordinatni sustav ramena nalazi se u globalnom koordinatnom sustavu čovjeka. U sustavu ramena nalazi se sustav lakta a u sustavu lakta je sustav šake i tako redom dalje po

pojedinim zglobovima prsta. Prilikom pokreta ruke bit će potrebno odrediti koordinate prsta u sustavu šake, zatim u sustavu lakta, ramena pa sve do globalnog koordinatnog sustava čovjeka. Pored ovih primjera vrlo će nam važno biti određivanje svih koordinata u sceni promatrano iz koordinatnog sustava kamere tako da je savladavanje povezivanja dva sustava jedan od bazičnih koncepata.

5.4.1 Translatirani koordinatni sustavi

Kako je ovo važna tematika koja može biti dosta zbunjujuća ako odmah krenemo u 3D prostor, počet ćemo s jednostavnim primjerima u 2D prostoru i razviti metodu određivanja koordinata. Uzmimo globalni koordinatni sustav (O_g, x, y) gdje je O_g ishodište a x i y koordinatne osi, koji je smješten kao na slici 5.10 (lijevo) i označen plavom bojom. U ovom sustavu se nalazi lokalni koordinatni sustav (O_l, x', y') na slici označen crvenom bojom koji je translatiran za (2, 3) te je njegovo ishodište O_l. U lokalnom koordinatnom sustavu nalazi se jedna točka $V_l = (1, 1)$. Potrebno je odrediti koordinate točke V_l ali promatrano iz globalnog koordinatnog sustava.



Slika 5.10: Translatirani koordinatni sustavi. (lijevo) Koordinate točke zadane u lokalnom koordinatnom sustavu promatrano iz globalnog. (desno) Koordinate točke zadane u globalnom koordinatnom sustavu promatrano iz lokalnog.

Ovo nam ne izgleda toliko složeno pa ćemo na ovom primjeru početi s razvojem naše metode. Prvo ćemo odrediti transformaciju između lokalnog i globalnog koordinatnog sustava. Zamislimo da se lokalni koordinatni sustav podudara s globalnim koordinatnim sustavom. Očito je potrebna translacija za (2, 3) kako bi lokalni sustav pomaknuli na traženu poziciju u globalnom koordinatnom sustavu.

Znači translacija za T će i bilo koju točku koja se nalazi u lokalnom sustavu pomaknuti u globalnom sustavu na traženu poziciju pa tako i našu promatranu točku $V_l = (1, 1)$.

$$V_g = V_l T = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 4 & 1 \end{bmatrix}$$

Znači, kada znamo transformaciju koja je potrebna da lokalni koordinatni sustav smjestimo u globalnom koordinatnom sustavu znamo i transformirati bilo koju točku toga lokalnog koordinatnog sustava. U promatranom primjeru našu točku $V_l = (1, 1)$ transformiramo matricom T i dobijemo točku $V_g = (3, 4)$ što je i vidljivo na slici 5.10. Odnosno, dobili smo koordinate točke V_l u globalnom koordinatnom sustavu.

Uzmimo sada obrnutu situaciju. Neka je poznata točka u globalnom koordinatnom sustavu $V_g = (1, 1)$ a potrebno je odrediti njene koordinate promatrano iz lokalnog koordinatnog sustava (O_l, x', y') kao na slici 5.10 (desno). Sada je potrebno zamisliti da lokalni koordinatni sustav označen crveno zajedno s promatranom točkom translatiramo u ishodište. Za to je potrebna inverzna transformacija, odnosno translacija za $T^{-1} = (-2, -3)$:

$$V_l = V_g T^{-1} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -3 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & 1 \end{bmatrix}$$

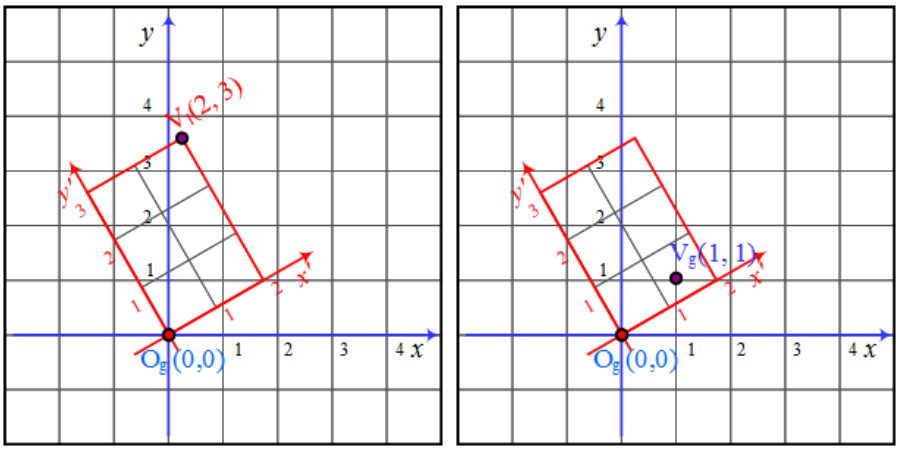
Doista, na slici 5.10 (desno) možemo vidjeti da se promatrana točka u crvenom koordinatnom sustavu nalazi na koordinatama $(-1, -2)$ lokalnog koordinatnog sustava. Na ovaj način ćemo određivati koordinate točke zadane u globalnom koordinatnom sustavu promatrano iz lokalnog koordinatnog sustava.

5.4.2 Rotirani koordinatni sustavi

Sada ćemo promotriti isti scenarij ali za rotirani koordinatni sustav. Prvo ćemo odrediti koordinate točke $V_l = (2, 3)$ zadane u lokalnom koordinatnom sustavu (O_l, x', y') koji je rotiran za 30ř kao na slici 5.11 (lijevo). Potrebna transformacija je rotacija za 30ř s obzirom na inicijalni položaj podudaran s globalnim koordinatnim sustavom:

$$V_g = V_l R = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \cos(30\text{ř}) & \sin(30\text{ř}) & 0 \\ -\sin(30\text{ř}) & \cos(30\text{ř}) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.23 & 3.60 & 1 \end{bmatrix}$$

Dobiveni rezultat odnosno koordinate $V_g = (0.23,\ 3.60)$ možemo otprilike procijeniti i na slici 5.11. Znači, to su koordinate točke zadane u lokalnom koordinatnom sustavu $V_l = (2,\ 3)$ ali promatrano iz globalnog koordinatnog sustava. Možemo primijetiti da retci u rotacijskoj matrici odgovaraju vektorima baze lokalnog koordinatnog sustava prikazanih u globalnom koordinatnom sustavu, odnosno prvi redak sadrži $x' = (\cos(30\check{r})\ \sin(30\check{r}))$, a drugi redak sadrži $y' = (-\sin(30\check{r})\ \cos(30\check{r}))$.



Slika 5.11: Rotirani koordinatni sustavi. (lijevo) Koordinate točke zadane u lokalnom koordinatnom sustavu promatrano iz globalnog. (desno) Koordinate točke zadane u globalnom koordinatnom sustavu promatrano iz lokalnog.

Za obrnuti slučaj kada promatramo točku u globalnom koordinatnom sustavu ali iz lokalnog koordinatnog sustava potrebna nam je inverzna transformacija rotacije slika 5.11 (desno). Sada je transformacija za promatranu točku $V_g = (1,\ 1)$ sljedeća :

$$V_l = V_g R^{-1} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \cos(30\check{r}) & -\sin(30\check{r}) & 0 \\ \sin(30\check{r}) & \cos(30\check{r}) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1.37 & 0.37 & 1 \end{bmatrix}$$

I dobili smo koordinate promatrane točke $V_g = (1,\ 1)$ u lokalnom koordinatnom sustavu $V_l = (1.37,\ 0.37)$ što možemo vidjeti i na slici 5.11 (desno). Možemo primijetiti da stupci u rotacijskoj matrici odgovaraju vektorima baze lokalnog koordinatnog sustava, odnosno $x' = (\cos(30\check{r})\ \sin(30\check{r}))$, $y' = (-\sin(30\check{r})\ \cos(30\check{r}))$.

5.4.3 Složenije transformacije koordinatnih sustava

U općem slučaju kod transformacija čvrstog tijela naš objekt možemo rotirati i translatirati u prostoru. Kada trebamo odrediti koordinate zadane u lokalnom

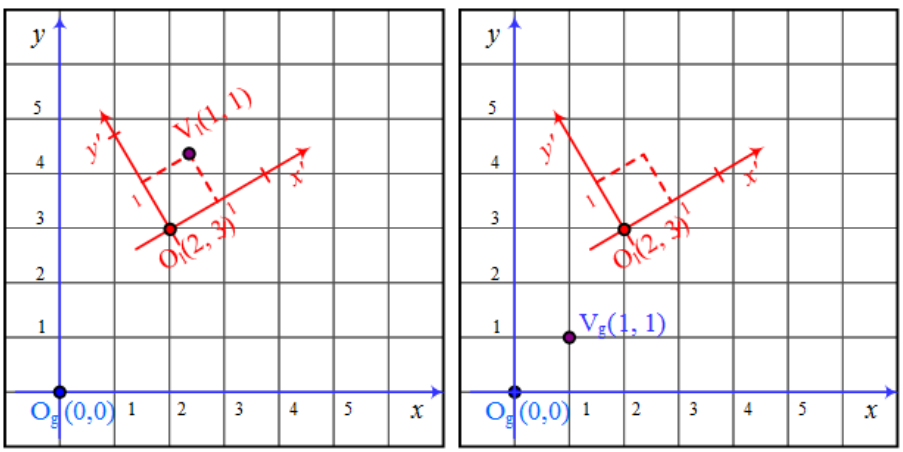
sustavu promatrano iz globalnog koordinatnog sustava važan je redoslijed transformacija kojima se lokalni koordinatni sustav transformirao. Translaciju uvijek možemo odrediti iz poznavanja ishodišta lokalnog koordinatnog sustava u globalnom a rotaciju iz zakrenutosti osi x' prema x osi. Na taj način složeniji slučaj rastavljamo na jednostavnije koji su prethodno opisani.

Slika 5.12 (lijevo) prikazuje takav složeniji slučaj. Da bi lokalni koordinatni sustav doveli na prikazanu poziciju iz osnovne pozicije kada je podudaran s globalnim koordinatnim sustavom prvo je potrebno načiniti rotaciju za 30ř oko ishodišta a zatim translaciju za $(2, 3)$. Obrnuti redoslijed transformacija ne bi dao ispravan rezultat. To znači da su i za bilo koju drugu točku potrebne transformacije rotacija pa zatim translacija:

$$V_g = (V_l R)\, T = V_l\, (RT) =$$

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \cos(30\text{ř}) & \sin(30\text{ř}) & 0 \\ -\sin(30\text{ř}) & \cos(30\text{ř}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 2.37 & 4.37 & 1 \end{bmatrix}$$

Slika 5.12 (desno) prikazuje određivanje koordinate zadane u globalnom koordinatnom sustavu promatrano iz lokalnog sustava. U ovom slučaju potrebno je promatrati inverznu ukupnu transformaciju $(RT)^{-1}$koja postaje $T^{-1}R^{-1}$:



Slika 5.12: Translacija i rotacija koordinatnog sustava. (lijevo) Koordinate točke zadane u lokalnom koordinatnom sustavu promatrano iz globalnog. (desno) Koordinate točke zadane u globalnom koordinatnom sustavu promatrano iz lokalnog.

$$V_l = V_g\,(RT)^{-1} = V_g T^{-1} R^{-1} =$$

$$= \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -3 & 1 \end{bmatrix} \begin{bmatrix} \cos(30\check{r}) & -\sin(30\check{r}) & 0 \\ \sin(30\check{r}) & \cos(30\check{r}) & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} -1.87 & -1.23 & 1 \end{bmatrix}$$

Postupak možemo na slici 5.12 (desno) zamisliti kao da želimo vratiti lokalni koordinatni sustav u inicijalni položaj odnosno podudariti s globalnim. Prvo ćemo ga translatirati u ishodište a zatim rotirati za kut od $(-30\check{r})$ i promotriti gdje će točke V_g završiti. Postupak je upravo obrnut nego kod određivanja točke zadane u lokalnom koordinatnom sustavu promatrano iz globalnog.

Sve navedene postupke možemo ostvariti i promatranjem vektora. Uzmimo posljednji slučaj gdje određujemo koordinate koje će točka zadana u globalnom koordinatnom sustavu imati u lokalnom koordinatnom sustavu. Radi se o vektoru $\overrightarrow{O_l V_g} = \vec{V}_g - \vec{O}_l = (1-2,\, 1-3)$. Taj je vektor potrebno projicirati na bazne vektore lokalnog koordinatnog sustava čime se dobivaju koordinate zadane točke u lokalnom koordinatnom sustavu:

$$V_l = (\overrightarrow{O_l V_g}\, proj\, \overrightarrow{x'}, \overrightarrow{O_l V_g}\, proj\, \overrightarrow{y'})$$

U općem slučaju kada umjesto promatrane rotacije imamo afine transformacije kao što su skaliranje, neuniformno skaliranje i smik možemo ukupnu transformaciju podijeliti na translaciju i „sve ostalo“. Potrebnu translacijsku matricu odredit ćemo iz poznavanja ishodišta lokalnog sustava u globalnom a poznavanje koordinatnih osi x' i y' upotrijebit ćemo za određivanje dijela matrice koja obavlja „sve ostalo“.

Možemo imati i problem s tri koordinatna sustava gdje je jedan globalni u kojem se nalaze dva lokalna koordinatna sustava. Iz prvog lokalnog koordinatnog sustava tada treba odrediti koordinate zadane u drugom lokalnom koordinatnom sustavu. Takav problem se rastavlja na dva prethodno opisana slučaja koja se onda kombiniraju. Znači, prvo se koordinate iz prvog lokalnog koordinatnog sustava odrede u globalnom sustavu a zatim se odrede njihove koordinate promatrano iz drugog lokalnog koordinatnog sustava. Također, sve što je opisano u 2D prostoru primjenjivo je i u 3D prostoru.

5.5 *OpenGL* i transformacije

Prisjetimo se zaključka s početka ovog poglavlja: djelovanje operatora na točku možemo iskazati ili množenjem točke matricom operatora, ili množenjem matrice

operatora točkom. Kroz ovu knjigu mi ćemo se držati prvog načina, dok *OpenGL* koristi drugi način. Prisjetimo se još i veze između matrica istog operatora uz te dvije konvencije. Neka uz prvu konvenciju operatoru odgovara matrica $\mathbf{\Psi}$, a uz drugu konvenciju matrica $\mathbf{\Omega}$. Vrijedi:

$$\mathbf{\Omega}^{\mathrm{T}} = \mathbf{\Psi}.$$

Pogledajmo sada koje nam naredbe stoje na raspolaganju u *OpenGL*-u, i kako izgledaju pripadne matrice.

5.5.1 Translacija

Za potrebe translacije *OpenGL* nam nudi naredbu: `glTranslate*(dx,dy,dz);` koja trenutnu matricu množi matricom $\mathbf{\Omega}_{tr}$:

$$\mathbf{\Omega}_{tr} = \begin{bmatrix} 1 & 0 & 0 & \Delta_x \\ 0 & 1 & 0 & \Delta_y \\ 0 & 0 & 1 & \Delta_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Uvjerite se da je to upravo transponirana matrica od matrice $\mathbf{\Psi}_{tr}$ koju smo prethodno izveli. Inverz ove matrice je:

$$\mathbf{\Omega}_{tr}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -\Delta_x \\ 0 & 1 & 0 & -\Delta_y \\ 0 & 0 & 1 & -\Delta_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Zvjezdica u imenu zapravo nam govori da postoji porodica takvih funkcija koje primaju argumente različitog tipa. Primjerice, ako kao vrijednosti dx, dy i dz predajemo **float**-ove, pozvat ćemo naredbu `glTranslatef(1.0f,2.4f,-0.7f);`.

5.5.2 Skaliranje

Za potrebe skaliranja *OpenGL* nam nudi naredbu `glScale*(sx,sy,sz);` koja trenutnu matricu množi matricom $\mathbf{\Omega}_{tr}$:

$$\mathbf{\Omega}_{sk} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Uvjerite se da je to upravo transponirana matrica od matrice $\mathbf{\Psi}_{sk}$ koju smo prethodno izveli. Inverz ove matrice je:

$$\mathbf{\Omega}_{sk}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

5.5.3 Rotacija

Za potrebe rotacije *OpenGL* nam nudi naredbu: glRotate∗(a,x,y,z);. Ova naredba trenutnu matricu množi matricom koja obavlja rotaciju za kut α (prvi argument naredbe) oko proizvoljne osi određene vektorom (x, y, z). Pri tome se i dalje rotacija obavlja oko ishodišta. Kada vektor (x, y, z) predstavlja koordinatne osi, matrice će odgovarati već prethodno izvedenima.

Tako naredba glRotate∗(a ,1,0,0); računa matricu rotacije oko osi x, pa će trenutnu matricu pomnožiti matricom $\mathbf{\Omega}_{rotx}$:

$$\mathbf{\Omega}_{rotx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

.

Naredba glRotate∗(a ,0,1,0); računa matricu rotacije oko osi y, pa će trenutnu matricu pomnožiti matricom $\mathbf{\Omega}_{roty}$:

$$\mathbf{\Omega}_{roty} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

.

Naredba glRotate∗(a ,0,0,1); računa matricu rotacije oko osi z, pa će trenutnu matricu pomnožiti matricom $\mathbf{\Omega}_{rotz}$:

$$\mathbf{\Omega}_{rotz} = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

.

Konačno, za proizvoljnu os određenu vektorom (x, y, z), naredba glRotate∗(a,x,y,z); računa matricu rotacije oko osi određene tim vektorom, pa će trenutnu matricu pomnožiti matricom $\mathbf{\Omega}_{rot}$:

$$\begin{bmatrix} u_x^2+(1-u_x^2)\cos(\alpha) & u_xu_y(1-\cos(\alpha))-u_z\sin(\alpha) & u_xu_z(1-\cos(\alpha))+u_y\sin(\alpha) & 0 \\ u_xu_y(1-\cos(\alpha))+u_z\sin(\alpha) & u_y^2+(1-u_y^2)\cos(\alpha) & u_yu_z(1-\cos(\alpha))-u_x\sin(\alpha) & 0 \\ u_xu_z(1-\cos(\alpha))-u_y\sin(\alpha) & u_yu_z(1-\cos(\alpha))+u_x\sin(\alpha) & u_z^2+(1-u_z^2)\cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Pri tome su u_x, u_y i u_z komponente vektora $\vec{u}$ koji je dobiven normiranjem zadanog vektora $\vec{v} = (x, y, z)$, tj:

$$\vec{u} = \frac{\vec{v}}{||\vec{v}||}.$$

5.6 Transformacije normala

Prilikom rada s objektima u prostoru scene često ćemo trebati normale (primjerice, normale poligona). Štoviše, kako bismo uštedjeli na vremenu, normale ćemo računati prilikom učitavanja objekta i kasnije ih samo koristiti. Da bismo to međutim mogli, pogledajmo kakav utjecaj transformacije imaju na normale. Pogledajmo to na jednostavnom primjeru u 2D prostoru. Neka je zadan segment pravca određen točkama $T_1 = (0, 0)$ i $T_2 = (5, 1)$. Vektor pravca na kojem leži taj segment je $\vec{v} = (5, 1) - (0, 0) = (5, 1)$. Normala $\vec{n}$ koja pripada tom segmentu tada je vektor $\vec{n} = (-1, 5)$ (uvjerite se da je $\vec{v} \cdot \vec{n} = 0$). Normala je dobivena iz zahtjeva da skalarni produkt bude nula ($v_x \cdot n_x + v_y \cdot n_y = 0$).

Neka je sada zadana afina transformacija koja obavlja skaliranje po osi y s faktorom 2. Pripadna matrica $\mathbf{M}$ je:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ova transformacija točke T_1 i T_2 preslikava u $T_3 = T_1\mathbf{M} = (0, 0)$ te $T_4 = T_2\mathbf{M} = (5, 2)$, pri čemu se segment određen točkama T_1 i T_2 preslikava u segment određen točkama T_3 i T_4. Odgovarajući vektor pravca sada je $\vec{v}' = (5, 2) - (0, 0) = (5, 2)$.

Provjerimo je li vektor $\vec{n}$ i dalje korektna normala? Ako je, mora vrijediti da je skalarni produkt jednak nuli, no to više nije slučaj:

$$v'_x \cdot n_x + v'_y \cdot n_y = 5 \cdot (-1) + 2 \cdot 5 = -5 + 10 = 5 \neq 0.$$

Pokušamo li normalu transformirati na isti način kao i segment, opet nećemo dobiti korektnu normalu. Evo i dokaza. Izračunajmo najprije transformiranu normalu matricom $\mathbf{M}$ (u homogenom prostoru).

$$\vec{n}' = \vec{n} \cdot \mathbf{M} = \begin{bmatrix} -1 & 5 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 10 & 1 \end{bmatrix}$$

Skalarni produkt "nove" normale i vektora pravca daje:

$$v'_x \cdot n'_x + v'_y \cdot n'_y = 5 \cdot (-1) + 2 \cdot 10 = -5 + 20 = 15 \neq 0.$$

Kako onda doći do potrebne transformacije? Vratimo se na početak. Imali smo vektor $\vec{v}$ i normalu $\vec{n}$. Ta dva vektora jesu bili okomiti, i zadovoljavali su izraz:

$$\vec{v} \cdot \vec{n} = 0.$$

Uvažavajući da vektore prikazujemo kao jednoretčane matrice, prethodni se izraz za skalarni produkt može zamijeniti matričnim množenjem, pri čemu drugi član treba transponirati:

$$\mathbf{v}\ \mathbf{n}^T = 0.$$

Transformacijom vektora $\vec{v}$ dobiva se vektor $\vec{v}\ \mathbf{M}$ čime u prethodni izraz između v i n^T dolazi matrica $\mathbf{M}$ zbog čega jednakost prestaje vrijediti. Da bismo je vratili, matricu $\mathbf{M}$ treba neutralizirati – množeći je njezinim inverzom. Evo ideje:

$$\mathbf{vMM}^{-1}\mathbf{n}^T = 0.$$

Grupiranjem slijedi:

$$(\mathbf{vM})(\mathbf{M}^{-1}\mathbf{n}^T) = \mathbf{v}'\ \mathbf{n}'^T = 0.$$

Sada je očito da je:

$$\mathbf{n}'^T = \mathbf{M}^{-1}\mathbf{n}^T \quad \Rightarrow \quad \mathbf{n}' = (\mathbf{M}^{-1}\mathbf{n}^T)^T = \mathbf{nM}^{-1^T}$$

Iz ovog razmatranja došli smo do konačnog zaključka: normale se transformiraju množenjem s transponiranim inverzom originalne transformacije. Provjerimo to još i na radnom primjeru.

$$\mathbf{M}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{M}^{-1^T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$n' = n\mathbf{M}^{-1^T} = \begin{bmatrix} -1 & \frac{5}{2} & 1 \end{bmatrix}$$

$$v'_x \cdot n'_x + v'_y \cdot n'_y = 5 \cdot (-1) + 2 \cdot \frac{5}{2} = -5 + 5 = 0.$$

5.7 Ponavljanje

1. Geometrijske transformacije opisujemo matrično, i pri tome možemo koristiti dvije konvencije: točka T je jednoretčani vektor i množimo ga transformacijskom matricom M s desne strane $(T \cdot M)$ ili pak točka T je jednostupčani vektor i množimo ga transformacijskom matricom M s lijeve strane $(M \cdot T)$. Uz ove konvencije:
   (a) u kakvom su međusobnom odnosu matrice koje odgovaraju istoj transformaciji?
   (b) ako točku transformiramo nizom različitih transformacija, kako glasi jedna (konačna) matrica čijim se množenjem u jednom koraku odrađuju sve zadane transformacije?
2. Kako su definirane linearne transformacije? Jesu li translacija, rotacija i skaliranje linearne transformacije?
3. Kako su definirane afine transformacije? Jesu li translacija, rotacija i skaliranje afine transformacije?
4. Kako su definirane euklidske transformacije? Jesu li translacija, rotacija i skaliranje euklidske transformacije?
5. Napišite matrice za 2D-transformacije: translaciju, rotaciju, skaliranje, smik. Napišite matrice koje obavljaju inverzne transformacije.
6. Napišite matrice za 3D-transformacije: translaciju, rotaciju, skaliranje, smik. Napišite matrice koje obavljaju inverzne transformacije.
7. Ako imamo zadane koordinate točke u lokalnom koordinatnom sustavu, kako određujemo koordinate iste točke u globalnom koordinatnom sustavu?
8. Ako imamo zadane koordinate točke u globalnom koordinatnom sustavu, kako određujemo koordinate iste točke u lokalnom koordinatnom sustavu?
9. Kako se transformiraju normale kada se nad objektom obavlja zadana transformacija?

Poglavlje 6

Projekcije i transformacije pogleda

6.1 Projekcije

Područje računalne grafike osim rada u 2D prostoru obuhvaća i rad u 3D prostoru. Štoviše, ljudima je koncepcija 3D prostora daleko bliža nego 2D prostor. Razloga tome je mnogo, a jedan je i taj što živimo u 3D prostoru pa su pojmovi poput iznad, ispod, lijevo, desno, ispred ili iza sasvim prirodni jasni. No u 2D prostoru ograničeni smo na samo dva smjera. U 2D prostoru ne možemo prikazati tijela; moguć je isključivo prikaz likova. S druge strane, prikazne jedinice danas su još uvijek dominantno dvodimenzionalne. Zaslon monitora nudi nam mogućnost prikazivanja u jednoj ravnini. S druge strane, čovjek nastoji i u svijet računala uvesti 3D prostor. Dakako, to je jednim dijelom moguće. Sjetimo se samo fotoaparata. Kada slikamo, slikamo objekte u 3D prostoru. Kao rezultat slikanja dobivamo sliku, dakle komad papira na kojemu je "slika" smještena u ravninu – u 2D prostor. Ipak, pogledom na sliku dobivamo jasnu predodžbu o onome što je slikano; imamo privid 3D prostora. Naravno, u tom 3D prostoru ne možemo pogledati kako bi objekti izgledali kada bismo ih pogledali malo desnije, ili pak kada bismo otišli iza njih. Dobivena slika jednostavno je zamrznut prikaz onoga što smo vidjeli točno s mjesta s kojeg smo gledali i točno u smjeru u kojem smo gledali. I to je mjesto na koje uskače računalna grafika. Tu se dakle pruža mogućnost da i na računalu kreiramo takve "snimke" koje nam pokazuju što bismo vidjeli od 3D prostora kada bismo stajali u jednoj točki u prostoru i gledali prema nekoj drugoj točki.

Metode koje opisuju na koji način "gledamo" i što bismo zapravo vidjeli zovu se – **projekcije**. Naziv "projekcije" dolazi od činjenice da objekte 3D prostora

"projiciramo" na ravninu u 2D prostor. Projekcije su preslikavanja koja nam govore na koji način treba točke 3D prostora preslikati u ravninu u točke 2D prostora. Postoji više vrsta projekcija, a mi ćemo u nastavku obraditi dvije:

- paralelna projekcija te
- perspektivna projekcija.

Evo jednostavnog primjera što znači projicirati objekte 3D prostora u 2D prostor. Uzmimo list papira i stavimo ga na stol. Iznad njega (ali ne na njega) postavimo nekakav objekt, primjerice olovku, a iznad postavimo uključenu svjetiljku. Rezultat je sjena olovke na papiru; 3D objekt preslikao se je u ravninu.

Ovaj jednostavan pokus pokazao nam je još nešto – projekcije su destruktivne. U 3D prostoru znamo i duljinu olovke, i debljinu olovke, promjer, udaljenost do papira, kut pod kojim je olovka nagnuta s obzirom na ravninu papira i sl. Projiciranjem u 2D prostor dobili smo sliku iz koje više ne možemo doznati te informacije.

6.1.1 Paralelna projekcija

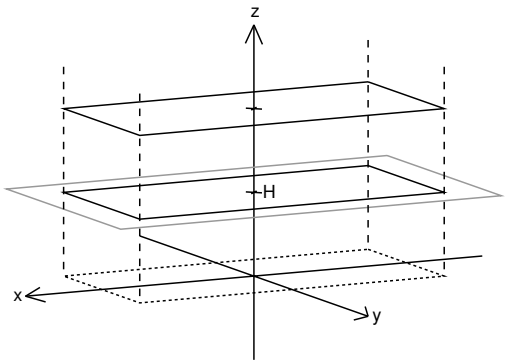
Jedan od najjednostavnijih modela projekcija jest model paralelne projekcije. Paralelne projekcije su one kod kojih su projektori (pravci uzduž kojih radimo projekciju) paralelni. Postoje dvije vrste takvih projekcija: *ortografske* i *kose*. Kod ortografskih projekcija projektori su okomiti na ravninu projekcije i mi ćemo u nastavku razmatrati upravo tu vrstu paralelne projekcije. Kod kosih projekcija projektori nisu okomiti na ravninu projekcije – tu vrstu projekcije dalje nećemo razmatrati.

Model ortografske projekcije podrazumijeva da je izvor svjetlosti smješten u beskonačnosti, pa su sve zrake svijetlosti koje dolaze do objekata okomite na ravninu projiciranja i samo takve zrake tvore sliku. Ovakvu projekciju daje točkasti izvor smješten iznad ravnine projiciranja i to na beskonačnoj udaljenosti. U tom slučaju projiciranjem objekta širokog 10 cm dobili bismo i sliku široku točno 10 cm.

Za opisivanje projekcije poslužiti ćemo se sljedećim primjerom. Želimo dobiti paralelnu projekciju točaka na ravninu $z = H$ pri čemu je H je proizvoljan broj (to je dakle projekcija na ravninu paralelnu s xy-ravninom koja je na visini H). Slika 6.1 prikazuje problem.

Na slici je prikazan lik koji se projicira. Ispod lika smještena je ravnina projekcije s likom koji se dobije projiciranjem. Na slici su također prikazani i projektori kroz sva četiri vrha lika. Oni su okomiti na ravninu projekcije.

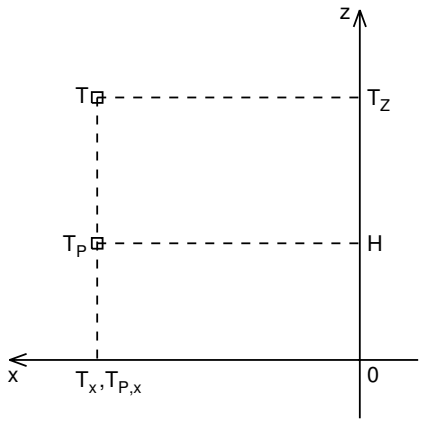
Ovaj model uzet je zbog jednostavnosti, kako bismo se lakše upoznali s idejom paralelne projekcije. Rješenje zadanog problema je jednostavno. Ako proizvoljnu točku T projiciramo, dobiti ćemo točku T_P i pri tome će za komponente točke T_P vrijediti:



Slika 6.1: Paralelna projekcija

$$T_{P,x} = T_x$$
$$T_{P,y} = T_y$$
$$T_{P,z} = H$$

To se jasno vidi sa slike 6.2. Na slici je prikazana situacija za x-komponentu, no isto vrijedi i za y-komponentu. Projicira se točka T, a projekcija je točka T_P.



Slika 6.2: Paralelna projekcija, analiza x-koordinate

Sve točke koje ćemo na ovaj način projicirati, imati će z-koordinatu jednaku H; dakle, sve će ležati u ravnini $z = H$. Ovdje prikazano paralelno projiciranje može se opisati matricom paralelne projekcije:

$$\pi_{par} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & H & 1 \end{bmatrix}$$

Tada se projekcija opisuje umnoškom točke i matrice projekcije:

$$\begin{aligned} T_{Ph} &= [\ T_{Ph,x} \quad T_{Ph,y} \quad T_{Ph,z} \quad T_{Ph,h}\ ] \\ &= T_h \cdot \pi_{par} \\ &= [\ T_x \quad T_y \quad T_z \quad 1\ ] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & H & 1 \end{bmatrix} \\ &= [\ T_x \quad T_y \quad H \quad 1\ ] \end{aligned}$$

Kada dobivene točke prikazujemo u ravnini, tada kao x- i y-koordinatu koristimo prve dvije komponente točke, dok ostale komponente zanemarujemo.

Prilikom crtanja točaka u ravnini pri tome treba voditi računa o još jednom problemu: ako se dvije točke 3D prostora preslikavaju u istu točku ravnine (jedna zelena a druga plava), koju ćemo točku tada prikazati u ravnini? Da bismo odgovorili na ovo pitanje, treba se prisjetiti koja je zapravo svrha projekcije – želimo "vidjeti" 3D prostor na zaslonu. Ako je to istina, tada se dvije različite točke 3D prostora koje se preslikavaju u istu točku 2D prostora mogu tumačiti kao dva objekta smještena jedan iza drugoga. Koji ćemo od tih objekata vidjeti, ovisi o tome gdje se nalazi promatrač. Pretpostavimo stoga da se u 3D prostoru u nekoj točki nalazi promatrač, potom se na određenoj udaljenosti nalazi ravnina projekcije i konačno, svi se objekti scene nalaze dalje iza te ravnine. U tom slučaju, promatrač će vidjeti onaj objekt koji je bliži ravnini projekcije, i to je uobičajeni scenarij koji ćemo dalje koristiti. Želimo li sačuvati informaciju o udaljenosti, matricu projekcije potrebno je modificirati. Iskoristit ćemo z-koordinatu projicirane točke za pohranu udaljenosti projicirane točke od same ravnine. Tada vrijedi:

$$\begin{aligned} T_{P,x} &= T_x \\ T_{P,y} &= T_y \\ T_{P,z} &= T_z - H \end{aligned}$$

dok se matrica paralelne projekcije koja čuva udaljenost točke od ravnine modificira u:

$$\pi'_{par} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -H & 1 \end{bmatrix}.$$

Rezultat koji se dobije množenjem točke koju projiciramo i ove matrice treba tumačiti na sljedeći način.

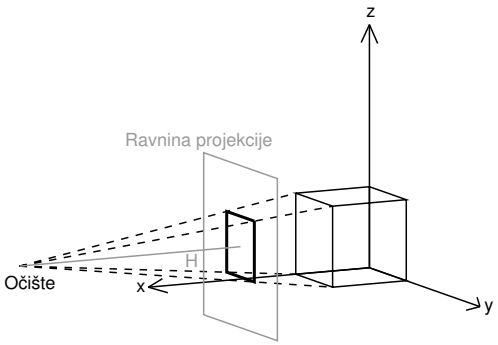
- Prve dvije komponente točke odgovaraju komponentama točke u ravnini projekcije.
- z-koordinata točke, budući da točka pripada ravnini $z = H$ iznosi upravo H.
- Treća komponenta točke odgovara udaljenosti točke koju smo projicirali od ravnine projekcije, i može poslužiti kao kriterij za određivanje koju točku treba prikazati u slučaju da se više točaka projicira u istu točku ravnine. Ovu informaciju možemo iskoristiti uz podatkovnu strukturu z-spremnika za određivanje konačne vidljive točke.

Matrica paralelne projekcije ispala je ovako jednostavna zbog toga što smo odabrali vrlo jednostavan primjer projekcije: projekciju na ravninu $z = H$. Razumno bi bilo pitati se a kako bi izgledala matrica paralelne projekcije na proizvoljnu ravninu u 3D prostoru. No odgovor na ovo glasi: takvu matricu (na svu sreću) ne moramo tražiti jer bi ispala krajnje komplicirana i nepregledna. Umjesto toga, primijenit ćemo postupak *transformacije pogleda* o kojem će biti više riječi u nastavku, i zatim iskoristiti upravo ovu jednostavnu matricu paralelne projekcije. Prije no što se upustimo u postupak transformacija pogleda, pogledajmo još i drugi tip projekcije.

6.1.2 Perspektivna projekcija

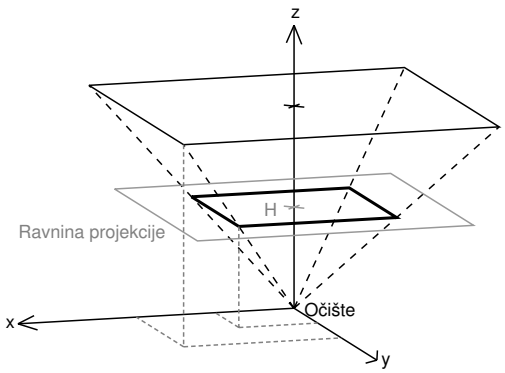
Perspektivna projekcija je model koji je čovjeku bliži. Naime, model uvodi dvije točke: očište i gledište. Očište je točka u kojoj se nalazi promatrač. Gledište određuje smjer prema kojem promatrač gleda. U točki gledišta stvara se ravnina projekcije koja je okomita na spojnicu očište-gledište. Točka gledišta pripada toj ravnini i ona tipično postaje ishodište lokalnog dvodimenzionalnog koordinatnog sustava koji razapinjemo u ravnini projekcije. Uočimo da dvodimenzionalni koordinatni sustav koji ovako dobijemo nije jednoznačan: smjer osi x može biti proizvoljno rotiran.

Projekcija proizvoljne točke T dobije se tako da se očište spoji pravcem sa zadanom točkom T. Kao projekcija točke uzima se točka u kojoj tako dobiveni pravac probada ravninu projekcije. Općeniti primjer perspektivne projekcije prikazan je na slici 6.3.



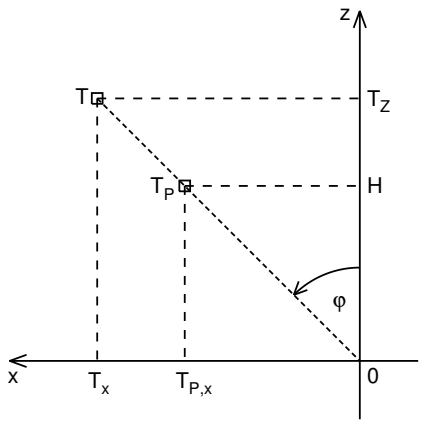
Slika 6.3: Perspektivna projekcija

Za matematičku analizu problema pokušajmo odrediti perspektivnu projekciju proizvoljne točke T, za slučaj da je očište smješteno u ishodište koordinatnog sustava, a gledište na z-os na visini $z = H$. Ovakvim odabirom očišta i gledišta postiže se da je ravnina projekcije upravo ravnina $z = H$ (dakle, ravnina paralelna s xy-ravninom podignuta od ishodišta za H). Slika 6.4 pokazuje perspektivnu projekciju lika uz tako zadano očište i gledište.



Slika 6.4: Perspektivna projekcija, analiza x-koordinate

Pogledajmo što se događa s pojedinim točkama. Očište (koje je smješteno u ishodištu) i vrhovi lika (promatramo sliku 6.4) spojeni su spojnicama koje probadaju i ravninu projekcije $z = H$. Na x- i y-koordinatnim osima označene su koordinate vrhova i probodišta ravnine projekcije. Analizu treba provesti za svaku koordinatu zasebno, no kako je rezultat identičan, u nastavku je na slici 6.5 prikazan slučaj za jedan vrh i to za x-koordinatu.



Slika 6.5: Perspektivna projekcija, analiza za x-os

Trokuti $0HT_P$ i $0T_zT$ su slični trokuti jer su pravokutni i dijele jednak kut ϕ. Tada vrijedi:

$$\frac{T_{P,x}}{H} = \frac{T_x}{T_z}$$

odakle slijedi:

$$T_{P,x} = T_x \cdot \frac{H}{T_z}.$$

Sličnom analiza za y-koordinatu projekcije slijedi:

$$\frac{T_{P,y}}{H} = \frac{T_y}{T_z} \quad \Rightarrow \quad T_{P,y} = T_y \cdot \frac{H}{T_z}.$$

Ovdje izvedeni odnosi mogu se zapisati i matrično, ako za sve točke iskoristimo njihove homogene inačice. Dobije se:

$$\pi_{persp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{H} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

U točnost se možemo uvjeriti pomnožimo li točku i matricu projekcije:

$$\begin{aligned} T_{Ph} &= [\ T_{Ph,x} \quad T_{Ph,y} \quad T_{Ph,z} \quad T_{Ph,h}\ ] \\ &= T_h \cdot \pi_{persp} \\ &= [\ T_x \quad T_y \quad T_z \quad 1\ ] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{H} \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ &= [\ T_x \quad T_y \quad 0 \quad \frac{T_z}{H}\ ] \end{aligned}$$

Prelaskom u radni prostor nakon dijeljenja s homogenim parametrom dobiju se upravo izrazi od kojih smo krenuli.

z-koordinata točke koja se dobije perspektivnom projekcijom preko prethodno izvedene matrice iznosi 0. Naime, kako točka leži u ravnini projekcije logično je za z-koordinatu uzeti vrijednost nula. No ponekad projekciju nećemo koristiti kao prijelaz iz 3D prostora u 2D prostor. U tom slučaju potrebno je projekciji svake točke ostaviti sve tri koordinate ispravnima: znači ako projiciramo na ravninu $z = H$, tada sve projicirane točke imaju z-koordinatu jednaku H. Matrica koja će raditi ovakvo projiciranje dobije se modifikacijom prethodne matrice:

$$\pi'_{persp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{H} \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

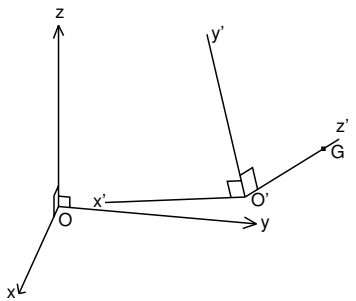
Kao i kod paralelne projekcije, i ovdje ćemo se zaustaviti i nećemo ići u kompliciranije slučajeve, odnosno u opći slučaj (što bismo vidjeli kada bi očište bilo negdje, a gledište negdje drugdje). Razlog tome je kompliciranost cijelog postupka ukoliko bismo išli ovako direktno, grubom silom. Umjesto toga, dovoljno je reći da se takav opći slučaj može postupkom transformacije pogleda svesti upravo na ovaj jednostavan. I stoga, pogledajmo što nam nudi transformacija pogleda...

6.2 Transformacije pogleda

Transformacije pogleda su postupci kojima se točke iz jednog koordinatnog sustava preslikavaju u drugi koordinatni sustav. Ideja je, kao i uvijek, krajnje jednostavna. Treba pronaći matricu 4×4 takvu da se množenjem proizvoljne točke i te matrice dobije točka s koordinatama koje bi originalna točka imala kada bismo je promatrali iz nekog drugog sustava. Postoji više načina kako se može izgraditi ovakva matrica. Najprije ćemo pogledati "intuitivni" način – primjenjivat ćemo elementarne transformacije koje smo već obradili kako bismo napravili traženo

preslikavanje. Pred kraj ovog poglavlja pokazat ćemo i drugi način koji se temelji direktno na spoznajama vektorskih prostora linearne algebre. Krenimo s prvim načinom, i to kroz primjere.

Slika 6.6 prikazuje dva koordinatna sustava. Pretpostavimo da su osi koordinatnog sustava x-y-z zadane jediničnim vektorima $\vec{i}$, $\vec{j}$ i $\vec{k}$ te da je njegovo ishodište u $(0, 0, 0)$. Ovo je *desni* koordinatni sustav (sjetite se što to znači) i njega ćemo smatrati osnovnim koordinatniom sustavom. Drugi koordinatni sustav ima ishodište O' koje je translatirano u odnosu na ishodište osnovnog koordinatnog sustava; osi su mu pri tome također proizvoljno zarotirane. Pretpostavimo također da je ovaj koordinatni sustav *lijevi*. Problem koji rješavamo je sljedeći: ako u osnovnom koordinatnom sustavu znamo koordinate neke točke, koje će biti njezine koordinate u ovom drugom koordinatnom sustavu?



Slika 6.6: Dva koordinatna sustava

Krenimo nakratko s jednostavnijim primjerom u 2D. Neka u koordinatnom sustavu S imamo točku T. Koje bi koordinate ta točka imala u sustavu S' koji bismo dobili kada bismo sustav S lagano trknuli tako da mu ishodište otklizi u točku O'? U sustavu S koordinate točke O' su (dx, dy). Problem je prikazan na slici 6.7 gdje su pomaci dx i dy označeni s Δx i Δy.

Sa slike 6.7 jasno se vidi veza između koordinata u oba sustava:

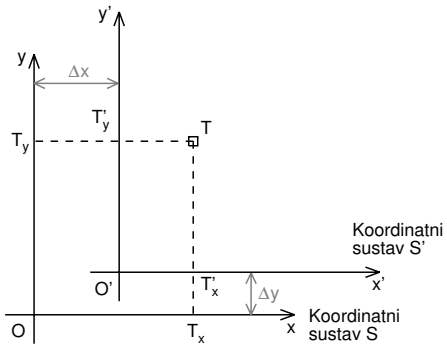
$$T_x = T_x^{'} + dx, \qquad T_y = T_y^{'} + dy$$

pa slijedi:

$$T_x^{'} = T_x - dx, \qquad T_y^{'} = T_y - dy.$$

Proširenjem na koordinatni sustav u 3D prostoru dobije se:

$$T_x^{'} = T_x - dx, \qquad T_y^{'} = T_y - dy, \qquad T_z^{'} = T_z - dz.$$

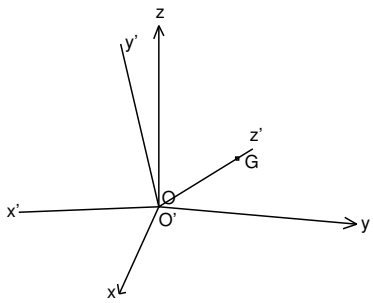


Slika 6.7: Translatirani koordinatni sustavi

Uočimo sada da je primjenjena transformacija upravo translacija za $(-dx, -dy, -dz)$. Stoga je prva matrica upravo matrica translacije:

$$\mathbf{\Theta}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -dx & -dy & -dz & 1 \end{bmatrix}.$$

Nakon primjene transformacije $\mathbf{\Theta}_1$ ishodišta obaju koordinatnih sustava su poklopljena – situacija je prikazana na slici 6.8.



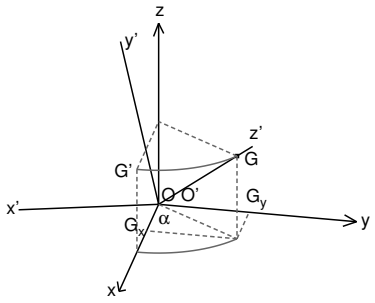
Slika 6.8: Dva koordinatna sustava nakon primjene transformacije $\mathbf{\Theta}_1$

Ovim razmatranjem naučili smo kako preslikati koordinate iz jednog koordinatnog sustava u drugi, ako je ishodište drugog koordinatnog sustava samo pomaknuto za neki vektor (dx, dy, dz). No što ukoliko je drugi koordinatni sustav umjesto pomaka uslijed našeg udarca iz gornjeg primjera doživio rotaciju?

Radi jednostavnosti pretpostaviti ćemo da mu je ishodište bilo učvršćeno u referentni sustav pa nije moglo doći do pomaka, već samo do rotacije. Ovaj rotirani sustav označiti ćemo sa S'. Kako će tada izgledati koordinate zadane točke T u tom sustavu?

U 2D sustavu svaka rotacija jednoznačno je određena kutom rotacije – jednim kutom. U 3D prostoru opis rotacije zahtjeva uporabu prostornog kuta – kuta koji se može u pravokutnom koordinatnom sustavu opisati pomoću dvije komponente kuta: kutom koji projekcija rotirane točke u xy-ravninu čini s x-osi koordinatnog sustava, te kutom koji projekcija rotirane točke u xz-ravninu čini sa z-osi koordinatnog sustava (zapravo, ovo nije jedina mogućnost; mogu se koristi bilo koja dva para ravnina i u njima odgovarajući kutovi). No unatoč ovako složenom opisu prostornog kuta, ideja rješenja je ista kao i kod prethodnog problema. U prethodnom slučaju rješenje smo dobili tako da smo pogledali što trebamo učiniti da bismo oba koordinatna sustava ponovno poklopili jedan preko drugoga; rješenje je bilo ishodištu novog sustava oduzeti vektor pomaka i time su se sustavi poklopili.

U ovom primjeru ishodišta se već poklapaju. Ono što se ne poklapa su osi. Ono što ćemo pokušati da bismo poklopili naše sustave jest sljedeće: uhvatit ćemo z-os novog sustava i zarotirati je tako da se poklopi sa z-osi starog sustava. Pri tome ćemo najprije z-os zarotirati za kut α u xy-ravnini i to u smjeru kazaljke na satu (dakle u matematički negativnom smjeru) tako da padne u xz-ravninu. Slika 6.9 prikazuje postupak.



Slika 6.9: Rotacija oko z-osi

Uzmimo da se točka $G = (G_x, G_y, G_z)$ nalazi na z-osi zarotiranog sustava S'. Kut α što ga projekcija te točke u xy-ravninu originalnog sustava zatvara s x-osi određen je izrazima:

$$\cos(\alpha) = \frac{G_x}{\sqrt{G_x^2 + G_y^2}}, \qquad \sin(\alpha) = \frac{G_y}{\sqrt{G_x^2 + G_y^2}}.$$

Rotiranjem za kut α točka G preslikat će se u točku G' koja leži u xz-ravnini. Pri tome su komponente točke G' određene izrazima:

$$G_x^{'} = \sqrt{G_x^2 + G_y^2}$$

$$G_y^{'} = 0$$

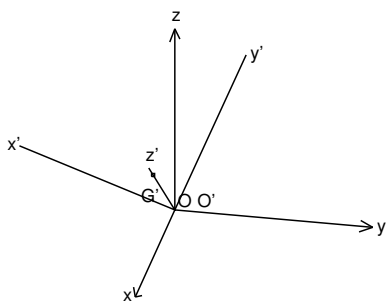
$$G_z^{'} = G_z$$

Rotacijom u xy-ravnini z-koordinata točke ostala je očuvana. y-koordinata pala je na nulu jer je točka rotacijom stigla u xz-ravninu. Kako se ovdje radi o rotaciji točke oko z-osi početnog koordinatnog sustava u *smjeru kazaljke na satu* (što je matematički negativan smjer odnosno odgovara kutu $-\alpha$), pripadna matrica rotacije glasi:

$$\mathbf{\Theta}_2 = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Ovdje treba spomenuti i specijalni slučaj koji može nastupiti. Ako je $G_x = 0$ i $G_y = 0$, tada se ova rotacija preskače, jer su z-osi obaju sustava već kolinearne te kut α nije definiran. Krenimo dalje.

Zajedno s točkom G koja se preslikala u G', z-os našeg rotiranog sustava također je pala u xz-ravninu referentnog sustava. Nova situacija prikazana je na slici 6.10.

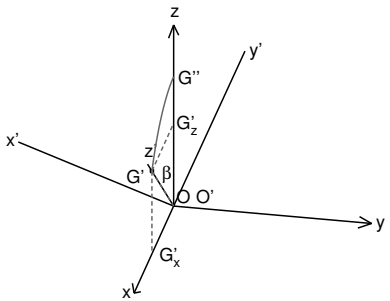


Slika 6.10: Dva koordinatna sustava nakon primjene transformacija $\mathbf{\Theta}_1$ i $\mathbf{\Theta}_2$

Još nam je preostalo da točku G' odvučemo na z-os referentnog koordinatnog sustava rotacijom u xz-ravnini, opet u smjeru kazaljke na satu (vidi sliku 6.11). Kut β koji predstavlja kut u xz-ravnini što ga točka G' zatvara s z-osi određen

je izrazima:

$$\cos(\beta) = \frac{G_z'}{\sqrt{(G_x')^2 + (G_z')^2}} \quad \sin(\beta) = \frac{G_x'}{\sqrt{(G_x')^2 + (G_z')^2}}.$$



Slika 6.11: Rotacija oko y-osi

Rotacijom točke G' dobiva se točka G'' koja leži i na z-osi referentnog sustava, te za njezine komponente vrijedi:

$$G_x'' = 0,$$

$$G_y'' = 0,$$

$$G_z'' = \sqrt{(G_x')^2 + (G_z')^2} = \sqrt{G_x^2 + G_y^2 + G_z^2}.$$

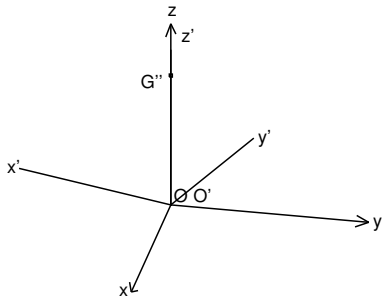
Ovim postupkom postigli smo poklapanje z-osi referentnog sustava s rotiranim sustavom – vidi sliku 6.12.

Matrični zapis ove transformacije glasi:

$$\mathbf{\Theta}_3 = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Pri izvođenju izraza za $\sin(\beta)$ i $\cos(\beta)$ specijalnog slučaja nema. Naime, pogreška bi nastupila samo ako bi G_x' i G_z' bili istodobno jednaki nuli, no to se ne može dogoditi osim u slučaju da se je polazna točka G poklapala s ishodištem referentnog sustava (što ne smije jer tada nismo niti definirali z-os).

Rotacijama $\mathbf{\Theta}_2$ i $\mathbf{\Theta}_3$ postigli smo ispravno poklapanje z-osi referentnog sustava i z-osi zarotiranog sustava. No što je s preostalim osima? Slika 6.12 ilustrira



Slika 6.12: Stanje nakon primjene transformacija $\mathbf{\Theta}_1$, $\mathbf{\Theta}_2$ i $\mathbf{\Theta}_3$

položaj svih šest osi: kako su osi z i z' poklopljene, osi x, x', y i y' sve leže u istoj ravnini. Budući da je osnovni koordinatni sustav bio desni a drugi koordinatni sustav lijevi – nemoguće je samo djelovanjem translacije i rotacije poklopiti sve osi. Bez ikakvih dodatnih podataka, možemo (potpuno proizvoljno, gledajući sliku 6.12) pretpostaviti da je između osi y i y' kut od 90ř (u smjeru kazaljke na satu). U tom slučaju, da bismo poklopili osi y i y', trebamo obaviti rotaciju oko osi z za 90ř u smjeru kazaljke na satu, što ćemo postići matricom:

$$\mathbf{\Theta}_4' = \begin{bmatrix} \cos(90\text{ř}) & -\sin(90\text{ř}) & 0 & 0 \\ \sin(90\text{ř}) & \cos(90\text{ř}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Nakon te transformacije osi z i z' te y i y' bit će poklopljene dok će osi x i x' biti kolinearne ali suprotne. Stoga će još trebati x koordinate točaka pomnožiti s -1 čime će se okrenuti smjer osi x'. Za to ćemo trebati transformaciju koja je opisana sljedećom matricom:

$$\mathbf{\Theta}_5 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

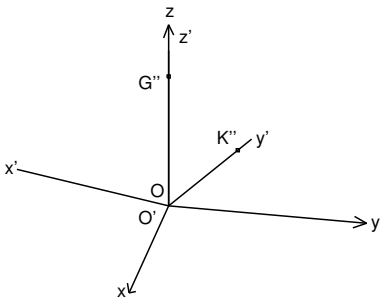
Međutim, umjesto proizvoljne pretpostavke o kutu između y i y', možemo od korisnika zatražiti još jedan podatak. Trodimenzionalni sustav jednoznačno je zadan ako je poznato npr. ishodište, točka na z-osi, točka na y-osi, te ako je poznato da je sustav primjerice desni. Točku ishodišta znamo – to je ista točka koja je i ishodište referentnog sustava. Točku na z-osi isto znamo: zadali smo je

kao točku G. Da bismo sustav definirali do kraja, pretpostavimo da je korisnik uz zadavanje očišta i gledišta zadao još i točku K koja leži negdje na osi y' drugog sustava, i riješimo problem do kraja.

Na sustav S' do ovog trena već smo djelovali rotacijama $\mathbf{\Theta}_2$ i $\mathbf{\Theta}_3$. Točka K uslijed tih rotacija prešla je u točku K'':

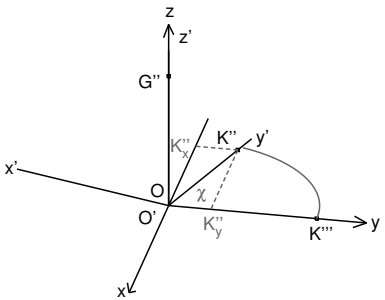
$$K_h'' = K_h \cdot \mathbf{\Theta}_2 \cdot \mathbf{\Theta}_3$$

pri čemu je K_h homogena inačica točke K; situacija je prikazana na slici 6.13.



Slika 6.13: Položaj točke K nakon primjene transformacija $\mathbf{\Theta}_1$, $\mathbf{\Theta}_2$ i $\mathbf{\Theta}_3$

Uslijed prethodno obavljenih rotacija z-komponenta točke K'' postala je nulom, dok su x- i y- komponente općenito različite od nule (barem jedna od njih). Situaciju prikazuje slika 6.14. Osi x' i y' rotiranog sustava sada leže u xy-ravnini originalnog koordinatnog sustava što slijedi iz činjenice da su im se osi z podudarile. To znači da je za podudaranje osi y i y' potrebna još samo jedna rotacija oko osi z.



Slika 6.14: Nova rotacija oko osi z kako bi točka K'' došla na os y

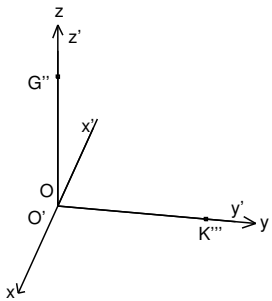
Kut χ koji određuje nepoklapanje između y- i y'- osi definiran je izrazima:

$$\cos(\chi) = \frac{K_y''}{\sqrt{\left(K_y''\right)^2 + \left(K_x''\right)^2}}, \quad \sin(\chi) = \frac{-K_x''}{\sqrt{\left(K_y''\right)^2 + \left(K_x''\right)^2}}.$$

Rotiranjem sustava S' oko osi z za kut χ postići ćemo potpuno poklapanje referentnog sustava i sustava S'. Matrica kojom se izvodi rotacija oko z-osi za kut χ glasi:

$$\mathbf{\Theta}_4 = \begin{bmatrix} \cos(\chi) & -\sin(\chi) & 0 & 0 \\ \sin(\chi) & \cos(\chi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

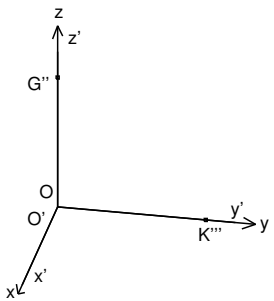
Rezultat ove rotacije prikazan je na slici 6.15.



Slika 6.15: Situacija nakon primjene transformacija $\mathbf{\Theta}_1$, $\mathbf{\Theta}_2$, $\mathbf{\Theta}_3$ i $\mathbf{\Theta}_4$

Kao i u alternativnom scenariju s pretpostavkom kuta od 90ř, završili smo s poklapanjem dviju od 3 osi pri čemu su osi x i x' postale kolinearne ali suprotnih smjerova što je posljedica činjenice da je originalni koordinatni sustav bio desni a drugi koordinatni sustav lijevi. Ovo možemo korigirati primjenom transformacije koja će okrenuti predznak x-komponenata svih točaka – prethodno definiranom transformacijom $\mathbf{\Theta}_5$. Stanje nakon primjene ove transformacije prikazano je na slici 6.16.

Pogledamo li koje smo sve transformacije primijenili za rješavanje problema rotiranog sustava, vidjeti ćemo da smo iskoristili transformacije $\mathbf{\Theta}_2$, $\mathbf{\Theta}_3$, $\mathbf{\Theta}_4$ ili $\mathbf{\Theta}_4^{'}$ te $\mathbf{\Theta}_5$. To znači da se cijeli posao može obaviti jednom transformacijom $\mathbf{\Theta}_2 \cdot \mathbf{\Theta}_3 \cdot \mathbf{\Theta}_4 \cdot \mathbf{\Theta}_5$ (odnosno $\mathbf{\Theta}_2 \cdot \mathbf{\Theta}_3 \cdot \mathbf{\Theta}_4^{'} \cdot \mathbf{\Theta}_5$). Dakle, ako imamo zadane koordinate točke T u referentnom sustavu i ako imamo zadanu jednu točku G koja leži na osi z našeg rotiranog sustava te točku K koja se nalazi na osi y našeg rotiranog sustava, tada ćemo koordinate točke T u rotiranom sustavu S' dobiti množenjem točke T i transformacijskih matrica.



Slika 6.16: Poklopljeni sustavi

Još nam je preostalo odgovoriti na najopćenitiji slučaj: kako bi glasile koordinate točke T u sustavu koji je uslijed "udarca" pretrpio i rotaciju, i translaciju? Tada mu se niti ishodišta, niti osi više ne poklapaju. Odgovor na ovo pitanje dat će nam kompozicija prethodnih slučajeva: prvo treba ishodište rotiranog sustava vratiti u ishodište referentnog sustava (matricom $\mathbf{\Theta}_1$), a zatim sustav treba još dodatno zarotirati matricom $\mathbf{\Theta}_2 \cdot \mathbf{\Theta}_3 \cdot \mathbf{\Theta}_4 \cdot \mathbf{\Theta}_5$. Pri tome treba obratiti pažnju na točke koje će se uzeti kao G točka i K točka za određivanje $\mathbf{\Theta}_2$, $\mathbf{\Theta}_3$ i $\mathbf{\Theta}_4$. Naime, kada matricom $\mathbf{\Theta}_1$ ishodište rotiranog sustava vratimo u ishodište referentnog sustava, tada i izvorne točke G i K treba translatirati istom matricom $\mathbf{\Theta}_1$ i tako dobivene točke treba uzeti za određivanje matrica $\mathbf{\Theta}_2$, $\mathbf{\Theta}_3$ i $\mathbf{\Theta}_4$.

U slučaju da su oba koordinatna sustava bila iste orijentacije (oba lijeva ili oba desna), tada bismo nakon transformacije $\mathbf{\Theta}_4$ dobili potpuno poklopljene sustave; u tom slučaju ne bismo koristili transformaciju $\mathbf{\Theta}_5$.

Ovime završavamo pregled transformacija pogleda.

6.3 Transformacija pogleda i perspektivna projekcija

U sekciji 6.1.2 pokazali smo osnove perspektivne projekcije. Uveli smo točku očišta kao onu točku gdje stavljamo naše oko, te točku gledišta kao onu točku koja predstavlja točku koja pripada ravnini projekcije, i koja ujedno tvori ishodište 2D koordinatnog sustava u toj ravnini. Dodatno smo rekli da je ravnina projekcije određena jednom pripadnom točkom (gledištem) i normalom (vektor očište-gledište). No tu smo stali. Kao primjer perspektivne projekcije uzeli smo najjednostavniji mogući slučaj: očište je u ishodištu koordinatnog sustava, a gledište je na z-osi i to točka $(0, 0, H)$ gdje je H udaljenost očišta od gledišta.

Kako bismo došli do općeg slučaja, razmotrit ćemo situaciju kada su očište i gledište proizvoljne točke u prostoru. Kako tada naći perspektivnu projekciju?

Pogledajmo još jednom što znamo. Znamo naći perspektivnu projekciju ako se očište nalazi u ishodištu, a gledište na z-osi. Međutim, u općem slučaju, točke očišta i gledišta su proizvoljne točke čije koordinate znamo u globalnom koordinatnom sustavu scene. Ono što nas zanima jest koje su koordinate objekata gledano iz novog koordinatnog sustava čije je ishodište u točki očišta te čija z-os prolazi kroz očište i gledište. Pogodili ste – trebamo transformaciju pogleda. Pa krenimo redom.

Očište ćemo u nastavku označavati s Q a gledište s R.

6.3.1 Korak 1

Prisjetimo li se transformacije pogleda, tada je prvi korak vraćanje novog ishodišta u ishodište referentnog sustava. Dakle, koristimo matricu $\mathbf{\Theta}_1$, uz pomake koji odgovaraju upravo koordinatama očišta. Naime, očište (ishodište novog koordinatnog sustava) trebamo vratiti u ishodište globalnog koordinatnog sustava. Transformacije ćemo raditi nad proizvoljnom točkom T.

$$\mathbf{\Theta}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -Q_x & -Q_y & -Q_z & 1 \end{bmatrix}.$$

Ova transformacija očište će preslikati u ishodište, dok će gledište preslikati u:

$$\begin{aligned} R_h^{'} &= R_h \cdot \mathbf{\Theta}_1 \\ &= [\ R_{h,x} \quad R_{h,y} \quad R_{h,z} \quad 1\ ] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -Q_x & -Q_y & -Q_z & 1 \end{bmatrix} \\ &= [\ R_{h,x} - Q_x \quad R_{h,y} - Q_y \quad R_{h,z} - Q_z \quad 1\ ]. \end{aligned}$$

Prema tome, za pojedine komponente vrijedi:

$$R_x^{'} = R_x - Q_x$$

$$R_y^{'} = R_y - Q_y$$

$$R_z^{'} = R_z - Q_z$$

6.3.2 Korak 2

Sada kada se ishodišta obaju sustava poklapaju, treba izvršiti rotaciju sustava tako da im se z-osi poklope. Prvi korak je rotacija u xy-ravnini matricom $\mathbf{\Theta}_2$.

$$\mathbf{\Theta}_2 = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

pri čemu vrijede izrazi koje smo već prethodno izveli:

$$\cos(\alpha) = \frac{G_x}{\sqrt{G_x^2 + G_y^2}}, \qquad \sin(\alpha) = \frac{G_y}{\sqrt{G_x^2 + G_y^2}}.$$

Kako je G točka upravo jednaka točki gledišta nakon prve transformacije: $G = R'$, pa se može pisati:

$$\cos(\alpha) = \frac{R_x^{'}}{\sqrt{\left(R_x^{'}\right)^2 + \left(R_y^{'}\right)^2}}, \qquad \sin(\alpha) = \frac{R_y^{'}}{\sqrt{\left(R_x^{'}\right)^2 + \left(R_y^{'}\right)^2}}.$$

Transformacija $\mathbf{\Theta}_2$ djelovanjem na točku R' daje točku R'':

$$\begin{aligned} R_h^{''} &= R_h^{'} \cdot \mathbf{\Theta}_2 \\ &= \begin{bmatrix} R_x^{'} & R_y^{'} & R_z^{'} & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} R_x^{'} \cdot \cos(\alpha) + R_y^{'} \cdot \sin(\alpha) & -R_x^{'} \cdot \sin(\alpha) + R_y^{'} \cdot \cos(\alpha) & R_z^{'} & 1 \end{bmatrix}. \end{aligned}$$

Uvrštavanjem izraza za $\sin(\alpha)$ i $\cos(\alpha)$ dobiva se:

$$R_x^{''} = \sqrt{\left(R_x^{'}\right)^2 + \left(R_y^{'}\right)^2}$$

$$R_y^{''} = 0$$

$$R_z^{''} = R_z^{'}$$

6.3.3 Korak 3

Sada je potrebno primijeniti i rotaciju u xz-ravnini transformacijom $\mathbf{\Theta}_3$.

$$\mathbf{\Theta}_3 = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Pri tome su:

$$\cos(\beta) = \frac{R_z''}{\sqrt{(R_x'')^2 + (R_z'')^2}} \quad \sin(\beta) = \frac{R_x''}{\sqrt{(R_x'')^2 + (R_z'')^2}}.$$

Transformacija $\mathbf{\Theta}_3$ djelovanjem na točku R'' daje točku R''', što po komponentama daje:

$$R_x''' = 0$$
$$R_y''' = 0$$
$$R_z''' = \sqrt{(R_x'')^2 + (R_z'')^2} = \sqrt{(R_x')^2 + \left(R_y'\right)^2 + (R_z')^2}$$

Primijenimo li još i transformaciju $\mathbf{\Theta}_4'$ (a po potrebi i $\mathbf{\Theta}_5$ ako su sustavi različito orijentirani), učinili smo sve potrebne transformacije kako bi sustav s ishodištem u očištu i točkom gledišta na njegovoj z-osi preveli u sustav kojemu se ishodište i z-os poklapaju s referentnim sustavom. Što smo time dobili? Ako znamo točku T u referentnom sustavu, tada su njezine koordinate u sustavu s ishodištem u očištu jednake:

$$T' = T \cdot \mathbf{\Theta}_1 \cdot \mathbf{\Theta}_2 \cdot \mathbf{\Theta}_3 \cdot \mathbf{\Theta}_4' \cdot \mathbf{\Theta}_5.$$

Budući da su nam time poznate koordinate u sustavu u kojem je očište jednako ishodištu, a gledište se nalazi na z-osi, tada točku T' znamo prethodno izvedenom jednostavnom matricom perspektivno projicirati, te možemo pisati:

$$T_p' = T' \cdot \phi_{persp} = T \cdot \mathbf{\Theta}_1 \cdot \mathbf{\Theta}_2 \cdot \mathbf{\Theta}_3 \cdot \mathbf{\Theta}_4' \cdot \mathbf{\Theta}_5 \cdot \phi_{persp}.$$

pri čemu je matrica perspektivne projekcije definirana kao:

$$\pi_{persp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{H} \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

H je pri tome udaljenost od očišta do gledišta pa se može izračunati iz tih podataka, ili se može primijetiti da smo tu udaljenost već izračunali prilikom transformacije pogleda, te se udaljenost nalazi kao z-komponenta točke R_z'''. Dakle, za H vrijedi:

$$\begin{aligned} H &= \sqrt{(Q_x - R_x)^2 + (Q_y - R_y)^2 + (Q_z - R_z)^2} \\ &= \sqrt{(R_x')^2 + \left(R_y'\right)^2 + (R_z')^2} \\ &= \sqrt{(R_x'')^2 + (R_z'')^2} \\ &= \sqrt{(R_z''')^2} \\ &= R_z''' \end{aligned}$$

Da smo željeli dobiti opću paralelnu projekciju, postupak bi bio identičan, samo bismo umjesto posljednje matrice perspektivne projekcije uzeli matricu paralelne projekcije.

U postupku se koristi matrica $\mathbf{\Theta}_4^{'}$ budući da prilikom zadavanja parametara za perspektivnu projekciju nismo zadali niti jedan podatak koji bi nam omogućio identificiranje položaja osi y. Ukoliko se želi potpuna kontrola nad slikom koju generira perspektivna projekcija, tada je potrebno zadati još i jednu točku K koja se nalazi na y-osi sustava u kojem radimo projekciju. Podsjetimo se još jednom zašto je to tako.

Kod opće perspektivne projekcije zadajemo dvije točke: očište Q i gledište R. Pri tome se stvara novi sustav s ishodištem u točci Q, i s pozitivnom z'-osi kroz točku R. Okomito na spojnicu R-Q u točki R stvara se ravnina projekcije, i lokalni dvodimenzionalni koordinatni sustav čije su osi x' i y'. Problem u svemu tome jest što specificiranjem samo točaka Q i R nemamo kontrolu nad x'- i y'-osima, tj. ne možemo nikako zadati: "želim da y'-os gleda baš ovako...". Jedna od mogućnosti da se dobije kontrola nad tim osima objašnjena je u poglavlju o transformacijama pogleda pomoću dodatne točke K. Ako prilikom zadavanja parametara perspektivne projekcije zadamo i točku K, tada ćemo umjesto matrice $\mathbf{\Theta}_4^{'}$ koristiti matricu $\mathbf{\Theta}_4$ čime ćemo dobiti kontrolu nad svim osima sustava. Implementacija ove jednostavne modifikacije ostavlja se čitateljima za vježbu. No postoji i drugi pristup.

6.3.4 View-up vektor

View-up vektor popularan je naziv za jedan od načina "kroćenja" svih osi sustava (dakle osi x' i y'). Evo ideje. Korisnik će prilikom zadavanja parametara npr. perspektivne projekcije zadati i vektor koji će pokazivati smjer y' osi. Ovo može biti dosta problematično jer jedinični vektor y' osi leži u ravnini projekcije, a korisnik prilikom zadavanja "što želi dobiti" ne mora sam računati sve – tome služe računala. Stoga je problem sljedeći: korisnik programa želi reći primjerice neka "os y' gleda prema gore". U prirodi "prema gore" obično označava u smjeru pozitivne z-osi, čiji je reprezentant npr. vektor $\vec{v}_{up} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. Pretpostavimo sada da su očište i gledište zadani tako da ravnina projekcije leži pod kutem od 45° prema ravnini xz, i okomita je na yz-ravninu. Vektor koji će u tom slučaju gledati "prema gore" biti će npr. $\vec{v} = \begin{bmatrix} 0 & -1 & 1 \end{bmatrix}$ jer on leži u ravnini projekcije (dok vektor $\vec{v}_{up}$ očito ne leži). Postavlja se pitanje kako uz približni vektor $\vec{v}_{up}$ koji zadaje korisnik pronaći pravi vektor $\vec{v}$ koji će pokazivati u smjeru osi y'. Zapravo, ono što nas u konačnici zanima jest, ako ćemo raditi cjelokupni postupak transformacije pogleda i projekcije, kako uz zadane točke očišta O i gledišta G te *view-up* vektor zadan od korisnika automatski odrediti točku K.

Jedan od najjednostavnijih načina jest uporabom vektorskog produkta. Označimo s $\vec{h}$ vektor koji predstavlja pozitivan smjer osi z sustava koji gradimo.

Njegovo ishodište je u točki O a pozitivna os z usmjerena je prema gledištu. Stoga vrijedi:

$$\vec{h} = G - O.$$

Os x sustava koji gradimo okomita je na ravninu kojoj pripadaju točke O i G i u kojoj leže i vektor $\vec{h}$ i vektor $\vec{v}_{up}$. Uzimajući u obzir da želimo izgraditi lijevi koordinatni sustav, os x koja se proteže u smjeru vektora $\vec{u}$ bit će definirana kao vektorski umnožak:

$$\vec{u} = \vec{h} \times \vec{v}_{up}.$$

Ovime smo već utvrdili dvije osi koordinatnog sustava: os z proteže se u smjeru vektora $\vec{h}$ a os x u smjeru vektora $\vec{u}$. Da bi konstruirani koordinatni sustav bio lijevi, os y koja mora biti okomita na prethodne dvije bit će određena vektorom $\vec{v}$ koji odgovara vektorskom produktu:

$$\vec{v} = \vec{u} \times \vec{h}.$$

Sada kada znamo smjer osi y, točku K možemo definirati kao bilo koju točku koja leži na pravcu:

$$K(\lambda) = O + \lambda \cdot \vec{v}$$

uz $\lambda > 0$; primjerice:

$$K = O + \vec{v}.$$

Nakon izračuna točke K može se krenuti u izgradnju matrica za opću perspektivnu projekciju koristeći pri tome matricu $\mathbf{\Theta}_4$. Uočimo još jednom: *view-up* vektor je vektor koji zadaje korisnik kako bi definirao smjer y-osi lokalnog 2D koordinatnog sustava razapetog u ravnini projekcije s ishodištem u gledištu. Tipično, *view-up* vektor ne mora ležati u ravnini projekcije; ravnina projekcije uvijek je okomita na spojnicu očište-gledište. Projekcija *view-up* vektora u ravninu projekcije daje vektor kolinearan jediničnom vektoru y-osi. U dodatku A.1 dan je još jedan način utvrđivanja točke K ako je zadan *view-up* vektor.

6.4 Transformacije pogleda na drugi način

6.4.1 Izvod

U prethodnom dijelu ovog poglavlja vidjeli smo jedan način prelaska između dva koordinatna sustava. U nastavku ćemo dati prikaz još jedne mogućnosti. Zanima nas koje će koordinate imati točka T u sustavu S' ako znamo koordinate točke T u referentnom sustavu S. Neka je referentni sustav S naš poznati osnovni sustav s osima x, y i z koje su u smjeru vektora $\vec{i}$, $\vec{j}$ i $\vec{k}$. Sustav S' možemo zadati na sljedeći način: ishodište mu je u točki C (čije koordinate znamo u sustavu S), pozitivna z-os zadana je vektorom $\vec{N}$, pozitivna y-os zadana je vektorom $\vec{V}$ a pozitivna x-os zadana je vektorom $\vec{U}$.

Prilikom zadavanja vektora $\vec{N}$, $\vec{V}$ i $\vec{U}$ treba paziti da su vektori međusobno okomiti i jedinični (znači, moraju biti ortonormirani). Ako zadani vektori nisu jedinični, treba ih prije uporabe normirati. Kako sada doći do koordinata točke T u sustavu S'? Prema već navedenom receptu, najprije treba ishodište sustava S' dovesti u ishodište sustava S. To ćemo postići matricom $\mathbf{\Theta}_1$ uz pomak koji odgovara negativnim koordinatama ishodišta sustava S' mjereno iz sustava S:

$$\mathbf{\Theta}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -C_x & -C_y & -C_z & 1 \end{bmatrix}.$$

Ovime smo postigli poklapanje ishodišta obaju sustava. Sada još treba očitati koordinate. No prije nego što se upustimo u to, prisjetimo se, što su zapravo koordinate? Koordinatni sustav zadan je svojim baznim vektorima. Ako od ishodišta krenemo malo po jednom vektoru, pa nakon toga krenemo po drugom vektoru, i tako po svim vektorima, došli smo u neku točku prostora. Koordinate te točke su upravo one "malo po jednom vektoru"-komponente. Točka u koju smo stigli može se predočiti vektorom (radij-vektorom): projekcije tog vektora na svaki bazni vektor daju upravo odgovor "koliko malo" smo se pomaknuli uzduž tih baznih vektora. Dakle, svaku koordinatu točke možemo dobiti tako da dani vektor projiciramo na odgovarajući bazni vektor. A duljina projekcije jednog vektora na drugi je upravo apsolutna vrijednost njihovog skalarnog produkta (ako je vektor na koji projiciramo jediničan; inače treba rezultat podijeliti s njegovom normom). Pogledajmo to na primjeru. U referentnom sustavu zadana je točka $T = (5, 2, 1)$. Pretvorbom u vektor dobivamo $5\vec{i} + 2\vec{j} + 1\vec{k}$ jer je referentni sustav upravo zadan vektorima $\vec{i}$, $\vec{j}$ i $\vec{j}$. Koliko iznosi x-komponenta točke? Množimo $(5\vec{i}+2\vec{j}+1\vec{k})$ skalarno s $(1\vec{i}+0\vec{j}+0\vec{k})$ i rezultat je upravo 5. y-komponenta dobije se množenjem $(5\vec{i}+2\vec{j}+1\vec{k})$ s $(0\vec{i}+1\vec{k}+0\vec{k})$ i rezultat je 2. z-koordinata dobije se množenjem $(5\vec{i}+2\vec{j}+1\vec{k})$ s $(0\vec{i}+0\vec{j}+1\vec{k})$ i rezultat je 1. Po ovoj analogiji može se pokazati da komponente točke T u sustavu S' odgovaraju upravo projekcijama vektora T na bazne vektore sustava S'. Kako je sustav S' zadan s tri bazna vektora: $\vec{U}$, $\vec{V}$ i $\vec{N}$ (koji su jedinični), vrijedi:

$$T_x^{'} = \vec{T} \cdot \vec{U} = T_x \cdot U_x + T_y \cdot U_y + T_z \cdot U_z,$$
$$T_y^{'} = \vec{T} \cdot \vec{V} = T_x \cdot V_x + T_y \cdot V_y + T_z \cdot V_z,$$
$$T_z^{'} = \vec{T} \cdot \vec{N} = T_x \cdot N_x + T_y \cdot N_y + T_z \cdot N_z.$$

Ovo se može i matrično zapisati, pa matrica $\mathbf{\Theta}_2$ glasi:

$$\mathbf{\Theta}_2 = \begin{bmatrix} U_x & V_x & N_x & 0 \\ U_y & V_y & N_y & 0 \\ U_z & V_z & N_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Ukupna matrica transformacije pogleda tada glasi: $\mathbf{\Theta} = \mathbf{\Theta}_1 \cdot \mathbf{\Theta}_2$.

U postupku smo pretpostavili da su zadana sva tri vektora: $\vec{U}$, $\vec{V}$ i $\vec{N}$. No lako je pokazati da ne moraju biti zadani baš svi vektori. Npr. ukoliko su zadani vektori $\vec{V}$ i $\vec{N}$, vektor $\vec{U}$ može se dobiti kao vektorski produkt vektora $\vec{V}$ i $\vec{N}$ (što će rezultirati desnom orijentacijom sustava S') ili kao vektorski produkt vektora $\vec{N}$ i $\vec{V}$ (što će rezultirati lijevom orijentacijom sustava S').

Nadalje, vektor $\vec{V}$ mora biti zadan tako da bude okomit na vektore $\vec{U}$ i $\vec{N}$. Ukoliko se korisniku dopusti da zada vektor koji je približno okomit, tada se pravi okomit vektor može izračunati iz poznatog *view-up* vektora.

Najslobodniji pristup dopustit će korisniku zadavanje trodimenzionalne točke C, vektora $\vec{N}$ kao pokazivača pozitivnog smjera osi z', vektor smjera $\vec{v}_{up}$ koji i ne mora biti baš okomit na vektor $\vec{N}$ te podatak želi li se lijevi ili desni sustav S'. U tom slučaju prikladan postupak računanja je sljedeći: prvo treba normirati vektor $\vec{N}$, zatim izračunati pravi vektor $\vec{V}$, te iz podatka o orijentaciji sustava treba izračunati vektor $\vec{U}$ kao vektorski produkt. Zatim se slože matrice $\mathbf{\Theta}_1$ i $\mathbf{\Theta}_2$ te ukupna matrica transformacije glasi: $\mathbf{\Theta} = \mathbf{\Theta}_1 \cdot \mathbf{\Theta}_2$.

Općeniti izvod direktne transformacije pogleda temeljem zadanih baznih vektora i koordinata ishodišta razrađen je u dodatku A u potpoglavlju A.2.

6.4.2 Primjena na perspektivnu projekciju

Točka C bit će očište. Gledište možemo zadati na dva načina.

1. Eksplicitnim zadavanjem točke gledišta R; tada se vektor $\vec{N}$ računa kao $R - C$ uz naknadno normiranje. Tada se za H može uzeti norma od $R - C$, ili se točka R može koristiti samo za određivanje vektora $\vec{N}$, dok se H i dalje zadaje proizvoljno.

2. Zadavanjem vektora $\vec{N}$ (koji po potrebi treba normirati), te zadavanjem udaljenosti H (u ovom slučaju H se mora zadati).

Uz ove podatke potrebno je zadati i *view-up* vektor, te željenu orijentaciju sustava.

6.5 Transformacija pogleda i projekcije u OpenGL-u

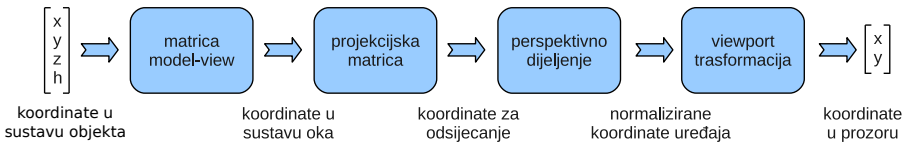
Nakon što smo se upoznali s matematičkim osnovama transformacija u 2D i 3D prostoru, njihovom uporabom kod transformacije pogleda i na kraju s projekcijama, pogledajmo kako je to sve organizirano u OpenGL-u. U ovoj sekciji naučit ćemo na koji način OpenGL radi transformacije vrhova i koje nam naredbe stoje na raspolaganju kojima možemo upravljati tim procesom. Pa krenimo redom.

Prisjetimo se – kada nešto crtamo u OpenGL-u, unutar bloka glBegin (...); /glEnd(); zadajemo jedan ili više vrhova kojima definiramo objekt koji postoji u sceni. Koordinate koje zadajemo su pri tome koordinate u trodimenzionalnom koordinatnom sustavu scene. Primjer funkcije koja crta žičani model kocke sa stranicom duljine w i s centrom smještenim u ishodište koordinatnog sustava dat je u nastavku. Tu funkciju koristit ćemo u ostatku ove sekcije u primjerima.

```
void kocka(float w) {
    float wp = w/2.0f;
    // gornja stranica
    glBegin(GL_LINE_LOOP);
    glVertex3f(-wp, -wp, wp);
    glVertex3f(wp, -wp, wp);
    glVertex3f(wp, wp, wp);
    glVertex3f(-wp, wp, wp);
    glEnd();

    // donja stranica
    glBegin(GL_LINE_LOOP);
    glVertex3f(-wp, wp, -wp);
    glVertex3f(wp, wp, -wp);
    glVertex3f(wp, -wp, -wp);
    glVertex3f(-wp, -wp, -wp);
    glEnd();

    // desna stranica
    glBegin(GL_LINE_LOOP);
    glVertex3f(wp, wp, -wp);
    glVertex3f(-wp, wp, -wp);
    glVertex3f(-wp, wp, wp);
    glVertex3f(wp, wp, wp);
    glEnd();

    // lijeva stranica
    glBegin(GL_LINE_LOOP);
    glVertex3f(wp, -wp, wp);
    glVertex3f(-wp, -wp, wp);
    glVertex3f(-wp, -wp, -wp);
    glVertex3f(wp, -wp, -wp);
    glEnd();

    // prednja stranica
    glBegin(GL_LINE_LOOP);
    glVertex3f(wp, -wp, -wp);
    glVertex3f(wp, wp, -wp);
    glVertex3f(wp, wp, wp);
    glVertex3f(wp, -wp, wp);
    glEnd();

    // straznja stranica
    glBegin(GL_LINE_LOOP);
```

```
    glVertex3f(-wp, -wp, wp);
    glVertex3f(-wp, wp, wp);
    glVertex3f(-wp, wp, -wp);
    glVertex3f(-wp, -wp, -wp);
    glEnd();
}
```

Nakon što OpenGL-u zadamo vrhove, nad svakim vrhom obavlja se niz transformacija kako bi se utvrdio njegov konačan položaj na ekranu. Proces je prikazan na slici 6.17 i predstavlja grubu strukturu *grafičkog protočnog sustava*.



Slika 6.17: Proces obrade vrhova pri generiranju konačne slike

Pojednostavljeno obradu možemo promatrati kao proces koji se odvija u četiri koraka. U prvom koraku nad vrhovima zadanim u prostoru scene primjenje se transformacija *model-view* matricom, čime se dobivaju koordinate u sustavu oka (tj. promatrača). Potom se primjenjuje projekcijska matrica, kako bi se dobile *clip*-koordinate. Ova transformacija ujedno definira i *volumen pogleda*, pa se svi objekti izvan tog volumena odsijecaju. Slijedi dijeljenje koordinata s homogenim parametrom kako bi se točke vratile u 3D radni prostor i preslikale na interval $[-1, 1]$; ovako dobivene koordinate zovemo normalizirane koordinate uređaja (engl. *normalized device coordinates*). Konačno, točke se primjenom *viewport*-transformacije prevode u koordinate prozora. U ovom koraku nacrtana se slika može još rastegnuti ili smanjiti kako bi stala u dio prozora odabran za njezin prikaz. Detaljnije ćemo se svakim od ovih koraka pozabaviti u nastavku.

Matrice koje se koriste za prva dva koraka OpenGL čuva zasebno, i omogućava manipuliranje svakom od njih nizom naredbi koje djeluju nad matricama. Kako je OpenGL stroj stanja, ta se činjenica koristi kako bi se smanjio broj parametara koji se predaju naredbama za manipuliranje s matricama. Naime, te naredbe ne primaju parametar koji bi rekao nad kojom se matricom djeluje; umjesto toga, OpenGL koristi pojam *odabrane matrice*, i sve matrične operacije djeluju nad tom matricom. Da bismo odabrali *model-view* matricu, potrebno je zadati naredbu:

```
glMatrixMode(GL_MODELVIEW);
```

Sve matrične naredbe koje zadamo nakon tog poziva modificirat će matricu *model-view*. Da bismo odabrali *projekcijsku* matricu, potrebno je zadati naredbu:

```
glMatrixMode(GL_PROJECTION);
```

To pak znači da će tipična konceptualna struktura OpenGL programa biti kako slijedi.

```
void crtanje() {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity(); // ponisti trenutne transformacije
    // dalje podesi projekcijsku matricu
    // ...
    // definiraj viewport(...)
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();  // ponisti trenutne transformacije
    // dalje podesi model-view matricu
    // ...
    nacrtajObjekteUSceni();
}
```

U stvarnosti, ovaj kod će biti raspodijeljen između više metoda. Podešavanje projekcijske matrice tipično će zajedno s definiranjem *viewport*-a biti smješteno u metodu reshape(...) jer ćemo tu trebati trenutne dimenzije prozora u kojem prikazujemo sliku, a promjene će se događati samo kada se mijenjaju dimenzije prozora. Podešavanje *model-view* matrice bit će pak smješteno u metodu display() jer se od trenutka do trenutka mogu mijenjati pozicije promatrača, smjer gledanja, položaji objekata u sceni i slično, pa tu matricu treba podešavati svaki puta kada se krene u crtanje slike.

Pogledajmo sada korake malo detaljnije.

6.5.1 Korak 1. Transformacije modela i pogleda

Nad koordinatama vrhova obavljaju se transformacije modela te transformacija pogleda. Transformacije modela su transformacije koje primjenjujemo kako bismo objekte pozicionirali na željeno mjesto u prostoru scene. Npr. pretpostavimo da trebamo nacrtati dva žičana modela kocke. Jedna kocka duljine stranice 10 mora se nalaziti u ishodištu. Druga kocka duljine stranice 5 mora biti zarotirana za 30° i pomaknuta za 10 u smjeru osi x. Jedna mogućnost jest ručno izračunati koordinate vrhova obaju točaka, i potom zadati niz glVertex3f(...) naredbi koje će OpenGL-u proslijediti te vrhove na crtanje. Bolja mogućnost jest iskoristiti našu postojeću funkciju kocka(...) koja uvijek crta kocku u ishodištu i podesiti OpenGL tako da vrhove koje zada ta funkcija transformira na željeni položaj (primjenom 3D transformacija). Za potrebe primjera pretpostavimo čak i da nemamo onako generičku funkciju kocka(w); koja može crtati kocku proizvoljne duljine stranice, već da imamo samo funkciju kocka1(); koja crta kocku duljine stranice 1. Implementacija te funkcije ovom općenitijom je trivijalna, i prikazana je u nastavku.

```
void kocka1() {
    kocka(1.0f);
}
```

Tražena dva objekta u scenu možemo dodati sljedećim isječkom koda.

```
void renderScene() {
    glColor3f(1.0f, 0.2f, 0.2f);
    glPushMatrix();
    glScalef(10.0f,10.0f, 10.0f);
    kocka1();
    glPopMatrix();

    glColor3f(0.0f, 0.2f, 1.0f);
    glPushMatrix();
    glTranslatef(10.0f, 0.0f, 0.0f);
    glRotatef(30.0f, 0.0f, 0.0f, 1.0f);
    glScalef(5.0f,5.0f, 5.0f);
    kocka1();
    glPopMatrix();
}
```

Metoda se sastoji od dva bloka koda. prvi blok nakon definiranja nijanse crvene boje na stog pohranjuje trenutnu *model-view* matricu (koja je u ovom trenutku još uvijek jednaka *view*-matrici). Naredbe koje slijede dio su definiranja *model*-matrice, no te se transformacije odmah primjenjuju na cjelokupnu *model-view* matricu. Matrica se modificira tako da se pomnoži matricom skaliranja s faktorom 10. Slijedi poziv funkcije kocka1(); koja stvara niz vrhova koji odgovaraju jediničnoj kocki. Međutim, te vrhove množi trenutna *model-view* matrica što rezultira 10× većom kockom od jedinične. Nakon toga, sa stoga se vraća originalna *model-view* matrica i prvi blok je gotov.

Crtanje druge kocke zahtjeva malo više posla. Postupak započinje definiranjem nijanse plave boje koja će se koristiti za crtanje, te pohranom trenutne *model-view* matrice na stog. Naš je zadatak sada transformirati kocku duljine stranice 1 u kocku duljine stranice 5 koja je još i zarotirana te pomaknuta po x-osi za 10. Za to trebamo tri naredbe: najprije naredbom glScalef (...) obavljamo skaliranje kocke. Potom primjenjujemo naredbu glRotatef (...) koja za 30° rotira kocku oko osi z (središte kocke je još uvijek u ishodištu pa je ovo korektno). Konačno, nakon što smo točke zarotirali, naredbom glTranslatef (...) ih pomičemo na konačnu poziciju – za deset po osi x. I tek tada pozivamo funkciju kocka1(); koja generira niz vrhova koji odgovaraju kocki duljine stranice 1 – na njih redom djeluje skaliranje, rotacija i zadnje translacija čime smo efektivno dobili kocku kakvu smo izvorno i htjeli. Uočimo još da je redosljed naredbi kako je napisan u izvornom kodu upravo obrnut od redosljeda koji smo upravo naveli. Ali to je u redu. Sjetimo se da zadnje definirana transformacija na točke djeluje prva, zbog toga što trenutnu transformacijsku matricu množi s desna.

U ovom trenutku trebali bismo već razlikovati dva tipa koordinata. Objekti su tipično zadani *koordinatama u prostoru objekta*. Tipično, ishodište tog koordinatnog sustava nalazi se u centru objekta a koordinate po svih komponentama

idu od -1 do 1. Primjer je naša jedinična kocka koja je omeđena ravninama paralelnim s xy-, xz- i yz-ravninama i s udaljenošću 0.5 do ishodišta. *model*-matricom koordinate tako zadanih objekata preslikavamo u *koordinate scene*; termin koji se u engleskom koristi jest *world-coordinates*. Zadatak *model*-matrice jest objekt pozicionirati na njegov stvarni položaj u sceni.

view-matrica zadužena je za transformaciju pogleda i tipično se definira prije poziva metode renderScene() – u metodi display(). Pojam *model-view*-matrica označava kompoziciju ovih dvaju matrica, i u stvarnosti, OpenGL i radi samo s jednom – konačnom – matricom. Definiranje *view*-matrice obavlja se u metodi display(), kako je prikazano u nastavku.

```
void display() {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, -20.0f);
    renderScene();
    glutSwapBuffers();
}
```

Nakon čišćenja pozadine, kao trenutna se matrica učitava matrica identiteta. Važno je pri tome uočiti da je u trenutku poziva metode display() odabran rad s matricom *model-view*. Naime, pred kraj metode reshape(...) poziva se glMatrixMode(GL_MODELVIEW); i nakon toga više u kodu nigdje ne mijenjamo odabranu matricu.

Transformacijom pogleda koju sada moramo napraviti trebamo definirati gdje se nalazi oko promatrača, te u kamo je pogled usmjeren. Inicijalno, OpenGL stavlja promatrača u ishodište koordinatnog sustava scene, okreće ga tako da gleda u smjeru $-z$ osi te smjer "gore" definira tako da se podudara s pozitivnom y-osi. Ako tu ostavimo promatrača, on će se naći usred prve kocke koju smo crtali, a to ne želimo. Stoga ćemo promatrača pomaknuti u točku $(0, 0, 20)$, tako da bude iznad obje kocke. Ovime je definiran koordinatni sustav oka (tj. promatrača) čije su osi kolinearne s odgovarajućim osima koordinatnog sustava scene, a ishodište je translatirano za $(0, 0, 20)$. U tom sustavu, sve x i y koordinate objekata ostaju očuvane, a z koordinate manje su za 20 – i time je upravo definirana potrebna transformacija pogleda, koju obavlja naredba glTranslatef(0.0f, 0.0f, −20.0f);.

Problem možemo riješiti i formalno. Naime, želimo da promatrač bude u točki $(0, 0, 20)$ te da gleda prema $-z$ osi. To znači da je očište $(0, 0, 20)$ a gledište, primjerice, $(0, 0, 0)$. Dodatno, *view-up* vektor je $(0, 1, 0)$, odnosno "gore" treba biti u smjeru porasta y-osi. Provedemo li postupak transformacije pogleda uz ove podatke, konačna matrica koju ćemo dobiti mora odgovarati upravo matrici translacije za -20 po osi z – provjerite.

Umjesto ručnog izračunavanja matrice transformacije pogleda, ili uporabe niza naredbi kojima opisujemo željenu translaciju i rotacije (koje nam trebaju za

transformaciju pogleda), u biblioteci GLU definirana je pomoćna naredba koju možemo iskoristiti za sve navedene izračune. Radi se o naredbi gluLookAt čiji je prototip dan u nastavku.

```
void gluLookAt(
    GLdouble eyex, GLdouble eyey, GLdouble eyez,
    GLdouble centerx, GLdouble centery, GLdouble centerz,
    GLdouble vupx, GLdouble vupy, GLdouble vupz);
```

Metoda gluLookAt prima očište, gledište i *view-up* vektor (za sve se zadaju svaka od koordinata zasebno), i temeljem toga izračunava matricu kojom množi trenutnu *model-view* matricu. Uporabom metode gluLookAt metodu display mogli smo napisati ovako:

```
void display() {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(0.0f, 0.0f, 20.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);
    renderScene();
    glutSwapBuffers();
}
```

Uporabom gluLookAt možemo vrlo jednostavno zadati proivoljnu lokaciju očišta, gledišta te *view-up* vektor, što znači da sami ne trebamo raditi kompleksan izračun matrica potrebnih za definiranje trnsformacije pogleda – ova će naredba to učiti za nas.

6.5.2 Korak 2. Projekcija

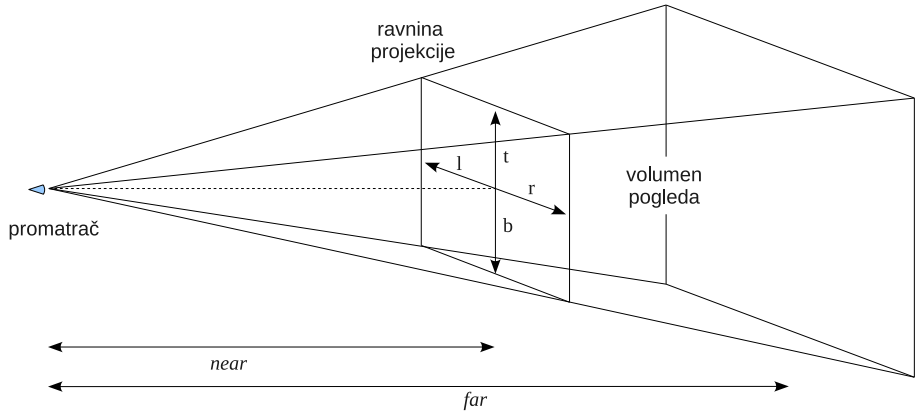
OpenGL kroz postojeće naredbe direktno podržava dvije vrste projekcija: perspektivnu i ortografsku. Iako je složenija, perspektivna se projekcija češće koristi, pa ćemo krenuti od nje.

Podrška za perspektivnu projekciju

Ugrađena naredba kojom se stvara matrica perspektivne projekcije je glFrustum. Njezin prototip prikazan je u nastavku.

```
void glFrustum(
    GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
    GLdouble near, GLdouble far);
```

Naredba definira ravninu projekcije koja je smještena ispred promatrača na udaljenosti near (mora biti pozitivan broj). Ravnina projekcije koja se stvara okomita je na poveznicu očište-gledište (slika 6.18). Mjereno od probodišta ravnine projekcije i pravca očište-gledište, stvara se pravokutno područje na kojem će se



Slika 6.18: Model koji koristi naredba *glFrustum*

stvoriti slika. To se područje od probodišta proteže left u lijevo, right u desno, bottom prema dolje i top prema gore. Sve što bi palo u ravninu projekcije ali izvan tog područja bit će odsiječeno. Konačno, na udaljenosti far od promatrača stvara se nova ravnina. Pri tome je uobičajeno far > near; ako je far < near, nastat će slika koja je zrcalna po osi z. Spojnice očišta i rubova pravokutnog područja u ravnini projekcije probadaju i tu drugu ravninu i time zatvaraju volumen pogleda – krnju piramidu, dio prostora koji je sprijeda ograđen ravninom projekcije, straga ravninom na udaljenosti far te uokolo navedenim spojnicama.

Uporabom ove naredbe možemo sada riješiti dio metode reshape();.

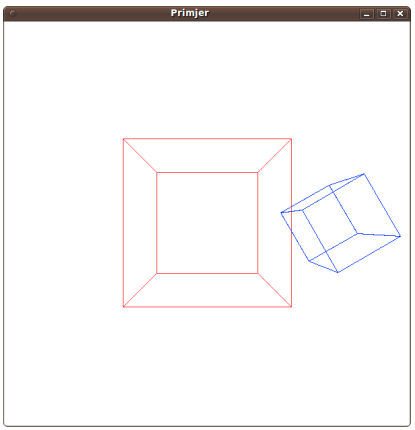
```
void reshape(int width, int height) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1.2f, 1.2f, -1.2f, 1.2f, 1.5f, 30.0f);
    glMatrixMode(GL_MODELVIEW);
    glViewport(0, 0, (GLsizei)width, (GLsizei)height);
}
```

Ulaskom u metodu reshape(); najprije odabiremo matricu projekcije. Potom je resetiramo na jediničnu matricu, i zatim pozivom metode glFrustum(...); definiramo površinu u ravnini projekcije koja je od promatrača udaljena 1.5. Ta se ravnina od probodišta sa spojnicom očište-gledište proteže 1.2 u svim smjerovima (prisjetimo se, promatrač je u točki $z = 20$ na osi z, i gleda prema ishodištu). Do prednje stranice veće kocke je udaljen za 15 a do njezine stražnje stranice za 25 (ta kocka ima stranice duljine 10). Stoga je kao *far* vrijednost odabrano 30, tako smo sigurni da je čitav objekt unutar volumena pogleda, i da će, shodno tome, biti i prikazan. Slika kojom rezultira ovaj program prikazana je na slici 6.19.

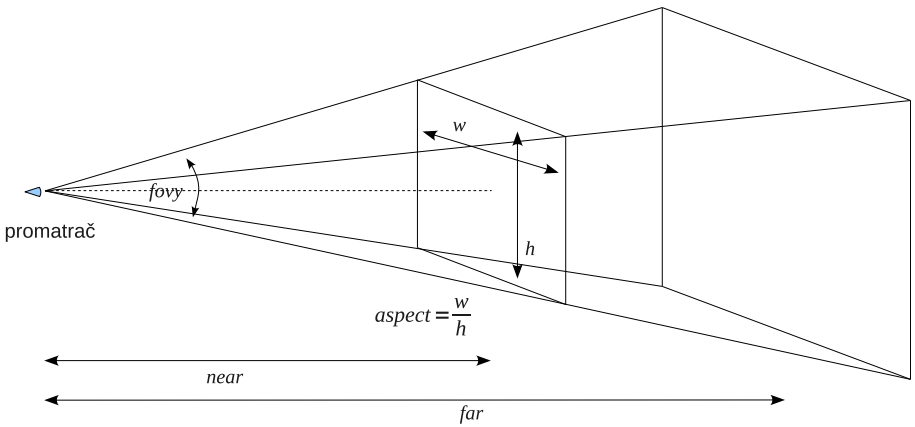
Biblioteka GLU nudi nam još jedan način definiranja matrice perspektivne projekcije – naredbu gluPerspective. Prototip naredbe prikazan je u nastavku.

```
void gluPerspective(
    GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);
```

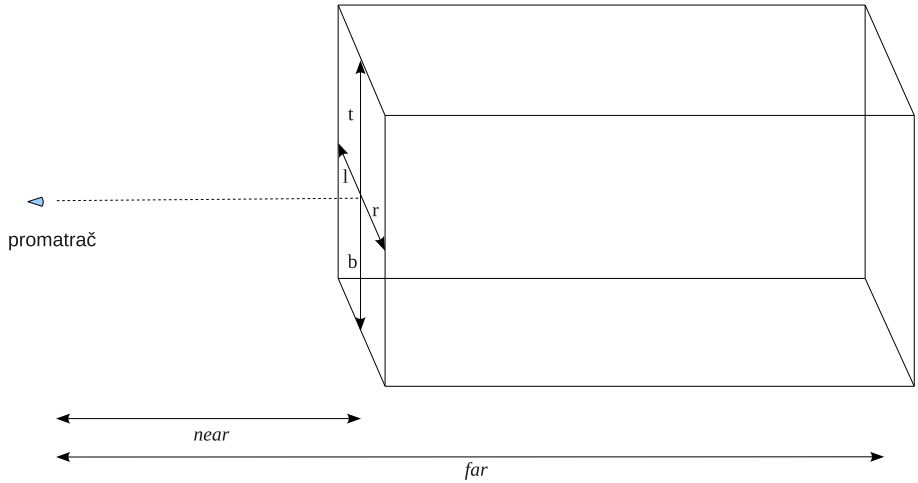
Naredba definira ravninu projekcije koja je smještena ispred promatrača na udaljenosti near (mora biti pozitivan broj). Ravnina projekcije koja se stvara okomita je na poveznicu očište-gledište (slika 6.20). Od probodišta ravnine projekcije tom spojnicom pravokutno područje proteže jednako i gore i dolje za $\text{near} \cdot \text{tg}\left(\frac{fovy}{2}\right)$, pri čemu je fovy (engl. *field-of-view*) kut u stupnjevima. Ukupna visina tada je određena izrazom $h = 2 \cdot \text{near} \cdot \text{tg}\left(\frac{fovy}{2}\right)$. Širina područja ne zadaje se direktno, već se određuje iz zadanog omjera širine i visine (što je dano parametrom aspect): $w = \text{aspect} \cdot h$. Konačno, volumen pogleda definiran je još i stražnjom ravninom koja se od promatrača nalazi na udaljenosti far.



Slika 6.19: Dvije kocke



Slika 6.20: Model koji koristi naredba *gluPerspective*



Slika 6.21: Model koji koristi naredba *glOrtho*

Podrška za ortografsku projekciju

Ugrađena naredba kojom se stvara matrica ortografske projekcije je glOrtho. Njezin prototip prikazan je u nastavku.

```
void glOrtho(
    GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
    GLdouble near, GLdouble far);
```

Naredba definira ravninu projekcije koja je smještena ispred promatrača na udaljenosti near (mora biti pozitivan broj). Ravnina projekcije koja se stvara okomita je na poveznicu očište-gledište (slika 6.21). Mjereno od probodišta ravnine projekcije i pravca očište-gledište, stvara se pravokutno područje na kojem će se stvoriti slika. To se područje od probodišta proteže left u lijevo, right u desno, bottom prema dolje i top prema gore. Sve što bi palo u ravninu projekcije ali izvan tog područja bit će odsiječeno. Konačno, na udaljenosti far od promatrača (pri čemu mora biti far $>$ near) stvara se nova ravnina. Spojnice rubova područja u ravnini projekcije paralelne sa z-osi probadaju i tu drugu ravninu i time zatvaraju volumen pogleda koji je u ovom slučaju kvadar.

6.5.3 Korak 3. Normalizacija koordinata

Nakon što su točke projicirane u ravninu projekcije, postupak normalizacije koordinata pretvara *clip*-koordinate x_c, y_c i z_c koordinate u raspon $[-1, 1]$. Nakon projekcije točke su još uvijek u homogenom prostoru. Konceptualno, to znači najprije sve komponente točke podijeliti homogenim parametrom h_c, čime se koordinate vraćaju u radni prostor. Potom se svaka od koordinata linearno mapira

na interval $[-1, 1]$. Primjerice, ako je korištena naredba glFrustum, prednja ravnina nalazi se na udaljenosti near od promatrača, i proteže lijevo, desno, dolje i gore za left , right , bottom i top. Pogledajmo najprije što se događa s x_c-koordinatom. Ona se iz intervala $[l, r]$ preslikava u $[-1, 1]$. Stoga možemo pisati:

$$\begin{aligned} x_{nd} &= \frac{x_c - l}{r - l} \cdot (1 - (-1)) - 1 \\ &= \frac{x_c - l}{r - l} \cdot 2 - 1 \\ &= \frac{2x_c - 2l}{r - l} - \frac{r - l}{r - l} \\ &= \frac{2x_c}{r - l} - \frac{r + l}{r - l} \end{aligned}$$

Analogno imamo izraz za dobivanje normalizirane y_{nd} koordinate.

$$y_{nd} = \frac{2y_c}{t - b} - \frac{t + b}{t - b}$$

Problem koji možda ovdje nije odmah očit jest u našoj pretpostavci – ako smo koristili naredbu glFrustum, tada normalizaciju radimo prema gornjim izrazima. Međutim, u ovom trenutku OpenGL više ne zna kako smo konstruirali matricu projekcije: je li to bila naredba glFrustum, ili naredba glProjection ili smo pak mi sami zadali direktno matricu element po element. Stoga nas ovakav pristup neće nigdje dovesti. OpenGL u ovom koraku radi samo jednu stvar: točke iz homogenog prostora pretvara u točke radnog prostora dijeljenjem s homogenim parametrom. Nakon toga očekuje se da su koordinate normalizirane. Da bi to bilo moguće, projekcijska matrica mora biti prikladno konstruirana tako da zadovolji taj zahtjev. To pak znači da će naredba koju koristimo za stvaranje projekcijske matrice morati o tome voditi računa. Sekcija 6.5.6 opisuje na koji naredba glFrustum gradi projekcijsku matricu, dok istu stvar za naredbu glOrtho sadrži sekcija 6.5.5. Nakon ovog koraka, sve točke koje imaju bilo koju koordinatu veću od 1 ili manju od -1 se odbacuju – takve točke predstavljaju točke koje su izvan volumena pogleda.

Zapamtimo: *clip*-koordinate kojima rezultira primjena projekcijske matrice iz prethodnog koraka su takve da povratkom u radni prostor (dijeljenjem s homogenim parametrom) rezultiraju normaliziranim koordinatama uređaja. Normalizirane koordinate uređaja imaju sve komponente u intervalu $[-1, 1]$.

6.5.4 Korak 4. *viewport* transformacija

Posljednji korak u crtanju scene jest odrediti gdje će se ta scena iscrtati u prozoru, odnosno uporaba transformacije koja točke iz normaliziranih koordinata uređaja

preslikava u ekranske koordinate. Kako bi bilo jasnije o čemu se radi, zamislite na tren da je kompletna slika nakon projekcije već nacrtana na pravokutnom području koje određuje bliža ravnina (engl. *near-plane*), tj. ravnina koja je od promatrača udaljena za near. Tu sliku sada treba negdje prekopirati na površinu prozora. I tu možemo birati – primjerice, hoćemo li sliku iskopirati preko čitave površine prozora, ili ćemo je nacrtati u gornjoj lijevoj četvrtini prozora, ili možda u donjoj desnoj četvrtini? Pravokutno područje unutar prozora u koje ćemo iskopirati sliku (uz eventualno potrebno rastezanje ili sažimanje) zove se *viewport*. U OpenGL-u *viewport* se definira naredbom glViewport(...), čiji je prototip prikazan u nastavku.

```
void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);
```

Pri tome su x i y koordinate relativne unutar prozora. $y = 0$ pri tome predstavlja dno prozora. Ovu naredbu tipično pozivamo unutar metode reshape jer se tada mijenjaju i dimenzije prozora, pa je potrebno prilagoditi i *viewport*. Primjerice, da bismo sliku preslikali na čitavu površinu prozora, pozvat ćemo naredbu:

```
glViewport(0, 0, width, height);
```

Detaljniji primjer pokazuje slika 6.22. Tako je na slici 6.22a prikazan rezultat kada se kao *viewport* odabere čitava površina ekrana. Na primjeru prikazanom na slici 6.22b, kao *viewport* je odabrana lijeva polovica prozora, pa je čitava slika komprimirana po x-osi tako da stane u taj dio. Na primjeru prikazanom na slici 6.22c, kao *viewport* je odabrana donja desna četvrtina prozora, pa je čitava slika komprimirana i po x-osi i po y-osi. Konačno, na primjeru prikazanom na slici 6.22d, *viewport* je širok i visok kao pola prozora, i smješten je za četvrtinu širine prozora od lijevog ruba prozora te za četvrtinu visine prozora od donjeg ruba prozora.

Transformacije koje se ovdje primjenjuju prikazane su u nastavku. x_w i y_w su koordinate u prozoru, x_{nd} i y_{nd} su normalizirane koordinate točke iz prethodnog koraka a x, y, *width* i *height* su parametri naredbe glViewport(...).

$$x_w = (x_{nd} + 1)\frac{w}{2} + x, \qquad y_w = (y_{nd} + 1)\frac{h}{2} + y.$$

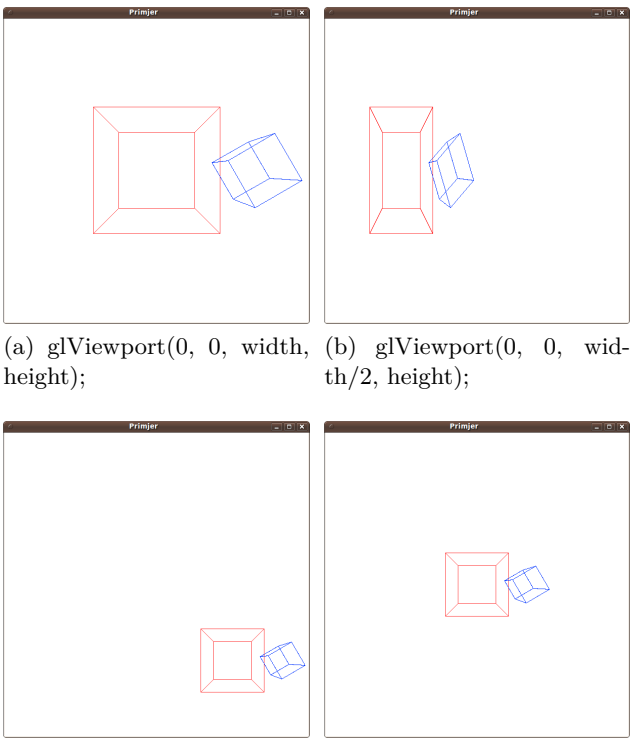
6.5.5 Izgradnja projekcijske matrice naredbe glOrtho

U ovoj podsekciji osvrnut ćemo se na prethodno postavljeni zahtjev da projekcijska matrica generira takve točke koje dijeljenjem homogenim parametrom odmah postaju normalizirane. Ovo ćemo pogledati na modelu zadavanja projekcije kakav koristi naredba glOrtho, a prikazan je na slici 6.21. Argumente naredbe glOrtho kraće ćemo pisati l, r, b, t, n, f. Oznake x_e, y_e i z_e koristit ćemo za koordinate točke u sustavu promatrača (dakle, nakon transformacije pogleda). Oznake x_p,

y_p i z_p koristit ćemo za koordinate točke u ravnini projekcije (koordinate odsijecanja, tj. *clip*-koordinate). Konačno, oznake x_{nd}, y_{nd} i z_{nd} koristit ćemo za normalizirane koordinate. Pa krenimo redom.

Nakon transformacije pogleda pretpostavka je da je ishodište koordinatnog sustava podudarno s promatračem, te da promatrač gleda u smjeru $-z$ osi. Međutim, parametri n i f zadaju se kao udaljenosti od promatrača, a z-koordinate dobivenih projiciranih točaka također želimo koristiti kao mjeru udaljenosti točke do promatrača. Ovo implicira koordinatni sustav u kojem udaljavanjem u smjeru pogleda z koordinate rastu, a ne padaju, pa ćemo u formulama po potrebi ubaciti promjenu predznaka. Ako je promatrač u ishodištu a ravnina projekcije na udaljenosti n od njega, točka (x_e, y_e, z_e) paralelno se transformira prema izrazima:

$$x_p = x_e, \qquad y_p = y_e, \qquad z_p = -z_e.$$



(a) glViewport(0, 0, width, height);

(b) glViewport(0, 0, width/2, height);

(c) glViewport(width/2, 0, width/2, height/2);

(d) glViewport(width/4, height/4, width/2, height/2);

Slika 6.22: Utjecaj odabranog *viewport*-a na konačni prikaz slike

Krenimo u normiranje ovih koordinata. x_p želimo preslikati iz raspona $[l, r]$ u $[-1, 1]$ a y_p želimo preslikati iz raspona $[b, t]$ u $[-1, 1]$. Izraze za ovo već smo izveli u prethodnoj sekciji. Uvažavanjem činjenice da je $x_p = x_e$ te $y_p = y_e$ dobivamo:

$$x_{nd} = x_c = \frac{2 \cdot x_e}{r - l} - \frac{r + l}{r - l}$$
$$y_{nd} = y_c = \frac{2 \cdot y_e}{t - b} - \frac{t + b}{t - b}$$

z_p želimo preslikati iz raspona $[n, f]$ u $[-1, 1]$, a to odgovara preslikavanju z_e iz raspona $[-n, -f]$ u $[-1, 1]$. Trivijalno je pokazati da se to postiže izrazom:

$$z_{nd} = z_c = \frac{-2 \cdot z_e}{f - n} - \frac{f + n}{f - n}.$$

Homogeni parametar h_c postavit ćemo na 1. Time je u potpunosti definirana i projekcijska matrica π_{par}.

$$\pi_{par} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{-2}{f-n} & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & -\frac{f+n}{f-n} & 1 \end{bmatrix}$$

6.5.6 Izgradnja projekcijske matrice naredbe glFrustum

U ovoj podsekciji osvrnut ćemo se na prethodno postavljeni zahtjev da projekcijska matrica generira takve točke koje dijeljenjem homogenim parametrom odmah postaju normalizirane. Ovo ćemo pogledati na modelu zadavanja projekcije kakav koristi naredba glFrustum, a prikazan je na slici 6.18. Argumente naredbe glFrustum kraće ćemo pisati l, r, b, t, n, f. Oznake x_e, y_e i z_e koristit ćemo za koordinate točke u sustavu promatrača (dakle, nakon transformacije pogleda). Oznake x_p, y_p i z_p koristit ćemo za koordinate točke u ravnini projekcije (koordinate odsijecanja, tj. *clip*-koordinate). Konačno, oznake x_{nd}, y_{nd} i z_{nd} koristit ćemo za normalizirane koordinate. Pa krenimo redom.

Nakon transformacije pogleda pretpostavka je da je ishodište koordinatnog sustava podudarno s promatračem, te da promatrač gleda u smjeru $-z$ osi. Međutim, parametri n i f zadaju se kao udaljenosti od promatrača, a z-koordinate dobivenih projiciranih točaka također želimo koristiti kao mjeru udaljenosti točke do promatrača. Ovo implicira koordinatni sustav u kojem udaljavanjem u smjeru pogleda z koordinate rastu, a ne padaju, pa ćemo u formulama po potrebi ubaciti promjenu predznaka. Ako je promatrač u ishodištu a ravnina projekcije na udaljenosti n od njega, točka (x_e, y_e, z_e) perspektivno se transformira prema izrazima:

$$x_p = \frac{n \cdot x_e}{-z_e}, \qquad y_p = \frac{n \cdot y_e}{-z_e}.$$

Kako x_p i y_p treba dijeliti s $-z_e$, odabrat ćemo da je homogeni parametar w_p upravo jednak $-z_e$ pa točke sada nećemo dijeliti, već će se to dogoditi u trenutku kada krenemo vraćati točke u radni prostor. Međutim, ostavimo za sada formule takve kako su napisane. Normirane koordinate dobit ćemo preslikavanjem x_p s intervala $[l, r]$ na interval $[-1, 1]$ i y_p s intervala $[b, t]$ na interval $[-1, 1]$. Već smo izveli izraze koji to opisuju:

$$x_{nd} = \frac{2x_p}{r-l} - \frac{r+l}{r-l}, \qquad y_{nd} = \frac{2y_p}{t-b} - \frac{t+b}{t-b}.$$

Uvrštavanjem prethodnih izraza za x_p i y_p u ove dobije se:

$$x_{nd} = \frac{\frac{2 \cdot n \cdot x_e}{r-l} + \frac{r+l}{r-l} \cdot z_e}{-z_e}, \qquad y_{nd} = \frac{\frac{2 \cdot n \cdot y_e}{t-b} + \frac{t+b}{t-b} \cdot z_e}{-z_e}.$$

clip-koordinate x_c i y_c definirane su kao brojnici razlomaka prethodnih izraza:

$$x_c = \frac{2 \cdot n \cdot x_e}{r-l} + \frac{r+l}{r-l} \cdot z_e, \qquad y_c = \frac{2 \cdot n \cdot y_e}{t-b} + \frac{t+b}{t-b} \cdot z_e.$$

Kako je $h_c = h_p = -z_e$, povratkom iz *clip*-koordinata u radni prostor računamo $\frac{x_c}{h_c}$ što je upravo x_{nd}, te $\frac{y_c}{h_c}$ što je upravo y_{nd}. Ostalo je još izračunati z_c. Kod klasične perspektivne projekcije, pisali bismo $z_p = -n$. Međutim, z_p želimo iskoristiti za mjeru udaljenosti točke do promatrača, kako bismo kasnije mogli odbacivati točke koje su iza točaka koje smo već nacrtali (kada ćemo raditi uklanjanje skrivenih površina). Kod perspektivne projekcije znamo da $z_p = z_c$ ne ovisi o x_e i y_e. To znači da može ovisiti još samo o z_e (ako pretpostavimo da točka T_e ima homogeni parametar jednak 1). Pretpostavimo da je ta ovisnost linearna, tj. da možemo uvesti neke dvije konstante A i B, i pisati $z_c = A \cdot z_e + B$. Normalizirana z_{nd} koordinata (kao i sve druge) dobije se dijeljenjem s homogenim parametrom h_c, a njega smo odabrali da je $-z_e$. Stoga možemo pisati:

$$z_{nd} = \frac{A \cdot z_e + B}{-z_e}.$$

Želimo da vrijednost normalizirane z_{nd} koordinate bude jednaka -1 ako je $z_e = -n$ (ako leži na bližoj ravnini), odnosno da vrijednost normalizirane z_{nd} koordinate bude jednaka $+1$ ako je $z_e = -f$ (ako leži na daljoj ravnini). Iz ovoga dobivamo sustav jednadžbi:

$$-1 = \frac{A \cdot (-n) + B}{-(-n)} = \frac{-A \cdot n + B}{n},$$

$$+1 = \frac{A \cdot (-f) + B}{-(-f)} = \frac{-A \cdot f + B}{f}.$$

čije je rješenje:

$$A = -\frac{f+n}{f-n}, \qquad B = \frac{-2 \cdot f \cdot n}{f-n}.$$

Uvrštavanjem u izraz za z_{nd} konačno dobivamo:

$$z_{nd} = \frac{-\frac{f+n}{f-n} \cdot z_e + \frac{-2 \cdot f \cdot n}{f-n}}{-z_e}.$$

Brojnik ovog izraza proglasit ćemo z_c. Time smo definirali izraze za sve komponente *clip*-koordinata u homogenom prostoru. Evo ih još jednom:

$$x_c = \frac{2 \cdot n \cdot x_e}{r-l} + \frac{r+l}{r-l} \cdot z_e, \qquad y_c = \frac{2 \cdot n \cdot y_e}{t-b} + \frac{t+b}{t-b} \cdot z_e,$$
$$z_c = -\frac{f+n}{f-n} \cdot z_e + \frac{-2 \cdot f \cdot n}{f-n}, \qquad h_c = -z_e.$$

Tada je projekcijska matrica π_{persp} koja točku T_e projicira u točku T_c na način $T_c = T_e \cdot \pi_{persp}$ definirana na sljedeći način.

$$\pi_{persp} = \begin{bmatrix} \frac{2 \cdot n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2 \cdot n}{t-b} & 0 & 0 \\ \frac{r+l}{r-l} & \frac{t+b}{t-b} & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & \frac{-2 \cdot f \cdot n}{f-n} & 0 \end{bmatrix}$$

Prikazana matrica je upravo matrica koju prema dokumentaciji i gradi metoda glFrustum[1]. Naravno, s obzirom da OpenGL množi matricu točkom, u stvarnosti koristi transponiranu matricu od ove koju smo izveli.

6.6 Ponavljanje

1. Pojasnite model ortografske projekcije. Kako se izvodi matrica ortografske projekcije?

2. Pojasnite model perspektivne projekcije. Kako se izvodi matrica perspektivne projekcije?

3. Kako se provodi postupak transformacije pogleda? Što je početno zadano, a što je rezultat tog postupka?

4. Je li transformacija pogleda jednoznačno određene zadavanjem samo očišta i gledišta? Što je i čemu služi view-up vektor?

5. Koju funkciju u OpenGL-u možemo koristiti za transformaciju pogleda?

[1] Dokumentacija s adrese `http://www.opengl.org/sdk/docs/man/xhtml/glFrustum.xml`.

6. Koje funkcije u OpenGL-u možemo koristiti za projekciju?

7. Što se događa tijekom normalizacije koordinata nakon postupka projekcije? Što su clip-koordinate?

8. Što se događa tijekom viewport transformacije?

Poglavlje 7

Krivulje

7.1 Uvod

Pojam krivulje promatrat ćemo u najširem obliku: krivulja je niz točaka (uz dodatne uvjete koji pobliže opisuju specifični tip krivulje). U ovom dijelu teksta pozabaviti ćemo se načinima zadavanja krivulja, klasifikacijama krivulja te poželjnim svojstvima krivulja.

7.1.1 Načini zadavanja krivulja

Krivulje obično zadajemo analitički – formulom (ili formulama). Tako imamo tri osnovne kategorije zapisa krivulje.

- *Eksplicitnom jednadžbom* $y = f(x)$. Problemi koji se pri tome javljaju svakom su poznati. Primjerice, jednadžba pravca u eksplicitnom obliku glasila je $y = ax + b$. Međutim, ovaj oblik zapisivanja imao je i ozbiljnih problema; najbanalniji jest nemogućnost prikaza pravca paralelnog s y-osi. Drugi primjer je jednadžba kružnice, koja se uobičajeno zapisuje u obliku: $y = \pm\sqrt{R^2 - x^2}$ – ne iz razonode, već jednostavno iz činjenice da se eksplicitnim oblikom ne mogu prikazati funkcije s višestrukim vrijednostima. Kod kružnice je to još nekako prolazilo dodavanjem znaka $\pm$ ispred korijena i prešutnim prihvaćanjem da tako zapravo radimo s dvije krivulje a ne s jednom.

- *Implicitnom jednadžbom* $f(x, y) = 0$. Ovakav oblik jednadžbe može prikazati višestruke vrijednosti, no ne može dati djelomičan prikaz. Npr. implicitni oblik jednadžbe kružnice je $x^2 + y^2 = R^2$. Ovime su obuhvaćene sve točke koje pripadaju kružnici, no sada riječ *sve* počinje predstavljati problem. Ponekad želimo prikazati samo npr. četvrt kružnice, što na ovaj način ne možemo specificirati.

- *Parametarskim zapisom.* Pri tome se uvodi jedan ili više parametara, te se sve koordinate opisuju kao eksplicitne funkcije tih parametara. Npr. za kružnicu možemo pisati ovako: parametar je t a koordinate su zadane izrazima $x = \cos(t)$, $y = \sin(t)$ uz $0 \leq t \leq 2\pi$. Ovaj oblik također nudi i mogućnost zadavanja samo određenog dijela krivulje. Tako primjerice, da bismo definirali samo prvu četvrtinu kružnice, parametar t umjesto na interval $0 \leq t \leq 2\pi$ treba ograničiti na interval $0 \leq t \leq \frac{\pi}{2}$.

Neovisno o analitičkom prikazu krivulje, krivulje možemo zadavati prema sljedećim kriterijima:

- tako da prolaze kroz zadane točke,
- tako da im definiramo vrijednosti prve derivacije u pojedinim točkama (tangente) te
- tako da im definiramo vrijednosti viših derivacija.

Pri tome se mogu ravnopravno koristiti i kombinacije navedenih kriterija; npr. možemo tražiti da krivulja prolazi kroz n zadanih točaka te da tangenta u prvoj točki bude pod kutom $\frac{\pi}{4}$. Prilikom ovakvog zadavanja treba imati u vidu da ponekad nije jednostavno dobiti sve potrebne parametre krivulje koja će ispunjavati ovakve miješane kriterije.

7.1.2 Klasifikacija krivulja i poželjna svojstva

Krivulje možemo klasificirati prema različitim kriterijima. Najčešće su klasifikacije sljedeće:

- periodičke $\leftrightarrow$ neperiodičke,
- racionalne $\leftrightarrow$ neracionalne te
- otvorene $\leftrightarrow$ zatvorene.

U nastavku nabrojimo još i svojstva koja želimo kod krivulja.

- Mogućnost prikaza višestrukih vrijednosti (za jedan x više mogućih y vrijednosti).
- Nezavisnost od koordinatnog sustava.
- Svojstvo lokalnog nadzora (promjena jedne točke uzrokuje promjenu oblika krivulje samo u okolini te točke).
- Smanjenje varijacije (zbog visokog stupnja polinoma krivulje ponekad se umjesto glatke krivulje pojavljuje nepoželjno istitravanje).

- Red neprekidnosti što veći (više o ovome u nastavku).
- Mogućnost opisa osnovnih geometrijskih oblika poput kružnice, elipse i sl.

7.1.3 Svojstvo neprekidnosti krivulja

Pojam *red neprekidnosti* dolazi iz matematičke analize i odnosi se na funkcije. Kako se krivulje zadaju također kao funkcije, treba pogledati na što se odnose redovi neprekidnosti u ovom slučaju, odnosno koju imaju interpretaciju. U nastavku ćemo razmotriti C-neprekidnosti. Uz te pojmove koriste se i istoznačnice C-kontinuiteti (engl. *C-continuity*) što ćemo također koristiti u nastavku teksta.

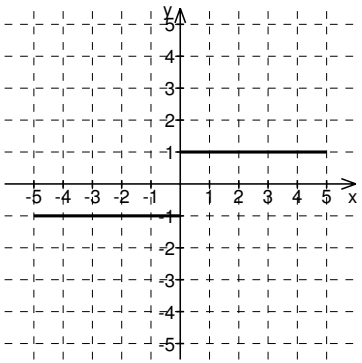
C-neprekidnosti

- C^0-kontinuitet (čita se "ce nula", ne "ce na nultu"). Zahtjeva se neprekidnost u koordinatama. Ovo je, funkcijski gledano, zahtjev na funkciju da nema diskontinuitete. Npr. funkcija $abs(x)$ je funkcija C^0-kontinuiteta dok funkcija $sign(x)$ nije funkcija C^0-kontinuiteta zbog skoka u $x = 0$.
- C^1-kontinuitet. Zahtjeva se neprekidnost (općenito nejediničnog) tangencijalnog vektora u svim točkama, tj. zahtijeva se neprekidnost prve derivacije. Drugim riječima, ne smije biti šiljaka, već krivulja mora biti glatka. Tako funkcija $abs(x)$ nije funkcija C^1-kontinuiteta, iako ima C^0-kontinuitet. Razlog je nagla promjena derivacije u ishodištu: prilazimo li ishodištu s lijeve strane, derivacija je -1; prilazimo li ishodištu s desne strane, derivacija je $+1$; u ishodištu se događa skok zbog kojeg ova funkcija nema C^1-kontinuitet.
- C^2-kontinuitet. Zahtjeva se neprekidnost druge derivacije u svim točkama. Ovaj kontinuitet govori nam o neprekidnosti u zakrivljenosti.
- C^3-kontinuitet. Zahtjeva se neprekidnost treće derivacije u svim točkama. Može se opisati kao neprekidnost u izvijanju.

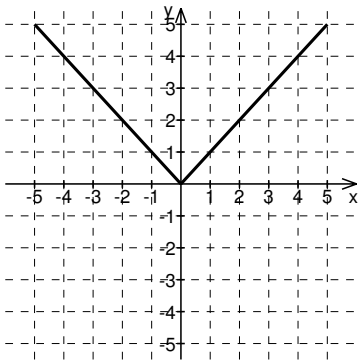
Spomenimo i da se za funkcije C^i-kontinuiteta kraće kaže da su funkcije klase C^i.

Nekoliko primjera dano je na slici 7.1. Slika 7.1a prikazuje funkciju $f(x) = sign(x)$. Ova funkcija nema čak niti C^0-kontinuitet jer u točki $x = 0$ ima prekid. Slika 7.1b prikazuje funkciju $f(x) = abs(x)$. Ova funkcija je kalse C^0 ali nije klase C^1: derivacija ove funkcije je upravo funkcija $sign(x)$ pa derivacija ima prekid u točki $x = 0$. Slično vrijedi i za po dijelovima definiranu funkciju prikazanu na slici 7.1c:

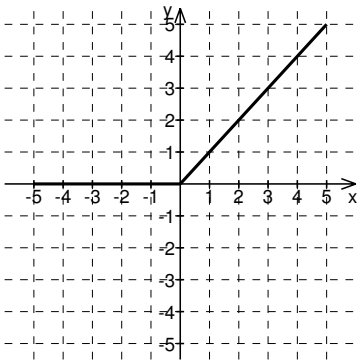
$$f(x) = \begin{cases} x, & x >= 0 \\ 0, & \text{inače} \end{cases}$$



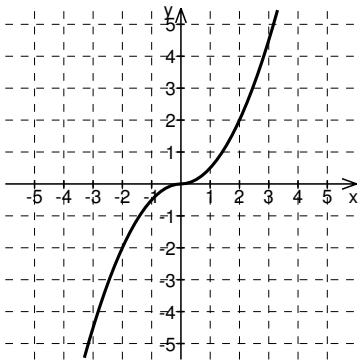
(a) Nema čak niti C^0



(b) Ima C^0 ali nema C^1



(c) Ima C^0 ali nema C^1



(d) Ima C^0 i C_1 ali nema C^2

Slika 7.1: Primjeri funkcija koje imaju različite C-kontinuitete.

kod koje derivacija također ima prekid u točki $x = 0$. Konačno, funkcija prikazana na slici 7.1d ima C^0 i C^1-kontinuitet. Radi se o funkciji:

$$f(x) = sign(x) \cdot \frac{x^2}{2}.$$

Funkcija je neprekidna i njezina prva derivacija je također neprekidna (derivacija ove funkcije je $abs(x)$); druga derivacija ove funkcije je $sign(x)$ koja ima prekid u $x = 0$ pa funkcija stoga nema C^2-kontinuitet.

C^2 kontinuitet je povezan sa zakrivljenosti krivulje. Zamislite da u zadanoj točki krivulje umjesto pravca koji dira krivulju u toj točki trebate konstruirati tangencijalnu kružnicu: kružnicu koja dira krivulju u toj točki i koja ima to

veći radijus što je u okolici te točke krivulja sličnija pravcu. Formalno, ovu kružnicu možemo zamisliti kao kružnicu čiji centar leži na sjecištu pravaca koje definiraju dvije normale na krivulju u točkama koje su beskonačno blizu zadanoj točki i također leže na krivulji, jedna s lijeve strane a druga s desne. Radijus odgovara udaljenosti od tako dobivenog centra pa do zadane točke na krivulji. Ako je krivulja pravac, promatramo li neku točku T, pravci određeni normalama u točkama $T+\epsilon$ i $T-\epsilon$ bit će paralelni i nikada se neće sjeći: radijus je beskonačan. Ako je pak krivulja kružnica, svi se pravci određeni bilo kojom točkom na kružnici (pa tako i onima u blizini zadane točke) i normalama u tim točkama sijeku upravo u centru kružnice (to je svojstvo kružnice), što je bilo i očekivano. Ako krivulja nije kružnica, radijus pripadne kružnice mijenjat će se iz točke u točku krivulje – bit će to manji što je u promatranoj točki zakrivljenost krivulje veća.

Sada bi trebalo biti jasno zašto krivulja sa slike 7.1d nema C^2 kontinuitet: centar pripadne kružnice u točki $0+\epsilon$ nalazi se iznad te točke; centar pripadne kružnice u točki $0-\epsilon$ nalazi se ispod te točke – krivulja skokovito mijenja zakrivljenost i u zakrivljenosti ima prekid.

7.2 Krivulje zadane parametarskim oblikom

U prethodnom potpoglavlju opisani su načini zapisivanja funkcija. U ovom potpoglavlju posvetit ćemo se najzanimljivijem obliku – parametarskom. Ovaj oblik specifičan je po tome što uvodi dodatne parametre zahvaljujući kojima omogućava opisivanje i "pravih" funkcija (u matematičkom smislu) i funkcija s višestrukim vrijednostima. Ideja je da se sve koordinate izraze kao funkcija tih novih parametara. Radimo li s jednim parametrom, njega ćemo tipično označavati oznakom t. U tom se slučaju ovisnost koordinata o parametru t može izraziti funkcijski kao:

$$x = f(t), \qquad y = g(t).$$

Funkcije f i g mogu biti različitih oblika. Neke od oblika upoznati ćemo u sljedećem potpoglavlju.

Parametarski oblik za svaku vrijednost parametra određuje po jednu točku krivulje. Zamislimo li da parametar t predstavlja vrijeme a krivulja putanju sićušne čestice, tada parametarski oblik za svaki trenutak određuje položaj čestice. Pustimo li vrijeme da teče kontinuirano, tada će se čestica gibati po zadanoj krivulji. Dakako, pri tome se moramo osloboditi fizikalnih zakonitosti koje idu uz ovu sliku; naime, kako će izgledati ta putanja ovisi jedino o funkcijama koje određuju ovisnosti koordinata o parametru. Tako je moguće da se čestica u jednom trenutku giba ulijevo, pa već u sljedećem zakrene za 90° i nastavi gibanje, ili pak prekine gibanje na tom mjestu i nastavi ga na nekom sasvim drugom.

Mijenjamo li parametar t po svim dozvoljenim vrijednostima (za koje su funkcije definirane), dobit ćemo cijelu krivulju. No mijenjamo li taj parametar unutar manjeg intervala $[t_0, t_1]$, čestica će opisati samo segment krivulje.

7.2.1 Uporaba parametarskog oblika

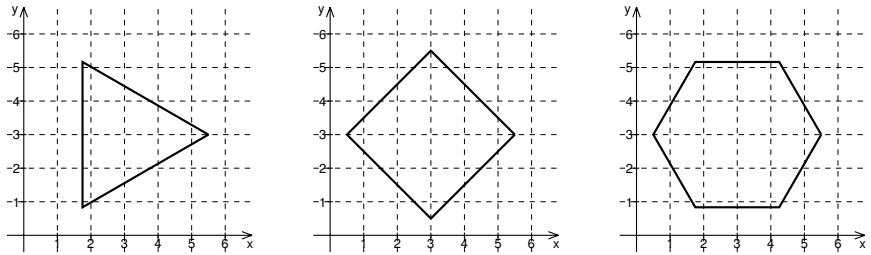
Krivulja je po definiciji skup točaka – beskonačan skup točaka. Parametarski oblik nam ovisno o parametru t daje za svaki t po jednu točku krivulje. Ako želimo nacrtati cijelu krivulju (dakle pokupiti sve točke), posao nećemo obaviti nikada. Krivulje obično prikazujemo na zaslonu, koji je rasterska jedinica. Zaslon se sastoji od niza elemenata koji mogu biti ili upaljeni ili ugašeni. Elementi zaslona adresiraju se cjelobrojnom adresom. Proizlazi da zaslon ima konačno mnogo elemenata i krivulja će na zaslonu biti aproksimirana konačnim brojem elemenata. No kako ćemo odrediti te elemente? Jedan od načina je određivanje konačnog broja točaka krivulje, te njihovo povezivanje linijskim segmentima. Za primjer ćemo uzeti prikaz kružnice. Kružnica se u parametarskom obliku može napisati:

$$x = R \cdot \cos(2\pi t) + centerX \text{ te}$$

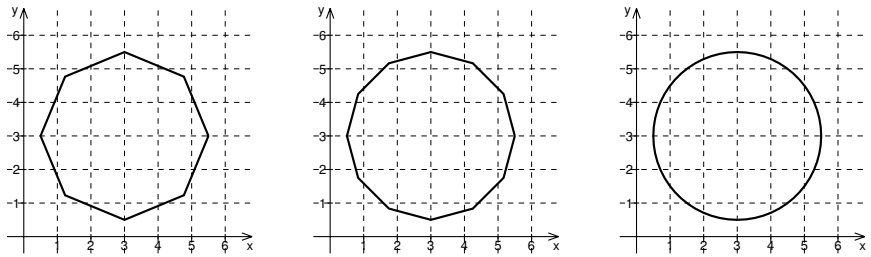
$$y = R \cdot \sin(2\pi t) + centerY.$$

Sve točke kružnice pokupit ćemo ako parametar t mijenjamo po intervalu $[0, 1\rangle$. Središte kružnice nalazi se u točki $(CenterX, CenterY)$ a radijus kružnice je R. Prema navedenom receptu, da bismo nacrtali krivulju (kružnicu), odredit ćemo $Divs$ točaka kružnice te susjedne točke spojiti linijama. Postavlja se pitanje koliko točaka treba odrediti? Pokušajmo to utvrditi eksperimentom. Na slici 7.2 prikazani su rezultati za različite vrijednosti broja točaka.

Iz slika je jasno vidljivo kada je prikaz bolji; što više točaka odredimo računski, prikaz na zaslonu je uvjerljiviji. Što nas opet vodi na zaključak da treba uzeti beskonačno finu podjelu. Nasreću, to i nije istina. Neka je kružnica radijusa 100 elemenata. I pretpostavimo da smo izračunali upravo onoliko točaka koliko je potrebno da bi svaka izračunata točka imala i susjednu koja je isto izračunata (drugim riječima, linije kojima spajamo te točke dugačke su točno jedan element – svaka spojnica degenerirala je u točku). U tom slučaju finijom podjelom nećemo ništa postići osim da na zaslonu više puta osvijetlimo iste točke. Koliko smo točaka osvijetlili? Pa gruba računica kaže da smo osvijetlili cijeli opseg kružnice, dakle $2R\pi = 628$ elemenata. U praksi ćemo, međutim, ipak ići na malo više točaka. Evo obrazloženja. Mijenjamo li parametar t od 0 do 0.25, opisat ćemo prvu četvrtinu kružnice. Pogledajmo kako naše funkcije raspodjeljuju točke u toj četvrtini. Prvih 45° prolazimo s t između 0 i 0.125. Drugih 45° prolazimo s t između 0.125 i 0.25. Oba ova segmenta imaju isti broj izračunatih točaka. Da smo umjesto kružnice crtali elipsu dosta izduženu po x-osi, tada bi u prvih



(a) divs = 3 (b) divs = 4 (c) divs = 6



(d) divs = 8 (e) divs = 12 (f) divs = 60

Slika 7.2: Utjecaj broja točaka na prikaz kružnice.

45° segment bio puno duži od segmenta u drugih 45°, a oba bi dobila isti broj točaka. Ovo vodi na zaključak da funkcije, s iznimkom posebnih slučajeva, već inherentno neravnomjerno raspoređuju točke te se može dogoditi da na jednom segmentu imamo puno izračunatih točaka a na drugom malo. Na segmentu na kojem ima malo točaka jasnije bi se vidjelo spajanje linijama a to nije poželjno. Stoga je potrebno napraviti već u startu gušću podjelu da se ovi efekti učine zanemarivim.

7.2.2 Primjer crtanja kružnice

U nastavku ćemo dati implementaciju funkcije koja crta kružnicu na zaslonu, oslanjajući se na parametarski zapis krivulje. Funkciji se predaju koordinate centra, radijus kružnice te željeni broj točaka koje će se računati.

```
typedef struct {
        double x;
        double y;
} Point2D;

void circle(Point2D center, double radius, int divs) {
```

```
    double t;
    Point2D p;

    if(divs<2) return;

    glBegin(GL_LINE_STRIP);
    for(int n=0; n<=divs; n++) {
        t = 2.0*PI/divs*n;
        p.x = radius*cos(t)+center.x;
        p.y = radius*sin(t)+center.y;
        glVertex2f(p.x, p.y);
    }
    glEnd();
}
```

Ovdje primijenjen postupak za izračunavanje točaka kružnice temelji se direktno na jednadžbi kružnice (parametarskom obliku), te kao takav ne spada u grupu optimalnih. Za iscrtavanje kružnice postoje i brži postupci, a jedan od njih je i Bresenhamov postupak za kružnice.

7.2.3 Primjer crtanja elipse

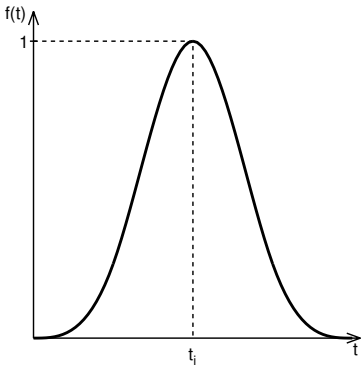
Prethodni primjer može se vrlo jednostavno modificirati tako da umjesto kružnice crta elipsu. Elipsa za razliku od kružnice ima dva radijusa – jedan određuje širinu elipse a drugi visinu. Funkcija koju ćemo napisati prihvatit će koordinate pravokutnika kojemu treba upisati elipsu.

```
void ellipse(Point2D p1, Point2D p2, int divs) {
    double t, rx, ry;
    Point2D p, center;

    if(divs<2) return;
    if(p1.x>p2.x) {
        double tmp = p1.x;
        p1.x = p2.x;
        p2.x = p1.x;
    }
    if(p1.y>p2.y) {
        double tmp = p1.y;
        p1.y = p2.y;
        p2.y = p1.y;
    }

    rx = (p2.x-p1.x)/2.0;
    ry = (p2.y-p1.y)/2.0;
    center.x = p1.x + rx;
    center.y = p1.y + ry;

    glBegin(GL_LINE_STRIP);
    for(int n=0; n<=divs; n++) {
```

```
        t = 2.0*PI/divs*n;
        p.x = rx*cos(t)+center.x;
        p.y = ry*sin(t)+center.y;
        glVertex2f(p.x, p.y);
    }
    glEnd();
}
```

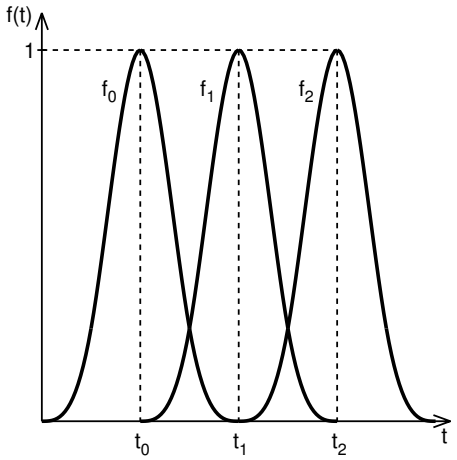
7.2.4 Konstrukcija krivulje s obzirom na zadane točke

Jedan od češćih problema koji se veže uz krivulje jest sljedeći: "zadano je $n+1$ točka; potrebno je povući krivulju s obzirom na te točke". Što točno znači "s obzirom", ovisi o potrebama; to može značiti "kroz točke" pa tada govorimo o *interpolaciji*, ili može značiti "tako da krivulja prolazi negdje u blizini tih točaka" pa govorimo o *aproksimaciji*. Također, mogu se postavljati različiti zahtjevi i na oblik krivulje. Primjerice, možemo tražiti da krivulja bude glatka i slično.



Slika 7.3: Primjer težinske funkcije zvonolikog oblika

Jedno od mogućih rješenja je uporaba *težinskih funkcija*. Ideja je sljedeća. Želimo konstruirati krivulju čiji se početak dobije za vrijednost parametra $t = t_s$ a kraj za $t = t_e$. Pri tome krivulja prolazi kraj točke T_i za vrijednost parametra $t = t_i$. Iskoristiti ćemo težinske funkcije zvonolikog oblika koje imaju maksimum za $t = t_i$, kao na slici 7.3. Težinskih funkcija bit će onoliko koliko je zadano točaka. Svaka točka obliku krivulje doprinositi će svojim koordinatama pomnoženim s težinskom funkcijom. Jednadžbu krivulje tada ćemo zapisati kao sumu svake točke pomnožene težinskom funkcijom. Težinske funkcije označavat ćemo oznakom f_i, pri čemu indeks i govori kojoj točki pripada ta težinska funkcija. Tako jednadžbu krivulje možemo zapisati:



Slika 7.4: Težinske funkcije uz tri točke krivulje

$$T_K = \sum_{i=0}^{n} f_i(t) \cdot T_i$$

gdje je T_K točka krivulje a T_i i-ta zadana točka (kako je zadano $n+1$ točaka, indeks ide od 0 do n). Uzmimo kao primjer da su zadane tri točke. Težinske funkcije mogle bi izgledati kao na slici 7.4. Ako krivulju crtamo od $t = t_0$ do $t = t_2$, što možemo reći o krivulji gledajući ove težinske funkcije?

Za $t = t_0$ je $f_0 = 1$, $f_1 = 0$, $f_2 = 0$, pa je:

$$T_K(t = t_0) = \sum_{i=0}^{2} f_i(t_0) \cdot T_i = f_0(t_0) \cdot T_0 + f_1(t_0) \cdot T_1 + f_2(t_0) \cdot T_2 = 1 \cdot T_0 + 0 \cdot T_1 + 0 \cdot T_2 = T_0.$$

Za $t = t_1$ je $f_0 = 0$, $f_1 = 1$, $f_2 = 0$, pa je:

$$T_K(t = t_1) = \sum_{i=0}^{2} f_i(t_1) \cdot T_i = f_0(t_1) \cdot T_0 + f_1(t_1) \cdot T_1 + f_2(t_1) \cdot T_2 = 0 \cdot T_0 + 1 \cdot T_1 + 0 \cdot T_2 = T_1.$$

Za $t = t_2$ je $f_0 = 0$, $f_1 = 0$, $f_2 = 1$, pa je:

$$T_K(t = t_2) = \sum_{i=0}^{2} f_i(t_2) \cdot T_i = f_0(t_2) \cdot T_0 + f_1(t_2) \cdot T_1 + f_2(t_2) \cdot T_2 = 0 \cdot T_0 + 0 \cdot T_1 + 1 \cdot T_2 = T_2.$$

Krivulja dakle prolazi kroz sve zadane točke, a međutočke se aproksimiraju. Možemo li pogledom na sliku 7.4. reći kako krivulja ovisi o promjeni pojedine točke? Na slici je jasno vidljivo da istovremeno na položaj točke djeluju najviše dvije težinske funkcije dok su ostale nula; naime, na intervalu $[t_0, t_1]$ djeluju f_0 i f_1 dok je f_2 jednaka nuli. Na intervalu $[t_1, t_2]$ djeluju f_1 i f_2 dok je f_0 jednaka nuli. To znači da oblik krivulje na intervalu $[t_0, t_1]$ određuju isključivo točke T_0 i T_1 dok točku T_2 možemo pomicati kamo god želimo bez da utječemo na ovaj segment krivulje. Oblik krivulje na intervalu $[t_1, t_2]$ određuju pak točke T_1 i T_2 dok točku T_0 možemo pomicati kamo god želimo bez da utječemo na ovaj segment krivulje. Ovakvo svojstvo krivulje kada promjena položaja jedne točke utječe na promjenu oblika krivulju isključivo u okolini te točke naziva se *svojstvo lokalnog nadzora*. Neke krivulje nemaju ovo svojstvo, pa pomaknemo li jednu točku krivulje, mijenja se cijela krivulja. Svojstvo lokalnog nadzora poželjno je svojstvo.

Pogledajmo još malo sliku 7.4. Na slici su sve težinske funkcije jednake. Takve funkcije nazivamo *uniformnima*. Kod težinskih funkcija možemo mijenjati dva parametra: visinu težinske funkcije te širinu težinske funkcije. Pogledajmo kakvog to utjecaja ima na oblik krivulje.

Ako mijenjamo visinu težinske funkcije, fizikalno to možemo protumačiti kao promjenu privlačenja dotične točke i krivulje. Što je težinska funkcija viša, to će odgovarajuća točka više utjecati na oblik krivulje; što je težinska funkcija niža, to će točka manje mijenjati oblik krivulje. Ovo nam omogućava da osim samih točaka odredimo i njihove "težine" te na temelju toga mijenjamo oblik krivulje.

Promjena širine težinske funkcije može se tumačiti kao promjena dosega privlačne sile između točke i krivulje. Što je težinska funkcija šira, to će odgovarajuća točka imati utjecaj na veći komad krivulje; što je težinska funkcija uža, to će točka utjecati na krivulju na manjem segmentu. Na slici 7.4 odabrane su težinske funkcije tako da svaka točka utječe na krivulju na segmentu od prethodnog susjeda točke pa do sljedećeg susjeda točke. Ovime smo postigli svojstvo lokalnog nadzora jer utjecaj jedne točke nije dopirao dalje od njezinih susjeda. Ako bismo širine težinskih funkcija udvostručili, tada bi utjecaj točke T_0 dopirao sve do točke T_2; a kako krivulju crtamo upravo od točke T_0 do točke T_2, tada bi točka T_0 utjecala na cijelu krivulju. Time bismo izgubili svojstvo lokalnog nadzora.

Ako se dozvoljava da se pojedine težinske funkcije razlikuju po širini i po visini, tada za takve težinske funkcije kažemo da su *neuniformne*.

7.2.5 Ponavljanje

U ovom potpoglavlju upoznali smo se sa značajem parametarskog oblika, načinima crtanja funkcija zadanih na taj način, te idejom konstrukcije krivulje s obzirom na zadane točke, što smo ilustrirali uporabom težinskih funkcija. Kao primjer smo pokazali zvonolike težinske funkcije. Primjer nam je poslužio i za

upoznavanje s utjecajima pojedinih parametara samih težinskih funkcija na oblik krivulje. Situacija u praksi, što se tiče ovog dijela je vrlo šarolika. Ne samo da se koriste svakakvi oblici težinskih funkcija, nego se i izvode novi, ovisno o zahtjevima koji se postavljaju na krivulje. Kao primjer krivulja koje nude dosta slobode a koriste zvonolike funkcije navest ćemo *NURBS* (neuniformni racionalni b-spline). Primjer često korištenih krivulja kod kojih svaka od težinskih funkcija nije nužno zvonolika na promatranom intervalu su Bézierove krivulje kod kojih prva i posljenja težinska funkcija nisu zvonolike.

U nastavku ćemo se upoznati s nekim od čestih oblika krivulja.

7.3 Bézierove krivulje

U računalnoj grafici jedan od čestih zadataka je provlačenje glatke krivulje između zadanog niza točaka (aproksimacija krivulje). Jedno od rješenja problema nude nam Bézierove krivulje. Do krivulja su nezavisno došli 1962. Bézier koji je radio za tvrtku Renault, te 1959. De Casteljau koji je radio za tvrtku Citroen. Da je riječ o istim krivuljama, 1970. godine dokazao je Robert Forest. Za matematički opis Bézierovih krivulja postoje dvije metode: prva koja se temelji na Bézierovim težinskim funkcijama te druga koje se temelji na Bernsteinovim težinskim funkcijama (a dobiva se kao rezultat primjene De Casteljauovog algoritma). Za praktičnu primjenu u računalima Bernsteinove težinske funkcije su daleko pogodnije. Iz povijesnih razloga te činjenice da je prva navedena metoda upravo metoda koja nosi ime po izumitelju ove porodice krivulja, u nastavku ćemo dati kratak osvrt na obje metode. Osnovni oblik Bézierove krivulje računat će se izravno na temelju unaprijed poznatih vrhova kontrolnog poligona; terminološki, takav oblik krivulje zvat ćemo *aproksimacijskom Bézierovom krivuljom*. Ukoliko su umjesto vrhova kontrolnog poligona poznati određeni uvjeti na Bézierovu krivulju (poput niza točaka kroz koje krivulja mora proći, vrijednosti tangenata u pojedinim točaka i slično), takav ćemo oblik krivulje nazivati *interpolacijskom Bézierovom krivuljom*. Treba napomenuti da se u svim slučajevima radi o istoj vrsti krivulje: Bézierovoj krivulji, a pojmove aproksimacijska odnosno interpolacijska koristit ćemo kao oznaku načina na koji krivulju određujemo.

U nastavku ćemo pokazati da aproksimacijska Bézierova krivulja prolazi kroz početnu i završnu točku, dok kroz ostale točke u općem slučaju ne prolazi. Interpolacijska Bézierova krivulja ovisno o uvjetima pod kojim je konstruirana može prolaziti kroz sve zadane točke.

7.3.1 Aproksimacijska Bézierova krivulja

Prilikom rada s Bézierovim krivuljama te njihovog matematičkog tretmana koristit ćemo sljedeće oznake.

- Kontrolne točke - vrhovi *kontrolnog poligona* bit će označeni oznakom T_i, gdje je i indeks kontrolne točke. Bézierova krivulja bit će zadana kontrolnim točkama T_0, T_1, ..., T_n. Pri tome n označava stupanj krivulje. Tako će krivulja trećeg stupnja ($n = 3$) biti zadana s četiri točke: T_0, T_1, T_2 i T_3. Točkama T_0, T_1, ..., T_n zapravo je (u matematičkom smislu) određena poligonalna linija. No u ovom kontekstu, uvriježeno je koristiti naziv "kontrolni poligon".

- Oznakom $\vec{r}_i$ označavat će se radij-vektori točaka vrhova poligona. Radij vektor je vektor koji spaja ishodište i zadanu točku T_i. Zbog toga su mu sve komponente jednake kao i kod same točke T_i. Dakle, radij-vektor koji spaja točku (1 2) je (1 2).

- Oznakom $\vec{a}_i$ označavat će se vektori između točaka T_{i-1} i T_i; kako točka T_0 nema prethodnika, vektor $\vec{a}_0$ definirat ćemo da je jednak radij-vektoru te točke, tj $\vec{a}_0 = \vec{r}_0$. Općenito se može zapisati:

$$\vec{a}_i = \begin{cases} T_i - T_{i-1}, & i > 0 \\ T_0 & i = 0 \end{cases} .$$

- Težinske funkcije bit će označavane oznakom $\psi_{i,n}$, pri čemu će kod Bézierovih težinskih funkcija umjesto ψ stajati f, a kod Bernsteinovih težinskih funkcija b. Pri tome i označava indeks funkcije i taj parametar će se mijenjati između 0 i n, dok n označava stupanj krivulje.

Bézierove težinske funkcije

Do ovih se funkcija dolazi metodom gibanja vrha sastavljenog otvorenog poligona, kao što je već spomenuto u uvodu. Sastavljeni otvoreni poligon označava poligon koji se dobije kada se svi vektori $\vec{a}_i$ izvornog poligona pomnože s određenim težinskim funkcijama i zatim zbroje. Kod nas će, dakako, vektori biti pomnoženi Bézierovim težinskim funkcijama. Slijedi da se može pisati:

$$\vec{p}(t) = \sum_{i=0}^{n} \vec{a}_i f_{i,n}(t)$$

pri čemu je $\vec{p}(t)$ radij-vektor točke koja pripada krivulji (vrh sastavljenog otvorenog poligona), a t je parametar kojim određujemo sve točke krivulje. Za $t = 0$ dobiva se početna točka krivulje a za $t = 1$ dobiva se završna točka krivulje. Sve ostale točke dobivaju se za $t \in (0, 1)$.

Do analitičkih izraza za funkcije $f_{i,n}$ dolazi se iz sljedećih zahtjeva.

1. Krivulja za $t = 0$ prolazi kroz prvu zadanu točku: $\vec{p}(0) = \vec{a}_0$, odakle slijedi da mora vrijediti $f_{0,n}(0) = 1$ te $f_{i,n}(0) = 0, i = 1, 2, ..., n$.

2. Krivulja za $t = 1$ prolazi kroz zadnju zadanu točku: $\vec{p}(1) = \sum_{i=0}^{n} \vec{a}_i$, odakle slijedi da mora vrijediti $f_{i,n}(1) = 1, i = 0, 1, ..., n$.

3. U početnoj točki nagib krivulje jednak je nagibu vektora $\vec{a}_1$, što se može osigurati ako se zahtijeva $f'_{1,n}(0) = 1$ te $f'_{i,n}(0) = 0, i \neq 1$ (ovo ćemo pokazati u nastavku).

4. U završnoj točki nagib krivulje jednak je nagibu vektora $\vec{a}_n$, što se može osigurati ako se zahtijeva $f'_{n,n}(1) = 1$ te $f'_{i,n}(1) = 0, i \neq n$.

5. Postavlja se zahtjev na druge derivacije u početnoj točki: $f''_{1,n}(0) \neq 0$, $f''_{2,n}(0) \neq 0$ te $f''_{i,n}(0) = 0, i \neq 1, 2$.

6. Postavlja se zahtjev na druge derivacije u završnoj točki: $f''_{n-1,n}(1) \neq 0$, $f''_{n,n}(1) \neq 0$ te $f''_{i,n}(1) = 0, i \neq n-1, n$.

7. Zahtjeva se simetričnost, odnosno traži se da redoslijed točaka ne utječe na izgled krivulje (zamjena početne i krajnje točke nema utjecaja), što vodi na zahtjev $f_{i,n}(t) = 1 - f_{n-i+1,n}(1-t)$, $i = 1, 2, ..., n$.

Težinske funkcije koje proizlaze iz ovih zahtjeva nazivaju se Bézierove težinske funkcije i zadane su izrazom:

$$f_{i,n}(t) = \frac{(-t)^i}{(i-1)!} \frac{d^{(i-1)}\Phi_n(t)}{dt^{(i-1)}}, \quad i = 1, 2, ..., n \tag{7.1}$$

$$f_{0,n}(t) = 1 \tag{7.2}$$

pri čemu je funkcija $\Phi_n(t)$ zadana izrazom:

$$\Phi_n(t) = \frac{1 - (1-t)^n}{-t}. \tag{7.3}$$

Zbog potrebe za deriviranjem i vrlo složenih izraza vidljivo je da ovakav zapis nije baš prikladan za uporabu u računalima. No ovi se izrazi mogu napisati u, za računala, puno prihvatljivijem obliku: kao rekurzivne funkcije.

$$f_{i,n}(t) = (1-t) f_{i,n-1}(t) + t \cdot f_{i-1,n-1}(t), \tag{7.4}$$

$$f_{0,0}(t) = 1, \quad f_{-1,k}(t) = 1, \quad f_{k+1,k}(t) = 0. \tag{7.5}$$

Pokažimo još kako smo došli do izraza iz točke 3 u zahtjevima na Bézierove težinske funkcije. Vektor $\vec{a}_1$ je razlika radij vektora $\vec{a}_1 = \vec{r}_1 - \vec{r}_0$, te svojim komponentama određuje nagib $\text{tg}\,(\delta) = \frac{a_{1,y}}{a_{1,x}}$. Krivulja je zadana u parametarskom obliku sumom $\vec{p}(t) = \sum_{i=0}^{n} \vec{a}_i f_{i,n}(t)$. Ova jednadžba vrijedi i za svaku komponentu zasebno. Nagib krivulje određen je derivacijom $\frac{dy}{dx}$, što se iz dane jednadžbe ne možemo dobiti direktno već malim trikom. Pomnožimo traženu derivaciju s prikladno napisanom jedinicom:

$$\begin{aligned}\frac{dy}{dx} &= \frac{dy}{dx}\cdot\frac{dt}{dt} = \frac{\frac{dy}{dt}}{\frac{dx}{dt}} \\ &= \frac{\frac{d}{dt}\left(\sum_{i=0}^{n} a_{i,y} f_{i,n}(t)\right)}{\frac{d}{dt}\left(\sum_{i=0}^{n} a_{i,x} f_{i,n}(t)\right)} \\ &= \frac{a_{0,y} f_{0,n}^{'}(t) + a_{1,y} f_{1,n}^{'}(t) + \cdots + a_{n,y} f_{n,n}^{'}(t)}{a_{0,x} f_{0,n}^{'}(t) + a_{1,x} f_{1,n}^{'}(t) + \cdots + a_{n,x} f_{n,n}^{'}(t)}.\end{aligned}$$

Zahtjev je da u početnoj točki ($t = 0$) nagib bude $\frac{a_{1,y}}{a_{1,x}}$, odakle odmah slijedi:

$$\frac{dy}{dx}(t=0) = \frac{a_{0,y} f_{0,n}^{'}(0) + a_{1,y} f_{1,n}^{'}(0) + \cdots + a_{n,y} f_{n,n}^{'}(0)}{a_{0,x} f_{0,n}^{'}(0) + a_{1,x} f_{1,n}^{'}(0) + \cdots + a_{n,x} f_{n,n}^{'}(0)} = \frac{a_{1,y}}{a_{1,x}}$$

što je moguće postići, npr., kada su derivacije svih težinskih funkcija u toj točki jednake nuli, osim derivacije funkcije $f_{1,n}$ jer ona množi tražene komponente vektora $\vec{a}_1$.

Slično se dobije i za zahtjev 4, gdje se traži jednakost nagiba u završnoj točki krivulje ($t = 1$), i vektora $\vec{a}_n$, ako se u gornju formulu uvrsti $t = 1$ i $\mathrm{tg}\,(\delta) = \frac{a_{n,y}}{a_{n,x}}$. Tada se izjednačavanjem dobije da derivacije svih težinskih funkcija osim $f_{n,n}$ moraju biti nula, a derivacija od $f_{n,n}$ mora biti 1.

Bernsteinove težinske funkcije

Do Bernsteinovih težinskih funkcija dolazi se De Casteljauovim algoritmom čija je ideja iznesena u nastavku. Taj je algoritam izuzetno važan za računalnu grafiku jer daje brz i numerički stabilan način računanja Bézierovih krivulja. Potrebno je odrediti točku krivulje za fiksni t. Npr. za $t = \frac{1}{4}$. Sve stranice izvornog poligona podijele se novom točkom u omjeru $t : (1 - t)$. Novodobivene točke spoje se spojnicama, i zatim se na spojnicama određuju nove točke opet zadane parametrom t. Postupak se ponavlja sve dok ne ostane samo jedna spojnica i na njoj ovako određena točka. Ta točka ujedno je i točka krivulje.

Postupak ćemo najbolje ilustrirati na primjeru. Neka su zadane točke T_0, T_1, T_2 i T_3 prema slici 7.5. Na slici 7.5(a) točke su spojene poligonalnom linijom. Odredimo točku za $t = \frac{1}{4}$. Točka $b_{1,0}$ označava točku na četvrtini spojnice između točaka T_0 i T_1. Točka $b_{1,1}$ je na četvrtini spojnice između točaka T_1 i T_2. Točka $b_{1,2}$ je na četvrtini spojnice između točaka T_2 i T_3. Točke $b_{1,0}$, $b_{1,1}$ i $b_{1,2}$ spojene su uporabom dviju novih spojnica. Na slici 7.5(b) te su spojnice podijeljene tako da se točka $b_{2,0}$ nalazi na četvrtini puta između $b_{1,0}$ i $b_{1,1}$, a točka $b_{2,1}$ nalazi se na četvrtini puta između $b_{1,1}$ i $b_{1,2}$. Ove dvije točke opet su spojene spojnicom.

Na slici 7.5(c) ta je spojnice podijeljena tako da se točka $b_{3,0}$ nalazi na četvrtini puta između $b_{2,0}$ i $b_{2,1}$. Time je postupak gotov jer više nemamo novih spojnica i točka $b_{3,0}$ predstavlja točku Bézierove krivulje za $t = \frac{1}{4}$.

Postupak možemo opisati na sljedeći način. Krenuli smo od točaka poligona:

$$\begin{aligned} b_{1,0} &= (T_1 - T_0) \cdot t + T_0, \\ b_{1,1} &= (T_2 - T_1) \cdot t + T_1, \\ b_{1,2} &= (T_3 - T_2) \cdot t + T_2. \end{aligned}$$

Zatim smo spojnice dijelili:

$$\begin{aligned} b_{2,0} &= (b_{1,1} - b_{1,0}) \cdot t + b_{1,0}, \\ b_{2,1} &= (b_{1,2} - b_{1,1}) \cdot t + b_{1,1}. \end{aligned}$$

I konačno smo i te spojnice podijelili:

$$b_{3,0} = (b_{2,1} - b_{2,0}) \cdot t + b_{2,0}.$$

Ako sada $b_{3,0}$ izrazimo samo preko točaka, dobit ćemo:

$$b_{3,0} = (1-t)^3 \cdot T_0 + 3t(1-t)^2 \cdot T_1 + 3t^2(1-t) \cdot T_2 + t^3 \cdot T_3$$

ili da to napišemo korektno preko radij-vektora točaka:

$$b_{3,0} = (1-t)^3 \cdot \vec{r}_0 + 3t(1-t)^2 \cdot \vec{r}_1 + 3t^2(1-t) \cdot \vec{r}_2 + t^3 \cdot \vec{r}_3.$$

Ova točka pripada krivulji, pa konačno možemo pisati:

$$\vec{p}(t) = (1-t)^3 \cdot \vec{r}_0 + 3t(1-t)^2 \cdot \vec{r}_1 + 3t^2(1-t) \cdot \vec{r}_2 + t^3 \cdot \vec{r}_3. \tag{7.6}$$

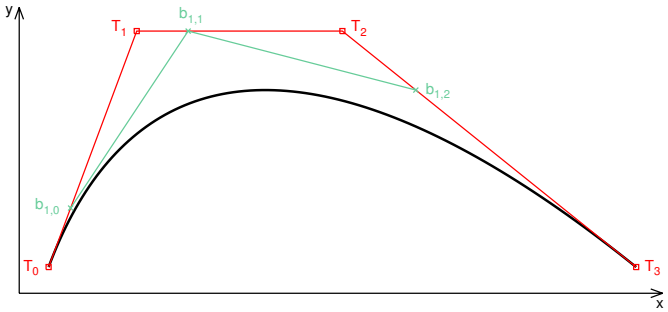
Dobivena je formula koja opisuje sve točke Bézierove krivulje zadane preko četiri točke, dakle krivulje trećeg reda. Pogleda li se formula malo bolje, vidi se da faktori ispred radij-vektora neobično podsjećaju na binomnu formulu. Da ima istine u tome, govori nam i sljedeći zapis ovdje prikazanih težinskih funkcija poznat kao Bernsteinove težinske funkcije:

$$\vec{p}(t) = \sum_{i=0}^{n} \vec{r}_i \cdot b_{i,n}(t) \tag{7.7}$$

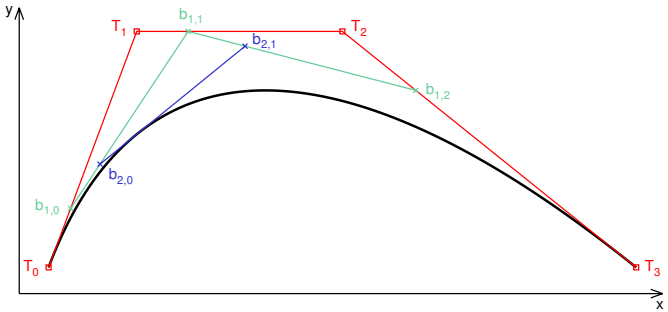
pri čemu su težinske funkcije dane relacijom:

$$b_{i,n}(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i} = \frac{n!}{i!(n-i)!} \cdot t^i \cdot (1-t)^{n-i}. \tag{7.8}$$

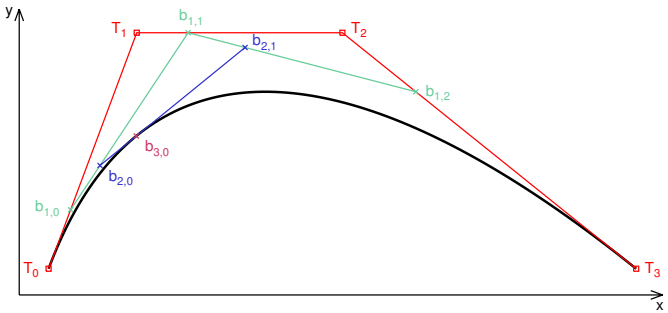
Ovo je najjednostavniji oblik za uporabu u računalima, i mi ćemo ga koristiti u nastavku. Pojava faktorijela također ne komplicira stvari.



(a) 1. korak rekurzije



(b) 2. korak rekurzije



(c) 3. korak rekurzije

Slika 7.5: Konstrukcija Bézierove krivulje.

Vratimo se sada na primjer ručne konstrukcije točaka Bézierove krivulje koji je rezultirao izrazom (7.6). Težinske funkcije uz pojedine radij-vektore upravo su Bernsteinove težinske funkcije:

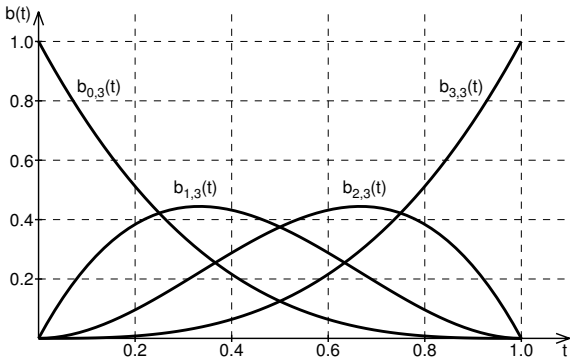
$$b_{0,3}(t) = (1-t)^3,$$

$$b_{1,3}(t) = 3(1-t)^2 t,$$

$$b_{2,3}(t) = 3(1-t)t^2 \text{ te}$$

$$b_{3,3}(t) = t^3.$$

Ove težinske funkcije prikazane su na slici 7.6.



Slika 7.6: Bernsteinove težinske funkcije za Bézierovu krivulju trećeg stupnja

Svojstva Bernsteinovih težinskih funkcija

Kako ćemo u nastavku dosta koristiti Bernsteinove težinske funkcije, navedimo nekoliko njihovih važnijih svojstava.

- Nenegativne su na intervalu $[0, 1]$.
- Vrijedi $\sum_{i=0}^{n} b_{i,n}(t) = 1$. Izvod ovog svojstva dan je još u potpoglavlju 3.4.
- Vrijedi simetričnost $b_{i,n}(1-t) = b_{n-i,n}(t)$.
- Vrijedi rekurzivna relacija: $b_{i,n}(t) = t \cdot b_{i-1,n-1}(t) + (1-t) \cdot b_{i,n-1}(t)$ uz $b_{0,0}(t) = 1$.
- Derivaciju je moguće zapisati kao kombinaciju dviju funkcija nižeg stupnja: $b'_{i,n}(t) = n \cdot (b_{i-1,n-1}(t) - b_{i,n-1}(t))$.

- Bernsteinova težinska funkcija $b_{i,n}(t)$ za $n > 0$ ima točno jedan maksimum koji poprima za $t = \frac{i}{n}$. Iznos tog maksimuma je 1 kod $b_{0,n}(t)$ i poprima se upravo za $t = 0$. Uvrštavanjem $t = \frac{i}{n}$ u izraz za $b_{i,n}(t)$ uz $i > 0$ dobiva se izraz za vrijednost maksimuma: $\binom{n}{i} i^i n^{-n} (n-i)^{n-i}$.

Matrični prikaz

Prilikom rada s krivuljama (što za potrebe crtanja, što za potrebe proračuna) često se koristi matrični prikaz. Pogledajmo kako bismo do njega došli za kubnu Bézierovu krivulju u radnom trodimenzionalnom prostoru. Općenit izraz koji definira Bézierovu krivulju koristeći Bernsteinove težinske funkcije dan je izrazom (7.7) koji za kubnu krivulju prelazi u:

$$\begin{aligned}\vec{p}(t) &= \sum_{i=0}^{3} \vec{r}_i \cdot b_{i,n}(t) \\ &= b_{0,3}(t) \cdot \vec{r}_0 + b_{1,3}(t) \cdot \vec{r}_1 + b_{2,3}(t) \cdot \vec{r}_2 + b_{3,3}(t) \cdot \vec{r}_3.\end{aligned}$$

Posljednji redak možemo prikazati matrično:

$$\vec{p}(t) = \begin{bmatrix} b_{0,3}(t) & b_{1,3}(t) & b_{2,3}(t) & b_{3,3}(t) \end{bmatrix} \cdot \begin{bmatrix} \vec{r}_0 \\ \vec{r}_1 \\ \vec{r}_2 \\ \vec{r}_3 \end{bmatrix}$$

gdje su $\vec{r}_i$ vektor retci (desna matrica je u ovom slučaju matrica 4×3). Supstitucijom težinskih funkcija $b_{i,3}(t)$ slijedi:

$$\vec{p}(t) = (1-t)^3 \cdot \vec{r}_0 + 3 \cdot (1-t)^2 \cdot t \cdot \vec{r}_1 + 3 \cdot (1-t) \cdot t^2 \cdot \vec{r}_2 + t^3 \cdot \vec{r}_3.$$

Posljednji redak matrično možemo prikazati kako slijedi:

$$\vec{p}(t) = \begin{bmatrix} (1-t)^3 & 3 \cdot (1-t)^2 \cdot t & 3 \cdot (1-t) \cdot t^2 & t^3 \end{bmatrix} \cdot \begin{bmatrix} \vec{r}_0 \\ \vec{r}_1 \\ \vec{r}_2 \\ \vec{r}_3 \end{bmatrix}$$

Raspišimo prikazane polinome.

$$\begin{aligned} (1-t)^3 &= -1 \cdot t^3 + 3 \cdot t^2 - 3 \cdot t + 1 \\ 3 \cdot (1-t)^2 \cdot t &= 3 \cdot t^3 - 6 \cdot t^2 + 3 \cdot t + 0 \\ 3 \cdot (1-t) \cdot t^2 &= -3 \cdot t^3 + 3 \cdot t^2 + 0 \cdot t + 0 \\ t^3 &= 1 \cdot t^3 + 0 \cdot t^2 + 0 \cdot t + 0 \end{aligned}$$

Jednoretčanu matricu težinskih funkcija (odnosno prikazanih polinoma) možemo zapisati kao produkt jednoretčane matrice potencija parametra te kvadratne matrice koeficijenata koje smo dobili raspisivanjem polinoma:

$$\left[\begin{array}{cccc} (1-t)^3 & 3\cdot(1-t)^2\cdot t & 3\cdot(1-t)\cdot t^2 & t^3 \end{array}\right] = \left[\begin{array}{cccc} t^3 & t^2 & t & 1 \end{array}\right] \cdot \left[\begin{array}{cccc} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array}\right]$$

pri čemu su koeficijenti svakog od polinoma popisani u stupcima. Uvjerite se da umnožak s desne strane doista daje jednoretčanu matricu težinskih funkcija. Ovaj izraz uvrstit ćemo u prethodni matrični oblik pri čemu ćemo po komponentama raspisati i radij vektore točaka kontrolnog poligona kako bismo dobili konačni matrični izraz:

$$\vec{p}(t) = \left[\begin{array}{cccc} t^3 & t^2 & t & 1 \end{array}\right] \cdot \left[\begin{array}{cccc} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array}\right] \cdot \left[\begin{array}{ccc} r_{0,x} & r_{0,y} & r_{0,z} \\ r_{1,x} & r_{1,y} & r_{1,z} \\ r_{2,x} & r_{2,y} & r_{2,z} \\ r_{3,x} & r_{3,y} & r_{3,z} \end{array}\right]. \tag{7.9}$$

Uvođenjem oznake $\mathbf{B}$ za matricu koeficijenata polinoma težinskih funkcija te $\mathbf{R}$ za matricu zadanih radij vektora možemo pisati:

$$\vec{p}(t) = \left[\begin{array}{cccc} t^3 & t^2 & t & 1 \end{array}\right] \cdot \mathbf{B} \cdot \mathbf{R} \tag{7.10}$$

što je rezultat na koji ćemo se kasnije pozvati.

Veza između Bézierovih i Bernsteinovih težinskih funkcija

U uvodu smo već rekli da se i pomoću Bézierovih i pomoću Bernsteinovih težinskih funkcija opisuje ista krivulja; to je dokazao Robert Forest. Mi ćemo u nastavku samo dati tu vezu:

$$f_{i,n}(t) = \sum_{j=i}^{n} b_{j,n}(t).$$

Direktno crtanje aproksimacijske Bézierove krivulje

Kod koji crta Bézierovu krivulju strukturom će biti sličan onome za kružnicu ili elipsu. Metoda draw_bezier kao argumente će primiti polje točaka kontrolnog poligona, broj tih točaka te parametar divs koji govori koliko ćemo gusto "uzorkovati" Bézierovu krivulju kako bismo ostatak pospajali linijskim segmentima. Točke krivulje računat ćemo uporabom Bernsteinovih težinskih funkcija.

Metoda na početku stvori pomoćno polje faktora u koje će se pohraniti vrijednost: $\binom{n}{i}$ kako se ne bi nepotrebno računala za svaki uzorak. To popunjava metoda compute_factors i to u linearnoj složenosti. Treba napomenuti da opisani algoritam nije računalno optimalan već predstavlja izravnu implementaciju izračuna točaka putem Bernsteinovih težinskih funkcija. Učinkovitije implementacije moguće je postići primjerice uporabom De Casteljauovog algoritma ali to ovdje nećemo raditi.

```
void compute_factors(int n, int *factors) {
    int i, a=1;

    for(i = 1; i <= n+1; i++) {
        factors[i-1] = a;
        a = a * (n-i+1) / i;
    }
}

void draw_bezier(Point2D *points, int points_count, int divs) {
    Point2D p;
    int n = points_count-1;
    int *factors = (int*)malloc(sizeof(int)*points_count);
    double t, b;

    compute_factors(n, factors);
    glBegin(GL_LINE_STRIP);
    for(int i = 0; i <= divs; i++) {
        t = 1.0 / divs * i;
        p.x = 0; p.y = 0;
        for(int j = 0; j <= n; j++) {
            if(j==0) {
                b = factors[j]*pow(1-t,n);
            } else if(j==n) {
                b = factors[j]*pow(t,n);
            } else {
                b = factors[j]*pow(t,j)*pow(1-t,n-j);
            }
            p.x += b * points[j].x;
            p.y += b * points[j].y;
        }
        glVertex2f(p.x, p.y);
    }
    glEnd();

    free(factors);
}
```

Linearnu složenost kojom metoda compute_factors računa vrijednost funkcije *povrh* možemo jednostavno objasniti. Neka je zadan stupanj n. Tada su faktori

redom:

$$
\begin{aligned}
&factors[0] = 1\\
&factors[1] = \tfrac{n}{1} = factors[0] \cdot \tfrac{n}{1}\\
&factors[2] = \tfrac{n\cdot(n-1)}{1\cdot 2} = factors[1] \cdot \tfrac{n-1}{2}\\
&factors[3] = \tfrac{n\cdot(n-1)\cdot(n-2)}{1\cdot 2\cdot 3} = factors[2] \cdot \tfrac{n-2}{3}\\
&\dots\\
&factors[n] = \tfrac{n\cdot(n-1)\cdot(n-2)\cdot\dots\cdot 1}{1\cdot 2\cdot 3\cdot\dots\cdot n} = factors[n-1] \cdot \tfrac{n-(n-1)}{n}
\end{aligned}
$$

pa se svaki faktor može dobiti direktno poznavanjem samo prethodnog člana.

Crtanje aproksimacijske Bézierove krivulje rekurzivnim algoritmom

Prethodno smo pokazali da se točke aproksimacijske Bézierove krivulje, uz poznate točke kontrolnog poligona, mogu računati kao umnožak triju matrica (vidi izraz (7.10)): jednoretčane matrice $\mathbf{T}$ čiji su elementi potencije parametra, kvadratne matrice $\mathbf{B}$ čiji su elementi, po stupcima, skalari koji u Bernsteinovim težinskim funkcijama dolaze uz pojedine potencije parametra, te matrice $\mathbf{R}$ koja je matrica radij-vektora vrhova kontrolnog poligona. Prethodno smo pokazali da je jedan mogući način crtanja tako zadane krivulje uzorkovanje u nizu različitih rastućih vrijednosti parametra i potom spajanje linijskim segmentima. Da bismo pokazali kako doći do rekurzivnog postupka (koji je još poznat pod nazivom *uniformna subdivizija Bézierove krivulje*), najprije trebamo još mrvicu matematike.

Razmotrimo slučaj kvadratne Bézierove krivulje. Točke te krivulje određene su sljedećim izrazom:

$$\vec{p}(t) = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \mathbf{B} \cdot \mathbf{R} \tag{7.11}$$

gdje je matrica $\mathbf{B}$ matrica koja odgovara kvadratnoj Bézierovoj krivulji.

Ideja rekurzivnog algoritma jest posao čitave krivulje podijeliti u dva jednostavnija posla: crtanje prve polovice krivulje te crtanje druge polovice krivulje. Ono što nas zanima jest kako napraviti *reparametrizaciju* krivulje: u prvom slučaju želimo utvrditi kako izgleda parametarski zapis prve polovice zadane Bézierove krivulje, odnosno zapis koji će, pretpostavimo li da se uvodi ovisnost o parametru λ, kada parametar λ mijenjamo od 0 i 1, generirati točke koje odgovaraju točkama zadane Bézierove krivulje koje se dobiju za vrijednosti parametra t od 0 do 0.5 (dakle, koje pokrivaju samo prvu polovicu zadane krivulje). Između λ i t postoji jednostavno linearno preslikavanje:

$$t = \frac{\lambda}{2}$$

pa supstitucijom u izraz (7.11) dobivamo točke prve polovice zadane Bézierove krivulje kao funkciju od λ:

$$\begin{aligned}\vec{p}(\lambda) &= \begin{bmatrix} \left(\frac{\lambda}{2}\right)^2 & \frac{\lambda}{2} & 1 \end{bmatrix} \cdot \mathbf{B} \cdot \mathbf{R} \\ &= \begin{bmatrix} \frac{\lambda^2}{4} & \frac{\lambda}{2} & 1 \end{bmatrix} \cdot \mathbf{B} \cdot \mathbf{R} \\ &= \begin{bmatrix} \lambda^2 & \lambda & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{B} \cdot \mathbf{R}\end{aligned}$$

što kraće možemo zapisati kao:

$$\vec{p}(\lambda) = \underline{\lambda} \cdot \mathbf{L} \cdot \mathbf{B} \cdot \mathbf{R}. \tag{7.12}$$

Matrica $\underline{\lambda}$ je pri tome jednoretčana matrica potencija parametra a matrica $\mathbf{L}$ se brine za transformaciju vrijednosti parametra. Ovaj se izraz međutim može svesti na izraz iz kojeg je vidljivo da smo opet dobili aproksimacijsku Bézierovu krivulju (tj. da je prva polovica zadane aproksimacijske Bézierove krivulje opet aproksimacijska Bézierova krivulja). Evo kako.

$$\begin{aligned}\vec{p}(\lambda) &= \underline{\lambda} \cdot \mathbf{L} \cdot \mathbf{B} \cdot \mathbf{R} \\ &= \underline{\lambda} \cdot (\mathbf{B} \cdot \mathbf{B}^{-1}) \cdot \mathbf{L} \cdot \mathbf{B} \cdot \mathbf{R} \\ &= \underline{\lambda} \cdot \mathbf{B} \cdot (\mathbf{B}^{-1} \cdot \mathbf{L} \cdot \mathbf{B} \cdot \mathbf{R}) \\ &= \underline{\lambda} \cdot \mathbf{B} \cdot \mathbf{R}_L\end{aligned}$$

gdje je $\mathbf{R}_L = \mathbf{B}^{-1} \cdot \mathbf{L} \cdot \mathbf{B} \cdot \mathbf{R}$. Ono što smo ovim postupkom napravili jest izračun nove matrice vrhova kontrolnog poligona koji generira prvu polovicu originalno zadane aproksimacijske Bézierove krivulje. Za kvadratnu aproksimacijsku Bézierovu krivulju matrica $\mathbf{B}$ glasi:

$$\mathbf{B} = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix},$$

njezin inverz je:

$$\mathbf{B}^{-1} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & \frac{1}{2} & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

pa je:

$$\mathbf{B}^{-1}\cdot\mathbf{L}\cdot\mathbf{B}=\begin{bmatrix}0&0&1\\0&\frac{1}{2}&1\\1&1&1\end{bmatrix}\cdot\begin{bmatrix}\frac{1}{4}&0&0\\0&\frac{1}{2}&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&-2&1\\-2&2&0\\1&0&0\end{bmatrix}$$
$$=\begin{bmatrix}1&0&0\\\frac{1}{2}&\frac{1}{2}&0\\\frac{1}{4}&\frac{1}{2}&\frac{1}{4}\end{bmatrix}$$

odnosno:

$$\mathbf{R}_L=\begin{bmatrix}1&0&0\\\frac{1}{2}&\frac{1}{2}&0\\\frac{1}{4}&\frac{1}{2}&\frac{1}{4}\end{bmatrix}\cdot\mathbf{R}. \tag{7.13}$$

Slijedi da, ako je originalna kvadratna Bézierova krivulja zadana vrhovima $\vec{r}_0$, $\vec{r}_1$ i $\vec{r}_2$, prva polovica te Bézierove krivulje je kvadratna Bézierova krivulja zadana kontrolnim poligonom čiji su vrhovi $\vec{r}_{L,0}=\vec{r}_0$, $\vec{r}_{L,1}=\frac{1}{2}\cdot\vec{r}_0+\frac{1}{2}\cdot\vec{r}_1$ te $\vec{r}_{L,2}=\frac{1}{4}\cdot\vec{r}_0+\frac{1}{2}\cdot\vec{r}_1+\frac{1}{4}\cdot\vec{r}_2$. Posao crtanja ove krivulje možemo prepustiti rekurzivnom pozivu.

Pogledajmo sada kako doći do druge polovice originalno zadane Bézierove krivlje. Pretpostavimo da želimo dobiti novu krivulju koja ovisi o parametru μ i čija se točka za $\mu=0$ poklapa s točkom originalne Bézierove krivulje za $t=0.5$ te čija se točka za $\mu=1$ poklapa s točkom originalne Bézierove krivulje za $t=1$. Očito, za zadani μ, ako želimo linearno preslikavanje, pripadni t određen je izrazom $t=\frac{\mu}{2}+\frac{1}{2}$. Uvrštavanjem u izraz (7.11) slijedi:

$$\begin{aligned}\vec{p}(\mu)&=\begin{bmatrix}\left(\frac{\mu}{2}+\frac{1}{2}\right)^2&\frac{\mu}{2}+\frac{1}{2}&1\end{bmatrix}\cdot\mathbf{B}\cdot\mathbf{R}\\&=\begin{bmatrix}\frac{\mu^2}{4}+\frac{\mu}{2}+\frac{1}{4}&\frac{\mu}{2}+\frac{1}{2}&1\end{bmatrix}\cdot\mathbf{B}\cdot\mathbf{R}\\&=\begin{bmatrix}\mu^2&\mu&1\end{bmatrix}\cdot\begin{bmatrix}\frac{1}{4}&0&0\\\frac{1}{2}&\frac{1}{2}&0\\\frac{1}{4}&\frac{1}{2}&1\end{bmatrix}\cdot\mathbf{B}\cdot\mathbf{R}\end{aligned}$$

što kraće možemo zapisati kao:

$$\vec{p}(\mu)=\underline{\mu}\cdot\mathbf{D}\cdot\mathbf{B}\cdot\mathbf{R}. \tag{7.14}$$

Matrica $\underline{\mu}$ je pri tome jednoretčana matrica potencija parametra a matrica $\mathbf{D}$ se brine za transformaciju vrijednosti parametra. I ovaj se izraz može svesti na izraz iz kojeg je vidljivo da smo opet dobili aproksimacijsku Bézierovu krivulju (tj. da je druga polovica zadane aproksimacijske Bézierove krivulje opet aproksimacijska

Bézierova krivulja). Evo kako.

$$\begin{aligned}\vec{p}(\mu) &= \underline{\mu} \cdot \mathbf{D} \cdot \mathbf{B} \cdot \mathbf{R} \\ &= \underline{\mu} \cdot (\mathbf{B} \cdot \mathbf{B}^{-1}) \cdot \mathbf{D} \cdot \mathbf{B} \cdot \mathbf{R} \\ &= \underline{\mu} \cdot \mathbf{B} \cdot (\mathbf{B}^{-1} \cdot \mathbf{D} \cdot \mathbf{B} \cdot \mathbf{R}) \\ &= \underline{\mu} \cdot \mathbf{B} \cdot \mathbf{R}_D\end{aligned}$$

gdje je $\mathbf{R}_D = \mathbf{B}^{-1} \cdot \mathbf{D} \cdot \mathbf{B} \cdot \mathbf{R}$. Ono što smo ovim postupkom napravili jest izračun nove matrice vrhova kontrolnog poligona koji generira drugu polovicu originalno zadane aproksimacijske Bézierove krivulje. Matrice $\mathbf{B}$ i $\mathbf{B}^{-1}$ prethodno smo već napisali, pa dalje možemo računati:

$$\begin{aligned}\mathbf{B}^{-1} \cdot \mathbf{D} \cdot \mathbf{B} &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & \frac{1}{2} & 1 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}\end{aligned}$$

odnosno:

$$\mathbf{R}_D = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{R}. \tag{7.15}$$

Slijedi da, ako je originalna kvadratna Bézierova krivulja zadana vrhovima $\vec{r}_0$, $\vec{r}_1$ i $\vec{r}_2$, druga polovica te Bézierove krivulje je kvadratna Bézierova krivulja zadana kontrolnim poligonom čiji su vrhovi $\vec{r}_{D,0} = \frac{1}{4} \cdot \vec{r}_0 + \frac{1}{2} \cdot \vec{r}_1 + \frac{1}{4} \cdot \vec{r}_2$, $\vec{r}_{D,1} = \frac{1}{2} \cdot \vec{r}_1 + \frac{1}{2} \cdot \vec{r}_2$ te $\vec{r}_{D,2} = \vec{r}_2$. Posao crtanja ove krivulje možemo prepustiti rekurzivnom pozivu.

Pretpostavimo li, dakle, da imamo originalno zadanu kvadratnu aproksimacijsku Bezirovu krivulju čiji su vrhovi kontrolnog poligona $\vec{r}_0$, $\vec{r}_1$ i $\vec{r}_2$, možemo izračunati dva nova kontrolna poligona: $\vec{r}_{L,0}$, $\vec{r}_{L,1}$ i $\vec{r}_{L,2}$ koji generira prvu polovicu krivulje te $\vec{r}_{D,0}$, $\vec{r}_{D,1}$ i $\vec{r}_{D,2}$ koji generira drugu polovicu krivulje, i svaku od te dvije nove krivulje možemo dalje rekurzivno dijeliti. Sada možemo definirati pseudokod algoritma.

crtaj($\vec{r}_0$, $\vec{r}_1$, $\vec{r}_2$, *dubina*)
  ako je *dubina*==0 **tada**
    crtaj_linijski_segment($\vec{r}_0$, $\vec{r}_2$**)**
    izlaz
  kraj ako
  $\vec{r}_{L,0} = \vec{r}_0$
  $\vec{r}_{L,1} = \frac{1}{2} \cdot \vec{r}_0 + \frac{1}{2} \cdot \vec{r}_1$
  $\vec{r}_{L,2} = \frac{1}{4} \cdot \vec{r}_0 + \frac{1}{2} \cdot \vec{r}_1 + \frac{1}{4} \cdot \vec{r}_2$

$\vec{r}_{D,0} = \vec{r}_{L,2}$
$\vec{r}_{D,1} = \frac{1}{2} \cdot \vec{r}_1 + \frac{1}{2} \cdot \vec{r}_2$
$\vec{r}_{D,2} = \vec{r}_2$
$dubina - -$
crtaj($\vec{r}_{L,0}$, $\vec{r}_{L,1}$, $\vec{r}_{L,2}$, $dubina$)
crtaj($\vec{r}_{D,0}$, $\vec{r}_{D,1}$, $\vec{r}_{D,2}$, $dubina$)
kraj

Pretpostavka prethodnog pseudokoda je da je uvjet zaustavljanja unaprijed zadana dubina koja se predaje pri prvom pozivu. Opisani rekurzivni algoritam nudi međutim i druge izvedbe kriterija zaustavljanja; primjerice, moguće je ostvariti verziju koja će provjeriti koliko dobro pravac povučen između $\vec{r}_0$ i $\vec{r}_2$ aproksimira krivulju: ako je pogreška mala, rekurzija se može prekinuti i crta se linijski segment; ako je pogreška velika, nastavlja se rekurzivna podjela. Ovakav pristup rezultirat će time da se zakrivljeniji dijelovi krivulje dijele do veće dubine a manje zakrivljeni do manje što će u konačnici ubrzati crtanje krivulje odnosno smanjiti broj primitiva koje treba nacrtati.

Izvedite za vježbu matrice $\mathbf{L}$ i $\mathbf{D}$ odnosno izraze za $\mathbf{R}_L$ i $\mathbf{R}_D$ za slučaj kubnih Bézierovih krivulja. Jesu li i u tom slučaju svi koeficijenti potencije broja 2?

Svojstva aproksimacijskih Bézierovih krivulja

1. Bézierova krivulja interpolira prvu i zadnju kontrolnu točku a ostale aproksimira. Ovo smo pokazali kod izvoda Bézierovim težinskim funkcijama.

2. Tangenta u početnoj odnosno u posljednjoj točki Bézierove krivulje kolinearna je s prvim odnosno posljednjim bridom kontrolnog poligona. Ovo smo također pokazali kod izvoda Bézierovim težinskim funkcijama.

3. Bézierova krivulja leži u konveksnoj ljusci svojih kontrolnih točaka. Konveksna ljuska olakšava postupak ispitivanja da li se krivulja siječe s nečim drugim jer je najprije moguće utvrditi postoji li sjecište s konveksnom ljuskom, što je bitno jednostavnije. Ako takvo sjecište ne postoji, onda nije potrebno daljnje ispitivanje.

4. Svojstvo smanjenja varijacije: krivulja nema više valova od kontrolnog poligona (ili drugim riječima, ravnina kojom presiječemo krivulju nema više sjecišta s tom ravninom no što ih ima kontrolni poligon s tom ravninom).

5. Krivulja nema svojstvo lokalnog nadzora. Naime, ukoliko pomaknemo samo jednu točku kojom smo zadali krivulju, cijela će krivulja promijeniti oblik. Poželjno bi bilo kada bi se krivulja promijenila samo u okolici pomaknute točke, no ovo kod Bézierovih krivulja ne vrijedi.

6. Broj točaka u direktnoj je vezi sa stupnjem krivulje. Npr. krivulja četvrtog stupnja zadana je pomoću pet kontrolnih točaka.

7. Neovisnost o afinim transformacijama (translacije, rotacije, skaliranja). To znači da ako Afinom transformacijom želimo transformirati Bézierovu krivulju, dovoljno je transformirati samo njezine kontrolne točke pa konstruirati krivulju. Perspektivne transformacije nisu afine transformacije pa za njih ovo ne vrijedi.

8. Simetričnost. Krivulja ima isti izgled ako zamijenimo redosljed kontrolnih točaka (tako da točka koja je bila prva postane zadnja, točka koja je bila druga postane predzadnja, itd).

S obzirom na navedena svojstva i postojanje učinkovitih načinja njihova prikaza, Bézierove su krivulje vrlo često korištene u računalnoj grafici.

7.3.2 Interpolacijska Bézierova krivulja

Interpolacijska krivulja prolazi svim zadanim točkama. Zadat ćemo točke (odnosno radij-vektore točaka) kroz koje želimo da krivulja prođe za neki parametar t. Zatim ćemo na temelju tih točaka izračunati kontrolne točke aproksimacijske krivulje, koja će proći kroz naše tražene točke.

Problem se općenito može zapisati ovako. Zadani su radij-vektori:

$$\vec{p}_0 = \vec{p}(t_0),\ \vec{p}_1 = \vec{p}(t_1),\ \ldots,\ \vec{p}_i = \vec{p}(t_i),\ \ldots,\ \vec{p}_n = \vec{p}(t_n).$$

Kako je krivulja zadana s $n+1$ radij-vektorom, krivulja je n-tog stupnja. Za opis krivulje koristit ćemo Bézierove težinske funkcije. Neka je kontrolni poligon aproksimacijske krivulje zadan vektorima $\vec{a}_0, \ldots, \vec{a}_n$ (koje ne znamo). Tada je svaka točka aproksimacijske krivulje dana sumom:

$$\vec{p}(t) = \sum_{i=0}^{n} \vec{a}_i f_{i,n}(t).$$

Postavimo sada zahtjev da ta krivulja *mora* proći kroz zadane radij-vektore $\vec{p}_i$.

$$\vec{p}(t_i) = \sum_{j=0}^{n} \vec{a}_j f_{j,n}(t_i) = \vec{p}_i.$$

Uočimo da smo ovime definirali $(n+1)$ jednadžbu, jer i ide od 0 do n. To pak možemo zapisati i matrično:

$$\begin{bmatrix} \vec{p}_0 \\ \vec{p}_1 \\ \vdots \\ \vec{p}_n \end{bmatrix} = \begin{bmatrix} f_{0,n}(t_0) & f_{1,n}(t_0) & \cdots & f_{n,n}(t_0) \\ f_{0,n}(t_1) & f_{1,n}(t_1) & \cdots & f_{n,n}(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ f_{0,n}(t_n) & f_{1,n}(t_n) & \cdots & f_{n,n}(t_n) \end{bmatrix} \cdot \begin{bmatrix} \vec{a}_0 \\ \vec{a}_1 \\ \vdots \\ \vec{a}_n \end{bmatrix} \tag{7.16}$$

ili kraće:

$$\mathbf{P} = \mathbf{F} \cdot \mathbf{A} \quad \Rightarrow \quad \mathbf{A} = \mathbf{F}^{-1} \cdot \mathbf{P}. \tag{7.17}$$

Iz ovih jednadžbi potrebno je odrediti matricu $\mathbf{A}$. Nakon toga su nam poznati svi vektori $\vec{a}_i$ te se pomoću njih i Bézierovih težinksih funkcija može direktno crtati aproksimacijska Bézierova krivulja koja pri tome interpolira izvorno zadane točke, ili se mogu izračunati radij-vektori točaka kontrolnog poligona aproksimacijske krivulje i zatim crtati krivulju pomoću Bernsteinovih težinskih funkcija.

Vrlo često se u praksi zadaju samo točke kroz koje želimo provući krivulju, ali se pri tome ne specificira za koju vrijednost parametra t krivulja mora proći kroz koju točku. Tada se za parametar može odabrati i vrlo jednostavan oblik:

$$t_i = \frac{i}{n}$$

pa se prethodni matrični račun pojednostavljuje:

$$\begin{bmatrix} \vec{p}_0 \\ \vec{p}_1 \\ \vdots \\ \vec{p}_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & f_{1,n}(\frac{1}{n}) & \cdots & f_{n,n}(\frac{1}{n}) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & f_{1,n}(\frac{n}{n}) & \cdots & f_{n,n}(\frac{n}{n}) \end{bmatrix} \cdot \begin{bmatrix} \vec{a}_0 \\ \vec{a}_1 \\ \vdots \\ \vec{a}_n \end{bmatrix}$$

Kao primjer možemo uzeti krivulju trećeg stupnja koju želimo provući kroz četiri točke: $\vec{p}_0$, $\vec{p}_1$, $\vec{p}_2$ i $\vec{p}_3$. Budući da vrijednosti parametara nisu zadane, uzet ćemo parametre prema relaciji:

$$t_i = \frac{i}{n}$$

čime dobivamo:

$$\vec{p}_0 = p(\frac{0}{3}),\ \vec{p}_1 = p(\frac{1}{3}),\ \vec{p}_2 = p(\frac{2}{3}),\ \vec{p}_3 = p(\frac{3}{3}).$$

Bézierove težinske funkcije za krivulju trećeg stupnja glase:

$$\begin{aligned} f_{0,3}(t) &= 1 \\ f_{1,3}(t) &= 3t - 3t^2 + t^3 \\ f_{2,3}(t) &= 3t^2 - 2t^3 \\ f_{3,3}(t) &= t^3 \end{aligned}$$

Uvrstimo li ovo u matricu, dobivamo:

$$\begin{bmatrix} \vec{p}_0 \\ \vec{p}_1 \\ \vec{p}_2 \\ \vec{p}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{19}{27} & \frac{7}{27} & \frac{1}{27} \\ 1 & \frac{26}{27} & \frac{20}{27} & \frac{8}{27} \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \vec{a}_0 \\ \vec{a}_1 \\ \vec{a}_2 \\ \vec{a}_3 \end{bmatrix}$$

Slijedi da je:

$$\mathbf{A} = \frac{1}{18} \cdot \begin{bmatrix} 18 & 0 & 0 & 0 \\ -33 & 54 & -27 & 6 \\ 21 & -81 & 81 & -21 \\ -6 & 27 & -54 & 33 \end{bmatrix} \cdot \mathbf{P}$$

odnosno po komponentama:

$$\begin{bmatrix} \vec{a}_0 \\ \vec{a}_1 \\ \vec{a}_2 \\ \vec{a}_3 \end{bmatrix} = \frac{1}{18} \cdot \begin{bmatrix} 18 & 0 & 0 & 0 \\ -33 & 54 & -27 & 6 \\ 21 & -81 & 81 & -21 \\ -6 & 27 & -54 & 33 \end{bmatrix} \cdot \begin{bmatrix} \vec{p}_0 \\ \vec{p}_1 \\ \vec{p}_2 \\ \vec{p}_3 \end{bmatrix}$$

Sada kada smo izračunali tražene vektore, jednadžba aproksimacijske Bézierove krivulje (koja je interpolacijska s obzirom na početno zadane točke $\vec{p}_i$) glasi:

$$\vec{p}(t) = \sum_{i=0}^{3} \vec{a}_i f_{i,3}(t) = 1 \cdot \vec{a}_0 + (3t - 3t^2 + t^3) \cdot \vec{a}_1 + (3t^2 - 2t^3) \cdot \vec{a}_2 + t^3 \cdot \vec{a}_3.$$

U ovom primjeru bilo je zadano $n + 1$ točka. No interpolacijska se krivulja može provlačiti i na temelju drugih podataka. Za izračun aproksimacijske krivulje potreban nam je $n + 1$ uvjet. Neki od načina zadavanja su sljedeći:

- $n + 1$ poznata točka (kao u primjeru),
- n poznatih točaka i poznata tangenta u nekoj od točaka (obično prva ili zadnja točka) ili općenito
- $n + 1$ poznatih uvjeta (bilo točaka, bilo derivacija, bilo kombinacija).

Krivulju je moguće računati i na temelju poznavanja viših derivacija i slično.

Interpolacijsku Bézierovu krivulju na isti način možemo dobiti i uporabom Bernsteinovih težinskih funkcija. Pogledat ćemo to na konkretnom primjeru gdje želimo provući Bézierovu krivulju kroz 4 zadane točke koristeći uniformno uzorkovanje vrijednosti parametra. Neka su zadane 4 točke kroz koje krivulja mora proći: $\vec{p}_0$, $\vec{p}_1$, $\vec{p}_2$ i $\vec{p}_3$. Zadavanjem četiri ograničenja definirana je kubna Bézierova krivulja koja je određena izrazom (7.10). Pri tome znamo da krivulja mora proći kroz zadane točke, odnosno da mora vrijediti:

$$\vec{p}_0 = \vec{p}(0) = \begin{bmatrix} 0^3 & 0^2 & 0 & 1 \end{bmatrix} \cdot \mathbf{B} \cdot \mathbf{R},$$

$$\vec{p}_1 = \vec{p}(\frac{1}{3}) = \begin{bmatrix} \left(\frac{1}{3}\right)^3 & \left(\frac{1}{3}\right)^2 & \frac{1}{3} & 1 \end{bmatrix} \cdot \mathbf{B} \cdot \mathbf{R},$$

$$\vec{p}_2 = \vec{p}(\frac{2}{3}) = \begin{bmatrix} \left(\frac{2}{3}\right)^3 & \left(\frac{2}{3}\right)^2 & \frac{2}{3} & 1 \end{bmatrix} \cdot \mathbf{B} \cdot \mathbf{R} \text{ te}$$

$$\vec{p}_3 = \vec{p}(1) = \begin{bmatrix} 1^3 & 1^2 & 1 & 1 \end{bmatrix} \cdot \mathbf{B} \cdot \mathbf{R}.$$

Ove četiri jednadžbe možemo skupiti u jednu matričnu jednadžbu:

$$\begin{bmatrix} \vec{p}_0 \\ \vec{p}_1 \\ \vec{p}_2 \\ \vec{p}_3 \end{bmatrix} = \begin{bmatrix} 0^3 & 0^2 & 0 & 1 \\ \left(\frac{1}{3}\right)^3 & \left(\frac{1}{3}\right)^2 & \frac{1}{3} & 1 \\ \left(\frac{2}{3}\right)^3 & \left(\frac{2}{3}\right)^2 & \frac{2}{3} & 1 \\ 1^3 & 1^2 & 1 & 1 \end{bmatrix} \cdot \mathbf{B} \cdot \mathbf{R}$$

gdje se s lijeve strane nalazi matrica $\mathbf{P}$. Uvođenjem oznake $\mathbf{T}$ za matricu potencija odabranih vrijednosti parametara možemo pisati:

$$\mathbf{P} = \mathbf{T} \cdot \mathbf{B} \cdot \mathbf{R} \tag{7.18}$$

U radnom trodimenzionalnom prostoru matrice $\mathbf{P}$ i $\mathbf{R}$ su matrice 4×3. Jedina nepoznanica u izrazu (7.18) je $\mathbf{R}$ odnosno matrica vrhova kontrolnog poligona koji definira aproksimacijsku Bézierovu krivulju koja će proći kroz četiri zadane točke. Tu matricu možemo odrediti množeći čitav izraz s lijeve strane inverzom umnoška $\mathbf{T} \cdot \mathbf{B}$:

$$\begin{aligned} (\mathbf{T} \cdot \mathbf{B})^{-1} \cdot \Big| \qquad & \mathbf{P} = \mathbf{T} \cdot \mathbf{B} \cdot \mathbf{R} \\ (\mathbf{T} \cdot \mathbf{B})^{-1} \cdot \mathbf{P} &= (\mathbf{T} \cdot \mathbf{B})^{-1} \cdot \mathbf{T} \cdot \mathbf{B} \cdot \mathbf{R} \end{aligned}$$

što daje:

$$\mathbf{R} = \mathbf{B}^{-1} \cdot \mathbf{T}^{-1} \cdot \mathbf{P} \tag{7.19}$$

Uočite da izraz (7.19) vrijedi za proizvoljnu Bézierovu krivlju a ne samo kubnu. Dakako, općenito kod krivulje n-tog stupnja matrice $\mathbf{B}$ i $\mathbf{T}$ će biti kvadratne matrice $(n+1) \times (n+1)$ dok će matrice $\mathbf{R}$ i $\mathbf{P}$ biti $(n+1) \times d$ gdje je d dimenzija prostora u kojem radimo (za slučaj radnog 3D prostora, matrica će biti $(n+1) \times 3$).

Pri implementaciji programskih biblioteka koje na učinkovit način crtaju Bézierove krivulje, poželjno je izbjeći operacije izračuna matričnog inverza zbog zahtjevnosti postupka. Također, sama matrica potencija može biti loše uvjetovana čime izračun inverza može dati vrlo netočne rezultate. Stoga se izraz:

$$\mathbf{P} = (\mathbf{T} \cdot \mathbf{B}) \cdot \mathbf{R}$$

može promatrati kao sustav linearnih jednadžbi koji se onda rješava učinkovitije. Načini rješavanja takvih sustava izlaze iz okvira ovog udžbenika i neće se obrađivati. Izraz (7.19) koji nam je interesantan s teorijskog stajališta i potreban je za daljnje izvode, dalje će biti korišten u tekstu.

Dobiveni izraz za matricu $\mathbf{R}$ sada možemo supstituirati direktno u izraz (7.10) pa slijedi:

$$\vec{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \mathbf{B} \cdot \mathbf{B}^{-1} \cdot \mathbf{T}^{-1} \cdot \mathbf{P}.$$

Uočimo da se matrica $\mathbf{B}$ i njezin inverz poništavaju pa je konačni oblik (u ovom primjeru kubne) interpolacijske Bézierove krivulje:

$$\vec{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \mathbf{T}^{-1} \cdot \mathbf{P}. \tag{7.20}$$

Kako je izraz (7.20) vrlo sličan izrazu (7.10), rezimirajmo što smo dobili.

- Aproksimacijska Bézierova krivulja prikazana uporabom Bernsteinovih težinskih funkcija određena je izrazom (7.10): jednoretčana matrica potencija parametra množi se kvadratnom matricom koeficijenata Bernsteinovih težinskih polinoma ($\mathbf{B}$) i potom matricom radij-vektora kontrolnog poligona ($\mathbf{R}$).
- Interpolacijska Bézierova krivulja određena je izrazom (7.20): jednoretčana matrica potencija parametra množi se inverzom kvadratne matrice potencija odabranih vrijednosti parametara za koje krivulja mora proći kroz zadane točke i potom se množi matricom vektora točaka kroz koje krivulja mora proći.

7.4 Parametarski prikaz krivulja pomoću polinoma

Priču o krivuljama koju smo započeli razmatranjem Bézierove krivulje možemo nastaviti razmatranjem krivulja u radnom prostoru koje su zadane parametarski pri čemu je svaka koordinata točke zadana polinomom. Bez gubitka općenitosti razmotrit ćemo slučaj kada su koordinate točke opisane kubnim polinomima (za bilo koji drugi stupanj polinoma postupak je identičan onome koji ćemo opisati). Neka je s T_K označena neka točka krivulje. U 3D radnom prostoru njezine će koordinate biti $T_{K,1}$, $T_{K,2}$ te $T_{K,3}$ (odnosno x, y i z). Svaka od tih koordinata opisana je kubnim polinomom:

$$\begin{aligned} T_{K,1} &= a_1 \cdot t^3 + b_1 \cdot t^2 + c_1 \cdot t + d_1 \\ T_{K,2} &= a_2 \cdot t^3 + b_2 \cdot t^2 + c_2 \cdot t + d_2 \\ T_{K,3} &= a_3 \cdot t^3 + b_3 \cdot t^2 + c_3 \cdot t + d_3 \end{aligned}$$

Koristeći uobičajenu notaciju za točku krivulje koja ovisi o parametru t možemo pisati:

$$\vec{p}(t) = \begin{bmatrix} T_{K,1} & T_{K,2} & T_{K,3} \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \\ d_1 & d_2 & d_3 \end{bmatrix} \tag{7.21}$$

što ćemo kraće pisati:

$$\vec{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \mathbf{K}. \tag{7.22}$$

Matrica $\mathbf{K}$ je matrica koeficijenata polinoma: broj redaka te matrice je $n+1$ gdje je n odabrani stupanj polinoma koje koristimo a broj stupaca odgovara broju komponenata koje ima točka; ako smo u 3D radnom prostoru, matrica će imati 3 stupca jer svaki stupac odgovara jednom polinomu. Ovako definirana krivulja jednoznačno je određena zadavanjem $(n+1)$-og ograničenja. Konkretno, kako koristimo kubne polinome, trebamo četiri ograničenja: primjerice, četiri točke kroz koje će krivulja proći. Neka te točke budu $\vec{p}_0$, $\vec{p}_1$, $\vec{p}_2$ i $\vec{p}_3$ i neka krivulja kroz njih prolazi za redom $t = 0$, $t = \frac{1}{3}$, $t = \frac{2}{3}$ i $t = 1$. Tada možemo pisati:

$$\begin{bmatrix} \vec{p}_0 \\ \vec{p}_1 \\ \vec{p}_2 \\ \vec{p}_3 \end{bmatrix} = \begin{bmatrix} 0^3 & 0^2 & 0 & 1 \\ \left(\frac{1}{3}\right)^3 & \left(\frac{1}{3}\right)^2 & \frac{1}{3} & 1 \\ \left(\frac{2}{3}\right)^3 & \left(\frac{2}{3}\right)^2 & \frac{2}{3} & 1 \\ 1^3 & 1^2 & 1 & 1 \end{bmatrix} \cdot \mathbf{K}$$

odnosno kraće (i uz već uvedene oznake):

$$\mathbf{P} = \mathbf{T} \cdot \mathbf{K}.$$

Kako su matrice $\mathbf{P}$ i $\mathbf{T}$ zadane, matricu $\mathbf{K}$ sada možemo odrediti množeći čitav izraz inverzom matrice $\mathbf{T}$ s lijeve strane pa dobivamo:

$$\mathbf{K} = \mathbf{T}^{-1} \cdot \mathbf{P} \tag{7.23}$$

čime, uvrštavanjem izraza za $\mathbf{K}$ u izraz (7.22), dolazimo do općenitog izraza za točke krivulje koja je zadana tako da prolazi kroz $(n+1) = 4$ zadane točke:

$$\vec{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \mathbf{T}^{-1} \cdot \mathbf{P}. \tag{7.24}$$

Usporedite sada izraz (7.24) i izraz (7.20) koji smo dobili za interpolacijsku Bézierovu krivulju – isti su. Krivulje zadane u radnom prostoru parametarski putem polinoma i interpolacijska Bézierova krivulja *iste su krivulje*. Ovo možemo pokazati i direktno. Uvedimo oznaku τ za jednoretčanu matricu potencija parametra: $\tau = \begin{bmatrix} t^n & t^{n-1} & \cdots & t & 1 \end{bmatrix}$. Interpolacijska Bézierova krivulja određena je izrazom:

$$\vec{p}(t) = \tau \cdot \mathbf{T}^{-1} \cdot \mathbf{P}$$

dok je parametarska krivulja definirana polinomima u radnom prostoru određena s:

$$\vec{p}(t) = \tau \cdot \mathbf{K}.$$

Sada je lako pokazati da za proizvoljnu matricu $\mathbf{K}$ (dakle, proizvoljnu krivulju koja je opisiva na ovaj način) možemo pronaći matrice $\mathbf{T}$ i $\mathbf{P}$ takve da vrijedi

$\mathbf{T}^{-1} \cdot \mathbf{P} = \mathbf{K}$. Ako je matrica K dimenzija $(n+1) \times d$, matrica $\mathbf{T}$ morat će biti dimenzija $(n+1) \times (n+1)$ jer je to matrica potencija odabranih vrijednosti parametara za koje interpolacijska Bézierova krivulja prolazi kroz zadane točke određene matricom $\mathbf{P}$. Matricu $\mathbf{T}$ možemo izgraditi na proizvoljan način; primjerice, popunjavajući retke tako da t mijenjamo uniformno: 0, $\frac{1}{n}$, $\frac{2}{n}$, …, 1 i za svaki t računamo t^n, t^{n-1}, itd. Treba samo paziti da u više od jednog retka ne stavimo istu vrijednost za parametar t. Ovako konstruirana matrica $\mathbf{T}$ bit će invertibilna. Jednom kada smo konstruirali matricu $\mathbf{T}$, matricu $\mathbf{P}$ možemo izračunati iz jednakosti $\mathbf{T}^{-1} \cdot \mathbf{P} = \mathbf{K}$ množenjem čitavog izraza matricom $\mathbf{T}$:

$$\mathbf{P} = \mathbf{T} \cdot \mathbf{K}.$$

Interpolacijska Bézierova krivulja uz ovako određene matrice $\mathbf{T}$ i $\mathbf{P}$ identična je krivulji opisanoj matricom $\mathbf{K}$. Uočimo i da rastav na matrice $\mathbf{T}$ i $\mathbf{P}$ nije jedinstven: za svaku matricu $\mathbf{T}$ koju konstruiramo na opisani način možemo pronaći odgovarajuću matricu $\mathbf{P}$. Vrijedi i obrat tvrdnje: svaka interpolacijska Bézierova krivulja može se prikazati kao parametarski zadana krivulja u radnom prostoru koja je opisana polinomima: pripadna matrica $\mathbf{K}$ naprosto je jednaka umnošku $\mathbf{T}^{-1} \cdot \mathbf{P}$.

Pokažimo da za svaku krivulju određenu matricom $\mathbf{K}$ postoji i pripadna aproksimacijska Bézierova krivulja (što je i za očekivati, zbog veze između interpolacijske i aproksimacijske Bézierove krivulje). Aproksimacijska Bézierova krivulja određena je izrazom:

$$\vec{p}(t) = \tau \cdot \mathbf{B} \cdot \mathbf{R}$$

dok je parametarska krivulja definirana polinomima u radnom prostoru određena s:

$$\vec{p}(t) = \tau \cdot \mathbf{K}.$$

Sada je lako pokazati da za proizvoljnu matricu $\mathbf{K}$ možemo pronaći matricu $\mathbf{R}$ takvu da je $\mathbf{B} \cdot \mathbf{R} = \mathbf{K}$. Matrica $\mathbf{B}$ kod aproksimacijske Bézierove krivulje izravno je određena zadavanjem stupnja krivulje i invertibilna je. Ako je matrica K dimenzija $(n+1) \times d$, matrica $\mathbf{B}$ morat će biti dimenzija $(n+1) \times (n+1)$, odnosno pripadna Bézierova krivulja bit će stupnja n. Čitav izraz možemo pomnožiti inverzom matrice $\mathbf{B}$ s lijeve strane čime dobivamo potrebnu matricu $\mathbf{R}$:

$$\mathbf{R} = \mathbf{B}^{-1} \cdot \mathbf{K}.$$

Ovime smo pronašli vrhove kontrolnog poligona koji generira aproksimacijsku Bézierovu krivulju koja je identična krivulji zadanoj matricom $\mathbf{K}$. Vrijedi i obrat: za svaku aproksimacijsku Bézierovu krivulju postoji pripadna matrica $\mathbf{K}$ i ona je upravo jednaka umnošku $\mathbf{B} \cdot \mathbf{R}$.

Primjer: 11

Zadane su četiri točke u radnom prostoru: $\vec{p}_0 = (1, -1)$, $\vec{p}_1 = (1, 2)$, $\vec{p}_2 = (2, 4)$, $\vec{p}_3 = (0, 5)$, kroz koje treba provući parametarsku krivulju čije su komponente opisane polinomima. Pretpostavite da se koristi jednoliko uzorkovanje vrijednosti parametra. Odredite matrični prikaz krivulje i skicirajte krivulju.

Rješenje:

Za rješavanje zadatka koristit ćemo izraz (7.22). Potrebno je odrediti matricu $\mathbf{K}$ što ćemo učiniti prema izrazu (7.23). Kako imamo zadane četiri točke, polinomi će biti 3. stupnja pa će matrica $\mathbf{T}$ biti matrica 4×4. U i-tom retku te matrice nalazit će se potencije vrijednosti parametra za koji krivulja mora proći kroz i-tu točku. Neka za $t = 0$ krivulja prolazi kroz $\vec{p}_0$, za $t = \frac{1}{3}$ kroz $\vec{p}_1$, za $t = \frac{2}{3}$ kroz $\vec{p}_2$ i konačno za za $t = 1$ kroz $\vec{p}_3$. Ta je matrica:

$$\mathbf{T} = \begin{bmatrix} 0^3 & 0^2 & 0 & 1 \\ \left(\frac{1}{3}\right)^3 & \left(\frac{1}{3}\right)^2 & \frac{1}{3} & 1 \\ \left(\frac{2}{3}\right)^3 & \left(\frac{2}{3}\right)^2 & \frac{2}{3} & 1 \\ 1^3 & 1^2 & 1 & 1 \end{bmatrix}.$$

Matrica $\mathbf{P}$ sadrži u svakom retku po jednu točku:

$$\mathbf{P} = \begin{bmatrix} 1 & -1 \\ 1 & 2 \\ 2 & 4 \\ 0 & 5 \end{bmatrix}.$$

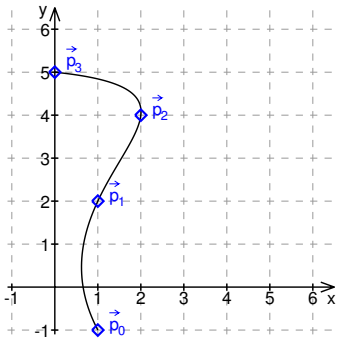
Prema izrazu (7.23), matricu $\mathbf{K}$ možemo odrediti na sljedeći način:

$$\begin{aligned} \mathbf{K} &= \mathbf{T}^{-1} \cdot \mathbf{P} \\ &= \begin{bmatrix} 0^3 & 0^2 & 0 & 1 \\ \left(\frac{1}{3}\right)^3 & \left(\frac{1}{3}\right)^2 & \frac{1}{3} & 1 \\ \left(\frac{2}{3}\right)^3 & \left(\frac{2}{3}\right)^2 & \frac{2}{3} & 1 \\ 1^3 & 1^2 & 1 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & -1 \\ 1 & 2 \\ 2 & 4 \\ 0 & 5 \end{bmatrix} \\ &= \begin{bmatrix} -4.5 & 13.5 & -13.5 & 4.5 \\ 9.0 & -22.5 & 18.0 & -4.5 \\ -5.5 & 9.0 & -4.5 & 1.0 \\ 1.0 & 0.0 & -0.0 & 0.0 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 \\ 1 & 2 \\ 2 & 4 \\ 0 & 5 \end{bmatrix} \\ &= \begin{bmatrix} -18.0 & 0.0 \\ 22.5 & -4.5 \\ -5.5 & 10.5 \\ 1.0 & -1.0 \end{bmatrix}. \end{aligned}$$

Krivulja je tada određena izrazom (7.22):

$$\vec{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -18.0 & 0.0 \\ 22.5 & -4.5 \\ -5.5 & 10.5 \\ 1.0 & -1.0 \end{bmatrix}.$$

Grafički prikaz krivulje dan je na slici u nastavku.



Pogledajmo još jedan primjer.

Primjer: 12

Parametarska krivulja čije su komponente opisane polinomima zadana je matricom $\mathbf{K}$ iz prethodnog primjera. Odredite vrhove kontrolnog poligona aproksimacijske Bézierove krivulje koja daje identičan prikaz.

Rješenje:

S obzirom da tražimo aproksimacijsku Bézierovu krivulju, tražit ćemo da vrijedi:

$$\mathbf{B} \cdot \mathbf{R} = \mathbf{K}.$$

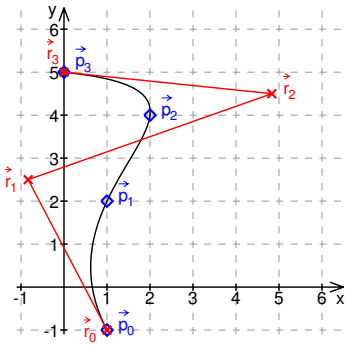
Kako je matrica $\mathbf{K}$ dimenzija 4×2, slijedi da je matrica $\mathbf{B}$ kvadratna ranga 4 odnosno da tražimo kubnu aproksimacijsku Bézierovu krivulju. Matricu $\mathbf{B}$ za kubnu Bézierovu krivulju već smo odredili:

$$\mathbf{B} = \begin{bmatrix} -1.00 & 3.00 & -3.00 & 1.00 \\ 3.00 & -6.00 & 3.00 & 0.00 \\ -3.00 & 3.00 & 0.00 & 0.00 \\ 1.00 & 0.00 & 0.00 & 0.00 \end{bmatrix}.$$

Matrica $\mathbf{R}$ je nepoznata a matricu $\mathbf{K}$ imamo. Stoga ćemo $\mathbf{R}$ izračunati na sljedeći način.

$$\begin{aligned} \mathbf{R} &= \mathbf{B}^{-1} \cdot \mathbf{K} \\ &= \begin{bmatrix} -1.00 & 3.00 & -3.00 & 1.00 \\ 3.00 & -6.00 & 3.00 & 0.00 \\ -3.00 & 3.00 & 0.00 & 0.00 \\ 1.00 & 0.00 & 0.00 & 0.00 \end{bmatrix}^{-1} \cdot \begin{bmatrix} -18.0 & 0.0 \\ 22.5 & -4.5 \\ -5.5 & 10.5 \\ 1.0 & -1.0 \end{bmatrix} \\ &= \begin{bmatrix} 0.00 & 0.00 & 0.00 & 1.00 \\ 0.00 & 0.00 & 1/3 & 1.00 \\ 0.00 & 1/3 & 2/3 & 1.00 \\ 1.00 & 1.00 & 1.00 & 1.00 \end{bmatrix} \cdot \begin{bmatrix} -18.0 & 0.0 \\ 22.5 & -4.5 \\ -5.5 & 10.5 \\ 1.0 & -1.0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & -1 \\ -5/6 & 5/2 \\ 29/6 & 9/2 \\ 0 & 5 \end{bmatrix}. \end{aligned}$$

Slijedi da su vrhovi kontrolnog poligona $\vec{r}_0 = (1, -1)$, $\vec{r}_1 = (-5/6, 5/2)$, $\vec{r}_2 = (29/6, 9/2)$ te $\vec{r}_3 = (0, 5)$. Slika u nastavku prikazuje krivulju s ucrtanim kontrolnim poligonom.



Razlog ovog kratkog izleta u parametarski prikaz krivulje u radnom prostoru polinomima jest uočiti da se na taj način ne dobiva ništa novo: to je samo drugi način na koji možemo opisivati iste krivulje. Međutim, porodica krivulja koje možemo opisati na ovaj način ne sadrži mnoštvo često korištenih krivulja – primjerice, kružnicu. Pogledajmo stoga općenitiji slučaj parametarskog opisivanja krivulja: opet ćemo koristiti polinome, ali ovaj puta u homogenom prostoru.

7.5 Prikaz krivulja pomoću razlomljenih funkcija

Jedan od načina zapisivanja krivulja koji do sada nismo spomenuli jest pomoću razlomljenih funkcija. Pri tome se krivulje opisuju parametarski uz jedan parametar t, u homogenom prostoru, a svaka koordinata točke polinomna je funkcija parametra t. Kako ćemo krivulje opisivati općenito u $3D$-prostoru, svaka točka T_K imat će svoje tri koordinate $T_{K,1}$ ili x, $T_{K,2}$ ili y i $T_{K,3}$ ili z u radnom prostoru, odnosno naziv T_{Kh} i četiri koordinate $T_{Kh,1}$, $T_{Kh,2}$, $T_{Kh,3}$ i $T_{Kh,h}$ u homogenom prostoru.

7.5.1 Prikaz krivulja pomoću kvadratnih razlomljenih funkcija

Kvadratne razlomljene funkcije omogućavaju nam jednostavan prikaz konika (krivulje koje nastaju kao presjecište stošca i ravnine: kružnice, elipse, parabole i hiperbole). Kako je riječ o kvadratnim funkcijama, funkcijske ovisnosti svih koordinata u homogenom prostoru bit će izražene kvadratnim polinomom, pa možemo pisati:

$$\begin{aligned} T_{Kh,1} &= a_1 \cdot t^2 + b_1 \cdot t + c_1 \\ T_{Kh,2} &= a_2 \cdot t^2 + b_2 \cdot t + c_2 \\ T_{Kh,3} &= a_3 \cdot t^2 + b_3 \cdot t + c_3 \\ T_{Kh,h} &= a \cdot t^2 + b \cdot t + c \end{aligned}$$

Povratkom u radni prostor dobiva se:

$$T_{K,1} = \frac{a_1 \cdot t^2 + b_1 \cdot t + c_1}{a \cdot t^2 + b \cdot t + c}$$
$$T_{K,2} = \frac{a_2 \cdot t^2 + b_2 \cdot t + c_2}{a \cdot t^2 + b \cdot t + c}$$
$$T_{K,3} = \frac{a_3 \cdot t^2 + b_3 \cdot t + c_3}{a \cdot t^2 + b \cdot t + c}$$

Razlog za naziv "razlomljene kvadratne" funkcije trebao bi biti jasan iz prethodnih jednadžbi. Zadržimo li se u homogenom prostoru, tada se jednadžbe po koordinatama mogu spojiti u jedan matrični zapis:

$$T_{Kh} = \begin{bmatrix} T_{Kh,1} & T_{Kh,2} & T_{Kh,3} & T_{Kh,h} \end{bmatrix} = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \end{bmatrix}$$
$$= \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \mathbf{K}.$$

Matrica $\mathbf{K}$ naziva se *karakteristična matrica krivulje*. Za određivanje matrice $\mathbf{K}$ dovoljno je poznavati tri točke kroz koje krivulja mora proći. Npr. neka prođe kroz točku T_A za $t = t_1 = 0$, kroz točku T_B za $t = t_2 = 0.5$ i kroz točku T_C za $t = t_3 = 1$. Tada vrijedi:

$$T_{Ah} = \begin{bmatrix} T_{Ah,1} & T_{Ah,2} & T_{Ah,3} & T_{Ah,h} \end{bmatrix} = \begin{bmatrix} t_1^2 & t_1 & 1 \end{bmatrix} \cdot \mathbf{K}$$
$$T_{Bh} = \begin{bmatrix} T_{Bh,1} & T_{Bh,2} & T_{Bh,3} & T_{Bh,h} \end{bmatrix} = \begin{bmatrix} t_2^2 & t_2 & 1 \end{bmatrix} \cdot \mathbf{K}$$
$$T_{Ch} = \begin{bmatrix} T_{Ch,1} & T_{Ch,2} & T_{Ch,3} & T_{Ch,h} \end{bmatrix} = \begin{bmatrix} t_3^2 & t_3 & 1 \end{bmatrix} \cdot \mathbf{K}$$

što možemo zapisati i matrično:

$$\begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} T_{Ah,1} & T_{Ah,2} & T_{Ah,3} & T_{Ah,h} \\ T_{Bh,1} & T_{Bh,2} & T_{Bh,3} & T_{Bh,h} \\ T_{Ch,1} & T_{Ch,2} & T_{Ch,3} & T_{Ch,h} \end{bmatrix} = \begin{bmatrix} t_1^2 & t_1 & 1 \\ t_2^2 & t_2 & 1 \\ t_3^2 & t_3 & 1 \end{bmatrix} \cdot \mathbf{K}.$$

Da bismo odredili matricu $\mathbf{K}$, cijelu jednadžbu potrebno je pomnožiti s inverznom matricom potencija odabranih vrijednosti parametara i to s lijeve strane:

$$\mathbf{K} = \begin{bmatrix} t_1^2 & t_1 & 1 \\ t_2^2 & t_2 & 1 \\ t_3^2 & t_3 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} t_1^2 & t_1 & 1 \\ t_2^2 & t_2 & 1 \\ t_3^2 & t_3 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_{Ah,1} & T_{Ah,2} & T_{Ah,3} & T_{Ah,h} \\ T_{Bh,1} & T_{Bh,2} & T_{Bh,3} & T_{Bh,h} \\ T_{Ch,1} & T_{Ch,2} & T_{Ch,3} & T_{Ch,h} \end{bmatrix}.$$

Uvrstimo li zadane t-ove u izraz, dobiva se:

$$\mathbf{K} = \begin{bmatrix} 0^2 & 0 & 1 \\ 0.5^2 & 0.5 & 1 \\ 1^2 & 1 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix}.$$

Zadamo li sada i točke T_A, T_B i T_C, matrica $\mathbf{K}$ može se odrediti u potpunosti.

7.5.2 Parametarske derivacije u homogenom prostoru

Svim koordinatama definirali smo funkcijsku ovisnost o parametru t. U ovom slučaju ta je funkcijska ovisnost definirana kvadratnim polinomom. No to znači da se te funkcije daju i derivirati. Pogledajmo kako bi izgledala prva derivacija po parametru t. Funkcijske ovisnosti koordinata definirane su relacijama:

$$\begin{aligned} T_{Kh,1} &= a_1 \cdot t^2 + b_1 \cdot t + c_1 \\ T_{Kh,2} &= a_2 \cdot t^2 + b_2 \cdot t + c_2 \\ T_{Kh,3} &= a_3 \cdot t^2 + b_3 \cdot t + c_3 \\ T_{Kh,h} &= a \cdot t^2 + b \cdot t + c \end{aligned}$$

Deriviranjem svake relacije po t dobiva se:

$$\begin{aligned} \frac{dT_{Kh,1}}{dt} &= a_1 \cdot 2t + b_1 \\ \frac{dT_{Kh,2}}{dt} &= a_2 \cdot 2t + b_2 \\ \frac{dT_{Kh,3}}{dt} &= a_3 \cdot 2t + b_3 \\ \frac{dT_{Kh,h}}{dt} &= a \cdot 2t + b \end{aligned}$$

Označimo li derivaciju u točki T_{Kh} po parametru t oznakom T'_{Kh} tada možemo derivaciju raspisati po komponentama u matričnom obliku:

$$\begin{aligned} T'_{Kh} &= \begin{bmatrix} T'_{Kh,1} & T'_{Kh,2} & T'_{Kh,3} & T'_{Kh,h} \end{bmatrix} = \begin{bmatrix} 2t & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \end{bmatrix} \\ &= \begin{bmatrix} 2t & 1 & 0 \end{bmatrix} \cdot \mathbf{K}. \end{aligned}$$

Možda da još jednom naglasimo što predstavlja oznaka T'_{Kh}. To nije derivacija krivulje u točki u geometrijskom smislu (što bismo trebali poistovjetiti s nagibom krivulje u toj točki). To je derivacija funkcija kojima su definirane ovisnosti o parametru t u nekoj proizvoljnoj točki T_{Kh}, odnosno gradijent u homogenom prostoru. Kako tih funkcija ima četiri, tako je i struktura T'_{Kh} četverokomponentni vektor.

Drugu derivaciju dobivamo deriviranjem prve derivacije; dobiva se:

$$\begin{aligned} \frac{d^2T_{Kh,1}}{dt^2} &= a_1 \cdot 2 \\ \frac{d^2T_{Kh,2}}{dt^2} &= a_2 \cdot 2 \\ \frac{d^2T_{Kh,3}}{dt^2} &= a_3 \cdot 2 \\ \frac{d^2T_{Kh,h}}{dt^2} &= a \cdot 2 \end{aligned}$$

ili matrično zapisano:

$$T''_{Kh} = \begin{bmatrix} T''_{Kh,1} & T''_{Kh,2} & T''_{Kh,3} & T''_{Kh,h} \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \end{bmatrix}$$
$$= \begin{bmatrix} 2 & 0 & 0 \end{bmatrix} \cdot \mathbf{K}.$$

Sve više derivacije daju:

$$\left.\begin{array}{l} \frac{d^n T_{Kh,1}}{dt^n} = 0 \\ \frac{d^n T_{Kh,2}}{dt^n} = 0 \\ \frac{d^n T_{Kh,3}}{dt^n} = 0 \\ \frac{d^n T_{Kh,h}}{dt^n} = 0 \end{array}\right\} \quad n > 2$$

odnosno matrično:

$$T^{(n)}_{Kh} = \begin{bmatrix} T^{(n)}_{Kh,1} & T^{(n)}_{Kh,2} & T^{(n)}_{Kh,3} & T^{(n)}_{Kh,h} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{K}, \quad n > 2.$$

Iz ovog jednostavnog izvoda jasno se vidi da su sve derivacije određene upravo karakterističnom matricom krivulje $\mathbf{K}$. Također se vidi da se sve derivacije mogu direktno dobiti samo deriviranjem elemenata matrice parametra t. Tako smo krenuli od:

$$T_{Kh} = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \mathbf{K}.$$

Prva derivacija je bila:

$$T'_{Kh} = \begin{bmatrix} 2t & 1 & 0 \end{bmatrix} \cdot \mathbf{K}.$$

Druga derivacija:

$$T''_{Kh} = \begin{bmatrix} 2 & 0 & 0 \end{bmatrix} \cdot \mathbf{K}.$$

I sve ostale više:

$$T^{(n)}_{Kh} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{K}, \quad n > 2.$$

7.5.3 Prikaz krivulja pomoću kubnih razlomljenih funkcija

Ove funkcije omogućavaju jednostavan prikaz konika, no mogu dati i infleksije, što kvadratne razlomljene funkcije nisu mogle. Kako je riječ o kubnim funkcijama, funkcijske ovisnosti svih koordinata u homogenom prostoru biti će izražene

kubnim polinomom:

$$T_{Kh,1} = a_1 \cdot t^3 + b_1 \cdot t^2 + c_1 \cdot t + d_1 \tag{7.25}$$
$$T_{Kh,2} = a_2 \cdot t^3 + b_2 \cdot t^2 + c_2 \cdot t + d_2 \tag{7.26}$$
$$T_{Kh,3} = a_3 \cdot t^3 + b_3 \cdot t^2 + c_3 \cdot t + d_3 \tag{7.27}$$
$$T_{Kh,h} = a \cdot t^3 + b \cdot t^2 + c \cdot t + d \tag{7.28}$$

Povratkom u radni prostor dobiva se:

$$T_{K,1} = \frac{a_1 \cdot t^3 + b_1 \cdot t^2 + c_1 \cdot t + d_1}{a \cdot t^3 + b \cdot t^2 + c \cdot t + d}$$
$$T_{K,2} = \frac{a_2 \cdot t^3 + b_2 \cdot t^2 + c_2 \cdot t + d_2}{a \cdot t^3 + b \cdot t^2 + c \cdot t + d}$$
$$T_{K,3} = \frac{a_3 \cdot t^3 + b_3 \cdot t^2 + c_3 \cdot t + d_3}{a \cdot t^3 + b \cdot t^2 + c \cdot t + d}$$

Jednadžbe dane za homogenih prostor opet vode na matrični zapis:

$$T_{Kh} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \\ d_1 & d_2 & d_3 & d \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \mathbf{A}. \tag{7.29}$$

gdje je $\mathbf{A}$ karakteristična matrica krivulje.

7.5.4 Parametarske derivacije u homogenom prostoru

U ovom su slučaju sve funkcijske ovisnosti kubne.

$$T_{Kh,1} = a_1 \cdot t^3 + b_1 \cdot t^2 + c_1 \cdot t + d_1$$
$$T_{Kh,2} = a_2 \cdot t^3 + b_2 \cdot t^2 + c_2 \cdot t + d_2$$
$$T_{Kh,3} = a_3 \cdot t^3 + b_3 \cdot t^2 + c_3 \cdot t + d_3$$
$$T_{Kh,h} = a \cdot t^3 + b \cdot t^2 + c \cdot t + d$$

Deriviranjem po parametru t dobiva se:

$$\frac{dT_{Kh,1}}{dt} = a_1 \cdot 3t^2 + b_1 \cdot 2t + c_1$$
$$\frac{dT_{Kh,2}}{dt} = a_2 \cdot 2t^2 + b_2 \cdot 2t + c_2$$
$$\frac{dT_{Kh,3}}{dt} = a_3 \cdot 2t^2 + b_3 \cdot 2t + c_3$$
$$\frac{dT_{Kh,h}}{dt} = a \cdot 2t^2 + b \cdot 2t + c$$

što se matrično može zapisati kao:

$$T'_{Kh} = \begin{bmatrix} T'_{Kh,1} & T'_{Kh,2} & T'_{Kh,3} & T'_{Kh,h} \end{bmatrix} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot \mathbf{A}. \tag{7.30}$$

Više derivacije iznose:

$$T''_{Kh} = \begin{bmatrix} 6t & 2 & 0 & 0 \end{bmatrix} \cdot \mathbf{A}.$$

$$T'''_{Kh} = \begin{bmatrix} 6 & 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{A}.$$

$$T^{(n)}_{Kh} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{A}, \quad n > 3.$$

7.5.5 Veza između parametarskih derivacija u radnom i homogenom prostoru

Prilikom zadavanja krivulja, kada se radi s derivacijama, najčešće se specificiraju derivacije u radnom prostoru: korisnik zadaje nagib odnosno tangencijalni vektor u pojedinim točkama krivulje koji odgovara gradijentu u toj točki u radnom prostoru. Međutim, kada radimo s razlomljenim krivuljama, radimo u homogenom prostoru. Stoga je potrebno pogledati kakav je odnos, odnosno koja je veza između parametarskih derivacija u radnom i homogenom prostoru.

Veza između radnih i homogenih koordinata već nam je poznata.

$$T_{K,i} = \frac{T_{Kh,i}}{T_{Kh,h}}, i \in 1, 2, 3$$

Pri tome oznakom T_K označavamo točku u radnom prostoru, a oznakom T_{Kh} točku u homogenom prostoru. Ovu ovisnost možemo zapisati matrično kako slijedi.

$$\begin{aligned} T_{Kh} &= \begin{bmatrix} T_{K,1} \cdot T_{Kh,h} & T_{K,2} \cdot T_{Kh,h} & T_{K,3} \cdot T_{Kh,h} & T_{Kh,h} \end{bmatrix} \\ &= T_{Kh,h} \cdot \begin{bmatrix} T_{K,1} & T_{K,2} & T_{K,3} & 1 \end{bmatrix} \end{aligned} \tag{7.31}$$

Doista, točki u radnom prostoru s koordinatama $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ u homogenom prostoru odgovara točka $\begin{bmatrix} 1 & 2 & 3 & 1 \end{bmatrix}$, točka $\begin{bmatrix} 2 & 4 & 6 & 2 \end{bmatrix}$, točka $\begin{bmatrix} 3 & 6 & 9 & 3 \end{bmatrix}$ kao i još bezbroj drugih.

Pretpostavimo da su nam poznate parametarske derivacije koordinata u 3D radnom prostoru – korisnik je zadao komponente tangencijalnog vektora, odnosno znamo:

$$T'_{K,1} = \frac{dT_{K,1}}{dt}, \quad T'_{K,2} = \frac{dT_{K,2}}{dt}, \quad T'_{K,3} = \frac{dT_{K,3}}{dt}.$$

Pri tome je potrebno prisjetiti se da su $T_{K,1}$, $T_{K,2}$ i $T_{K,3}$ funkcije koje ovise o parametru t pa zadavanjem različitih vrijednosti parametra t dobivamo pojedine točke tako definirane krivulje.

Vezu između koordinata točke u radnom i homogenom prostoru dali smo izrazom (7.31), pa za pojedine komponente možemo očitati:

$$T_{Kh,1} = T_{K,1} \cdot T_{Kh,h}, \quad T_{Kh,2} = T_{K,2} \cdot T_{Kh,h}, \quad T_{Kh,3} = T_{K,3} \cdot T_{Kh,h}.$$

Oznakama $T_{Kh,1}$, $T_{Kh,2}$, $T_{Kh,3}$ i $T_{Kh,h}$ označili smo komponente točke u homogenom prostoru. Svaka od tih komponenata također je funkcija ovisna o parametru t i te funkcijske ovisnosti znamo. Primjerice, radimo li s kubnim razlomljenim funkcijama, funkcijske ovisnosti homogenih komponenata $T_{Kh,1}$, $T_{Kh,2}$, $T_{Kh,3}$ i $T_{Kh,h}$ o parametru t dane su izrazima (7.25)-(7.28).

Kako su prve tri komponente u prethodnim izrazima definirane kao umnožak dviju funkcija (funkcije koja opisuje ovisnost komponente u radnom prostoru o parametru t i funkcije koja opisuje kako se mijenja homogeni parametar u homogenom prostoru u ovisnosti o parametru t), parametarske derivacije komponenata u homogenom prostoru mogu se primjenom pravila derivacije umnoška prikazati kako slijedi.

$$T'_{Kh,1} = \left(T_{K,1} \cdot T_{Kh,h}\right)' = T'_{K,1} \cdot T_{Kh,h} + T_{K,1} \cdot T'_{Kh,h} \tag{7.32}$$

$$T'_{Kh,2} = \left(T_{K,2} \cdot T_{Kh,h}\right)' = T'_{K,2} \cdot T_{Kh,h} + T_{K,2} \cdot T'_{Kh,h} \tag{7.33}$$

$$T'_{Kh,3} = \left(T_{K,3} \cdot T_{Kh,h}\right)' = T'_{K,3} \cdot T_{Kh,h} + T_{K,3} \cdot T'_{Kh,h} \tag{7.34}$$

Zapisano matrično dobije se:

$$\begin{aligned}
T'_{Kh} &= \begin{bmatrix} T'_{Kh,1} & T'_{Kh,2} & T'_{Kh,3} & T'_{Kh,h} \end{bmatrix} \\
&= \begin{bmatrix} T'_{K,1} \cdot T_{Kh,h} + T_{K,1} \cdot T'_{Kh,h} & T'_{K,2} \cdot T_{Kh,h} + T_{K,2} \cdot T'_{Kh,h} & T'_{K,3} \cdot T_{Kh,h} + T_{K,3} \cdot T'_{Kh,h} & T'_{Kh,h} \end{bmatrix} \\
&= \begin{bmatrix} T'_{Kh,h} & T_{Kh,h} \end{bmatrix} \cdot \begin{bmatrix} T_{K,1} & T_{K,2} & T_{K,3} & 1 \\ T'_{K,1} & T'_{K,2} & T'_{K,3} & 0 \end{bmatrix} \\
&= \begin{bmatrix} T'_{Kh,h} & T_{Kh,h} \end{bmatrix} \cdot \begin{bmatrix} T_K & 1 \\ T'_K & 0 \end{bmatrix}
\end{aligned}$$

Na ovaj način dobili smo direktnu vezu između traženih parametarskih derivacija u homogenom prostoru i poznatih parametarskih derivacija u radnom prostoru. Uočimo sada da prikazana veza ne omogućava jednoznačno određivanje parametarskih derivacija u homogenom prostoru ako su poznate parametarske derivacije u radnom prostoru. Pogledajmo to na primjeru. Neka je poznato da krivulja u radnom 3D prostoru prolazi kroz točku $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ i da je u toj točki tangencijalni vektor jednak $\begin{bmatrix} 2 & -1 & 1 \end{bmatrix}$. Uvrštavanjem podataka u izraz:

$$T'_{Kh} = \begin{bmatrix} T'_{Kh,h} & T_{Kh,h} \end{bmatrix} \cdot \begin{bmatrix} T_{K,1} & T_{K,2} & T_{K,3} & 1 \\ T'_{K,1} & T'_{K,2} & T'_{K,3} & 0 \end{bmatrix}$$

dobivamo:

$$\begin{aligned} T'_{Kh} &= \begin{bmatrix} T'_{Kh,h} & T_{Kh,h} \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 & 1 \\ 2 & -1 & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} T'_{Kh,h} + 2T_{Kh,h} & 2T'_{Kh,h} - T_{Kh,h} & 3T'_{Kh,h} + T_{Kh,h} & T'_{Kh,h} \end{bmatrix}. \end{aligned}$$

Rezultat pokazuje da su nam ostala još dva stupnja slobode: parametarsku derivaciju u homogenom prostoru moći ćemo jednoznačno odrediti tek kada još na neki način fiksiramo vrijednosti $T'_{Kh,h}$ i $T_{Kh,h}$. Primjerice, odaberemo li da je derivacija funkcije koja opisuje ovisnost homogenog parametra o parametru t u promatranoj točki jednaka 1 a da je iznos homogenog parametra u toj istoj točki jednak 2, dobit ćemo konačan rezultat:

$$\begin{aligned} T'_{Kh} &= \begin{bmatrix} 1+2\cdot 2 & 2\cdot 1-2 & 3\cdot 1+2 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 5 & 0 & 5 & 1 \end{bmatrix}. \end{aligned}$$

Razlog za nepotpunost specificiranja parametarskih derivacija u homogenom prostoru ako su poznate samo parametarske derivacije u radnom prostoru i točka kroz koju krivulja prolazi (a u kojoj je zadana derivacija) leži u činjenici da u homogenom prostoru imamo jedan stupanj slobode više: u nedavnom primjeru podsjetili smo se da jednoj točki u radnom prostoru odgovara beskonačno mnogo točaka u homogenom prostoru. Taj stupanj slobode više rezultira vezom između parametarskih derivacija u radnom i homogenom prostoru koja kao slobodne parametre ima vrijednosti $T'_{Kh,h}$ i $T_{Kh,h}$. Tek kada na neki način odredimo i te vrijednosti, vrijednost parametarske derivacije u homogenom prostoru postat će jedinstvena.

Druga parametarska derivacija u homogenom prostoru dobije se deriviranjem prve parametarske derivacije u homogenom prostoru; nakon što se deriviraju izrazi i nakon ubacivanja u matricu dobiva se:

$$\begin{aligned} T''_{Kh} &= \begin{bmatrix} T''_{Kh,1} & T''_{Kh,2} & T''_{Kh,3} & T''_{Kh,h} \end{bmatrix} \\ &= \begin{bmatrix} (T'_{K,1}\cdot T_{Kh,h}+T_{K,1}\cdot T'_{Kh,h})' & (T'_{K,2}\cdot T_{Kh,h}+T_{K,2}\cdot T'_{Kh,h})' & (T'_{K,3}\cdot T_{Kh,h}+T_{K,3}\cdot T'_{Kh,h})' & T''_{Kh,h} \end{bmatrix} \\ &= \begin{bmatrix} T''_{Kh,h} & 2\cdot T'_{Kh,h} & T_{Kh,h} \end{bmatrix} \cdot \begin{bmatrix} T_{K,1} & T_{K,2} & T_{K,3} & 1 \\ T'_{K,1} & T'_{K,2} & T'_{K,3} & 0 \\ T''_{K,1} & T''_{K,2} & T''_{K,3} & 0 \end{bmatrix} \\ &= \begin{bmatrix} T''_{Kh,h} & 2\cdot T'_{Kh,h} & T_{Kh,h} \end{bmatrix} \cdot \begin{bmatrix} T_K & 1 \\ T'_K & 0 \\ T''_K & 0 \end{bmatrix} \end{aligned}$$

I više derivacije mogu se izvesti na sličan način.

7.5.6 Primjer

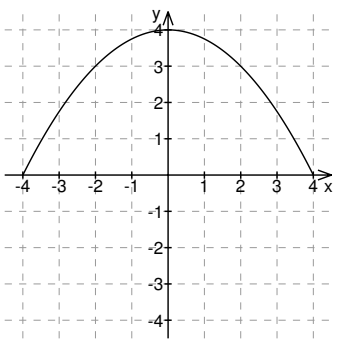
Pomoću razlomljene kvadratne funkcije želimo odrediti krivulju koja će prolaziti sljedećim točkama:

- za $t = t_1 = 0$ kroz točku [R 0 0 1],
- za $t = t_2 = 0.5$ kroz točku [0 R 0 1] te
- za $t = t_2 = 1$ kroz točku [-R 0 0 1].

Na prvi pogled popis točaka odgovara kružnici, no pogledajmo što ćemo dobiti. Izračunajmo matricu $\mathbf{K}$ temeljem izračuna koji smo već prethodno napravili.

$$\mathbf{K} = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} R & 0 & 0 & 1 \\ 0 & R & 0 & 1 \\ -R & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -4R & 0 & 0 \\ -2R & 4R & 0 & 0 \\ R & 0 & 0 & 1 \end{bmatrix}.$$

Uvrstimo li da je $R = 4$, prikaz upravo izvedene krivulje dan je na slici 7.7. Na slici jasno vidimo da se radi o paraboli.



Slika 7.7: Krivulja nastala naivnim pokušajem dobivanja kružnice. Umjesto kružnice dobili smo parabolu.

Svaka točka ove naše krivulje određena je jednostavnom relacijom:

$$T_{Kh} = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \mathbf{K} = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & -4R & 0 & 0 \\ -2R & 4R & 0 & 0 \\ R & 0 & 0 & 1 \end{bmatrix}.$$

Ako želimo prikazati kružnicu, poslužit ćemo se parametarskim oblikom jednadžbe kružnice. Znamo da kod kružnice vrijedi veza:

$$x = R \cdot \cos\phi$$

$$y = R \cdot \sin\phi$$

Znamo i da postoji veza između sinusa i kosinusa:

$$t = \mathrm{tg}\left(\frac{\phi}{2}\right) \quad x = R \cdot \frac{1-t^2}{1+t^2} \quad y = R \cdot \frac{2t}{1+t^2}$$

Kako je nazivnik isti i za x i za y-koordinatu, njega ćemo uzeti kao funkciju koja opisuje homogeni parametar. Tada možemo pisati:

$$T_{Kh,x} = -R \cdot t^2 + R$$

$$T_{Kh,y} = 2R \cdot t$$

$$T_{Kh,z} = 0$$

$$T_{Kh,h} = t^2 + 1$$

iz čega direktno slijedi matrica $\mathbf{K}$:

$$\mathbf{K} = \begin{bmatrix} -R & 0 & 0 & 1 \\ 0 & 2R & 0 & 0 \\ R & 0 & 0 & 1 \end{bmatrix}$$

pa krivulju možemo zapisati kao:

$$T_{Kh} = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -R & 0 & 0 & 1 \\ 0 & 2R & 0 & 0 \\ R & 0 & 0 & 1 \end{bmatrix}.$$

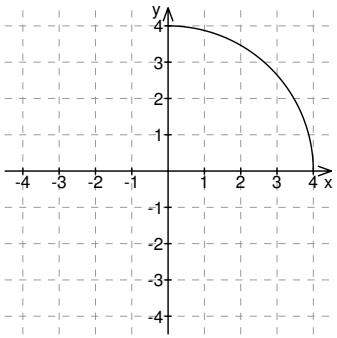
Prikaz krivulje koja se dobije uz ovakvu matricu te uz $R = 4$ dan je na slici 7.8. Ako se parametar mijenja od 0 do 1, dobivaju se točke krivulje koje leže u prvom kvadrantu.

7.5.7 Određivanje kubnog segmenta krivulje određenog rubnim uvjetima

U nastavku ćemo odrediti matricu $\mathbf{A}$ za kubnu razlomljenu krivulju koja je zadana pomoću početne i završne točke u radnom prostoru, te prve derivacije u tim točkama (odnosno tangencijalnim vektorima u tim točkama u radnom prostoru); početna točka dobiva se za parametar $t = t_0 = 0$ dok se završna točka dobiva za parametar $t = t_1 = 1$. Dakle, poznato nam je sljedeće:

$$T(0) = \begin{bmatrix} T_1(0) & T_2(0) & T_3(0) \end{bmatrix},$$

$$T(1) = \begin{bmatrix} T_1(1) & T_2(1) & T_1(1) \end{bmatrix},$$



Slika 7.8: Segment kružnice.

$$T^{'}(0) = \left[\begin{array}{ccc} T_1^{'}(0) & T_2^{'}(0) & T_3^{'}(0) \end{array} \right],$$

$$T^{'}(1) = \left[\begin{array}{ccc} T_1^{'}(1) & T_2^{'}(1) & T_3^{'}(1) \end{array} \right].$$

Razmotrit ćemo matricu ξ čiji su retci koordinate prve i zadnje točke u homogenom prostoru te parametarske derivacije u istim točkama u homogenom prostoru. Neka $T_{h,i}(t)$ opisuje funkcijsku ovisnost i-te komponente točke u homogenom prostoru u ovisnosti o parametru t, a $T_{h,i}^{'}(t)$ funkcijsku ovisnost i-te komponente parametarske derivacije u homogenom prostoru o parametru t. Elementi matrice ξ tada su:

$$\xi = \begin{bmatrix} T_{h,1}(0) & T_{h,2}(0) & T_{h,3}(0) & T_{h,h}(0) \\ T_{h,1}(1) & T_{h,2}(1) & T_{h,3}(1) & T_{h,h}(1) \\ T_{h,1}^{'}(0) & T_{h,2}^{'}(0) & T_{h,3}^{'}(0) & T_{h,h}^{'}(0) \\ T_{h,1}^{'}(1) & T_{h,2}^{'}(1) & T_{h,3}^{'}(1) & T_{h,h}^{'}(1) \end{bmatrix}. \tag{7.35}$$

Koordinate točaka za $t = 0$ i $t = 1$ u radnom prostoru imamo. Stoga za raspisivanje prva dva retka možemo koristiti izraz (7.31). Kako su nam poznate i derivacije u radnom prostoru u točkama koje dobijemo za $t = 0$ i $t = 1$, treći i četvrti redak možemo raspisati koristeći izraze (7.32)-(7.34). Dobivamo sljedeće.

$$\xi = \begin{bmatrix} T_1(0) \cdot T_{h,h}(0) & T_2(0) \cdot T_{h,h}(0) & T_3(0) \cdot T_{h,h}(0) & T_{h,h}(0) \\ T_1(1) \cdot T_{h,h}(1) & T_2(1) \cdot T_{h,h}(1) & T_3(1) \cdot T_{h,h}(1) & T_{h,h}(1) \\ T_1^{'}(0) \cdot T_{h,h}(0) + T_1(0) \cdot T_{h,h}^{'}(0) & T_2^{'}(0) \cdot T_{h,h}(0) + T_2(0) \cdot T_{h,h}^{'}(0) & T_3^{'}(0) \cdot T_{h,h}(0) + T_3(0) \cdot T_{h,h}^{'}(0) & T_{h,h}^{'}(0) \\ T_1^{'}(1) \cdot T_{h,h}(1) + T_1(1) \cdot T_{h,h}^{'}(1) & T_2^{'}(1) \cdot T_{h,h}(1) + T_2(1) \cdot T_{h,h}^{'}(1) & T_3^{'}(1) \cdot T_{h,h}(1) + T_3(1) \cdot T_{h,h}^{'}(1) & T_{h,h}^{'}(1) \end{bmatrix}$$

Matricu ξ možemo napisati kao umnožak dviju matrica: matrice u kojoj se nalaze samo vrijednosti homogenog parametra i njegovih derivacija te matrice u kojoj se nalaze samo zadane točke i njihove derivacije u radnom prostoru. Ovaj rastav

prikazan je u nastavku.

$$\xi = \begin{bmatrix} T_{h,h}(0) & 0 & 0 & 0 \\ 0 & T_{h,h}(1) & 0 & 0 \\ T'_{h,h}(0) & 0 & T_{h,h}(0) & 0 \\ 0 & T'_{h,h}(1) & 0 & T_{h,h}(1) \end{bmatrix} \cdot \begin{bmatrix} T_1(0) & T_2(0) & T_3(0) & 1 \\ T_1(1) & T_2(1) & T_3(1) & 1 \\ T'_1(0) & T'_2(0) & T'_3(0) & 0 \\ T'_1(1) & T'_2(1) & T'_3(1) & 0 \end{bmatrix} \tag{7.36}$$

$$= \mathbf{H} \cdot \mathbf{V}. \tag{7.37}$$

Matrica $\mathbf{H}$ u ovom se raspisu pojavila zbog veze između parametarskih derivacija u radnom i homogenom prostoru i činjenice da parametarske derivacije u homogenom prostoru nisu jednoznačno određene samo poznavanjem parametarskih derivacija u radnom prostoru – o tome smo već prethodno diskutirali. Matrica $\mathbf{V}$ u prva dva retka sadrži koordinate početne i krajnje točke u radnom prostoru a u druga dva retka vrijednosti parametarskih derivacija u tim točkama, također u radnom prostoru. Četvrti stupac te matrice fiksno je postavljen na $\begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}^T$ kako bi vrijedila jednakost $\xi = \mathbf{H} \cdot \mathbf{V}$.

Osim na prikazani način, matricu ξ definiranu izrazom (7.35) možemo raspisati na još jedan način, a to je uporabom izraza (7.29) za prva dva retka te izraza (7.30) za druga dva retka:

$$\xi = \begin{bmatrix} 0^3 & 0^2 & 0^1 & 1 \\ 1^3 & 1^2 & 1^1 & 1 \\ 3 \cdot 0^2 & 2 \cdot 0 & 1 & 0 \\ 3 \cdot 1^2 & 2 \cdot 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \\ d_1 & d_2 & d_3 & d \end{bmatrix} \tag{7.38}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \\ d_1 & d_2 & d_3 & d \end{bmatrix} \tag{7.39}$$

$$= \mathbf{B} \cdot \mathbf{A} \tag{7.40}$$

Matrica $\mathbf{B}$ je matrica parametara dok je matrica $\mathbf{A}$ matrica koja određuje krivulju. Kako izrazi (7.37) i (7.40) oba predstavljaju raspis iste matrice ξ, možemo ih izjednačiti pa vrijedi:

$$\mathbf{H} \cdot \mathbf{V} = \mathbf{B} \cdot \mathbf{A}.$$

Odavde možemo odrediti matricu $\mathbf{A}$:

$$\mathbf{A} = \mathbf{B}^{-1} \cdot \mathbf{H} \cdot \mathbf{V} = \mathbf{M} \cdot \mathbf{H} \cdot \mathbf{V}.$$

Matrica $\mathbf{M}$ naziva se *univerzalna transformacijska matrica* i jednaka je inverzu matrice $\mathbf{B}$. Kako smo matricu $\mathbf{B}$ već odredili, matrica $\mathbf{M}$ iznosi:

$$\mathbf{M} = \mathbf{B}^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Ako su nam poznate točke krivulje i derivacije u njima (matrica $\mathbf{V}$), te homogeni parametri i njihove derivacije za obje točke ($\mathbf{H}$), na temelju izvedene relacije možemo jednoznačno odrediti matricu $\mathbf{A}$ kao umnožak univerzalne transformacijske matrice, matrice homogenog parametra i matrice točaka.

7.5.8 Hermitova krivulja

Hermitova krivulja specijalan je slučaj kubne razlomljene krivulje – to je kubna razlomljena krivulja koja je zadana početnom i završnom točkom te tangencijalnim vektorima u početnoj i završnoj točki. Kod Hermitove krivulje funkcija po kojoj se mijenja homogeni parametar nije kubna, već je konstanta i iznosi 1. Drugim riječima, za točku krivulje T_{Kh} u homogenom prostoru vrijedi:

$$T_{Kh,1} = P_3(t) \quad T_{Kh,2} = Q_3(t) \quad T_{Kh,3} = R_3(t) \quad T_{Kh,h} = 1$$

gdje su P_3, Q_3 i R_3 polinomi trećeg stupnja po parametru t.

Iskoristimo sada izraz za matricu $\mathbf{A}$:

$$\mathbf{A} = \mathbf{M} \cdot \mathbf{H} \cdot \mathbf{V}.$$

Matrica $\mathbf{M}$ je poznata, matricu $\mathbf{V}$ lako odredimo ako znamo dvije točke i derivacije u njima. Ostaje nam još matrica $\mathbf{H}$. Ona u sebi sadrži homogene parametre (koji su u ovom slučaju za sve točke jednaki 1 – Hermitova krivulja), i njihove derivacije. Kako su funkcije homogenih parametara konstante, njihova je derivacije 0. Time matrica $\mathbf{H}$ postaje jedinična.

$$\begin{bmatrix} T_{h,h}(0) & 0 & 0 & 0 \\ 0 & T_{h,h}(1) & 0 & 0 \\ T'_{h,h}(0) & 0 & T_{h,h}(0) & 0 \\ 0 & T'_{h,h}(1) & 0 & T_{h,h}(1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{I}$$

Između Hermitove krivulje i Bézierove krivulje za kubni slučaj postoji veza. Već smo rekli da Hermitova krivulja traži da je vrijednost homogenog parametra u svim točkama jednaka jedan. To znači da su prve tri homogene koordinate jednake radnim koordinatama. U tom je slučaju za prelazak iz homogenih u radne koordinate dovoljno u izrazu:

$$\mathbf{A} = \mathbf{M} \cdot \mathbf{V}.$$

matricu $\mathbf{V}$ zamijeniti samo s njena prva tri stupca. Dobije se:

$$\vec{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \mathbf{M} \cdot \begin{bmatrix} \vec{p}_0 \\ \vec{p}_1 \\ \vec{p}_0{}' \\ \vec{p}_1{}' \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \vec{r}_0 \\ \vec{r}_1 \\ \vec{r}_2 \\ \vec{r}_3 \end{bmatrix}$$

Lijeva strana jednadžbe dolazi od Hermitove krivulje; desna strana jednadžbe je Bézierova krivulja. Zaključak je su to iste krivulje, odnosno dva različita načina zapisa iste krivulje. Pri tome se Hermitova krivulja zadaje početnom i krajnjom točkom i derivacijama u njima, dok se Bézierova krivulja zadaje preko četiri točke kontrolnog poligona (početnom, dvijema kontrolnima i završnom).

7.5.9 Određivanje matrice A - primjer

Neka su početna i konačna točka segmenta krivulje u radnom prostoru:

$$T(0) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix},$$

$$T(1) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}.$$

Neka su parametarske derivacije u radnom prostoru u početnoj i konačnoj točki (odnosno tangencijalni vektori):

$$T'(0) = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix},$$

$$T'(1) = \begin{bmatrix} 1 & -1 & 0 \end{bmatrix}.$$

Treba odrediti matricu $\mathbf{A}$. Za matricu $\mathbf{A}$ smo izveli izraz:

$$\mathbf{A} = \mathbf{M} \cdot \mathbf{H} \cdot \mathbf{V}.$$

$\mathbf{M}$ je poznata:

$$\mathbf{M} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

$\mathbf{H}$ možemo odrediti:

$$\mathbf{H} = \begin{bmatrix} T_{h,h}(0) & 0 & 0 & 0 \\ 0 & T_{h,h}(1) & 0 & 0 \\ T'_{h,h}(0) & 0 & T_{h,h}(0) & 0 \\ 0 & T'_{h,h}(1) & 0 & T_{h,h}(1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & 1 & 0 \\ 0 & b & 0 & 1 \end{bmatrix}.$$

Vrijednosti koje homogeni parametar poprima u početnoj i konačnoj točki krivulje u homogenom prostoru odabrali smo da bude 1. Ono što je još ostalo slobodno su vrijednosti derivacija funkcije homogenog parametra u početnoj i krajnjoj točki: te smo vrijednosti supstituirali simbolima a i b i te ćemo vrijednosti izračunati u nastavku.

Matrica $\mathbf{V}$ također je poznata:

$$\mathbf{V} = \begin{bmatrix} T(0) & 1 \\ T(1) & 1 \\ T'(0) & 0 \\ T'(1) & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}.$$

Množenjem ove tri matrice dobiva se:

$$\mathbf{A} = \mathbf{M} \cdot \mathbf{H} \cdot \mathbf{V} = \begin{bmatrix} b & 0 & 0 & a+b \\ -b & -1 & 0 & -2a-b \\ 1 & 1 & 0 & a \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Da bismo odredili matricu $\mathbf{A}$ jednoznačno, vidimo da nam nedostaje još jedan uvjet. Tako možemo tražiti sljedeće: neka krivulja za $t = t_2 = \frac{1}{2}$ prođe kroz točku radnog prostora (1/2 1/2 0). Ovaj podatak pomoći će nam da odredimo traženu matricu. No računu treba pristupiti oprezno. Zadali smo točku u radnom prostoru i tražimo da krivulja prođe kroz nju. Jednadžba krivulje, međutim, daje sve točke u homogenom prostoru. I to za svaku točku radnog prostora daje samo jednu točku homogenog prostora (iako tih točaka za svaku točku radnog prostora ima beskonačno). Ukoliko ovu činjenicu ignoriramo, mogli bismo reći sljedeće: točka leži na krivulji, pa vrijedi:

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 1 \end{bmatrix} = \begin{bmatrix} t_2^3 & t_2^2 & t_2 & 1 \end{bmatrix} \cdot \mathbf{A}$$
$$= \begin{bmatrix} \frac{1}{8} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix} \cdot \begin{bmatrix} b & 0 & 0 & a+b \\ -b & -1 & 0 & -2a-b \\ 1 & 1 & 0 & a \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ovaj sustav predstavlja četiri jednadžbe s četiri nepoznanice, i pri tome je nerješiv (zapravo, rješiv je: rješenje ne postoji). Naime, već izjednačavanjem po drugoj komponenti dobiva se:

$$\frac{1}{2} = -\frac{1}{4} + \frac{1}{2} = \frac{1}{4}$$

što je očito besmisleno. Jedno od loših tumačenja ovog rješenja je da krivulja jednostavno ne može proći kroz tu točku uz zadane parametre. I ovo je mjesto na

kojem treba razmisliti. Krivulja (očito) ne može proći kroz tu točku homogenog prostora, no može li možda proći kroz neku drugu točku homogenog prostora a da pri tome prolazi kroz istu točku radnog prostora? Odgovor na ovo je potvrdan! Naime, sustav je ispravno napisati i rješavati po komponentama samo ukoliko su sve komponente u potpunosti nezavisne – što ovdje nisu. Da bismo dobili naše rješenje, potrebno je primijeniti zavisnosti koje znamo i raditi jednačenje po stvarno-nezavisnim komponentama: komponentama radnog prostora! Tada ćemo dobiti sustave:

$$\frac{\frac{1}{8}b - \frac{1}{4}b + \frac{1}{2}}{\frac{1}{8}(a+b) + \frac{1}{4}(-2a-b) + \frac{1}{2}a + 1} = \frac{\frac{1}{2}}{1}$$

$$\frac{-\frac{1}{4} + \frac{1}{2}}{\frac{1}{8}(a+b) + \frac{1}{4}(-2a-b) + \frac{1}{2}a + 1} = \frac{\frac{1}{2}}{1}$$

$$\frac{0}{\frac{1}{8}(a+b) + \frac{1}{4}(-2a-b) + \frac{1}{2}a + 1} = \frac{0}{1}$$

Sada umjesto četiri "nezavisne" jednadžbe imamo samo tri, i to rješive. Treća jednadžba je identitet pa otpada. Prve dvije daju:

$$\frac{-b+4}{a-b+8} = \frac{1}{2} \qquad \frac{2}{a-b+8} = \frac{1}{2} \qquad \Rightarrow \qquad a = -2,\ b = 2.$$

Tražena matrica $\mathbf{A}$ glasi:

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ -2 & -1 & 0 & 2 \\ 1 & 1 & 0 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Ostalo nam je još da pogledamo kroz koju je točku homogenog prostora krivulja prošla za traženu točku radnog prostora:

$$T_{Kh} = \begin{bmatrix} \left(\frac{1}{2}\right)^3 & \left(\frac{1}{2}\right)^2 & \frac{1}{2} & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 & 0 \\ -2 & -1 & 0 & 2 \\ 1 & 1 & 0 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{2} \end{bmatrix}$$

a odgovarajuća točka radnog prostora je doista tražena:

$$T_K = \begin{bmatrix} \frac{\frac{1}{4}}{\frac{1}{2}} & \frac{\frac{1}{4}}{\frac{1}{2}} & \frac{0}{\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}.$$

7.6 Veza između krivulja

Na kraju pregleda prethodno obrađenih krivulja spomenimo još da iste možemo koristiti ako je na temelju niza poznatih diskretnih vrijednosti potrebno napraviti kontinuiranu interpolaciju vrijednosti. Pri tome zahtjev može biti da krivulja u poznatim točkama poprima jednake vrijednosti zadanima (tj. da ih interpolira) ili da nije nužno da poprima baš egzaktne vrijednosti već može u određenoj mjeri odstupati (pa ih aproksimira). Podaci pri tome mogu biti koordinate u prostoru, podaci o boji, o vremenskim trenutcima ili bilo koje vrijednosti koje su nam poznate u diskretnim trenucima, a nas zanima kontinuirana promjena promatranih vrijednosti. Najjednostavniji oblik interpolacije je linearna interpolacija i za opisane krivulje linearna interpolacija predstavlja najjednostavniji slučaj.

Zanimljivo je da povijesni razvoj računalne grafike započeo izradom krivulja u industriji plovila, vozila i aviona kao zahtjevnog postupka izračuna u kojem su računala našla jednu od prvih primjena. Različiti načini razmišljanja kojim su krenuli projektanti u postupku izrade krivulje kao što je Bézierov i De Casteljauov doveli su do potpuno identične krivulje. Pokazali smo da i treći pristup, odnosno Hermitova krivulja kod koje polazimo od dvije rubne točke krivulje i derivacija u njima opet možemo svesti na kubnu Bézierovu krivulju. Kako se sve navedene krivulje temelje na parametarskom prikazu polinomima, to i je nekako očekivano. Opisane krivulje u praksi imaju niz primjena, a već i najjednostavnije – kvadratne Bézierove krivulje – dovoljan su alat za izradu skalabilnih računalnih fontova, što ćemo i pokazati u primjeru u nastavku.

Posebno moramo istaknuti najjednostavnije krivulje koje nas okružuju, a to su kružnice odnosno općenito konike. Njih, osim parabole, ne možemo dobiti bez da koristimo razlomljene polinome. Zato smo pokazali kako razlomljenim polinomima dolazimo do sasvim jednostavne krivulje kao što je kružnica. Sve prethodno obrađene krivulje koriste se za predstavljanje pojedinačnih segmenata čijim se povezivanjem dobiva složenija krivulja koja se zove *B-spline*. B-spline je osnova *NURBS* (engl. *Non-Uniform Rational B-Spline*) krivulja i ploha koje su danas osnova bilo kojeg alata za geometrijsko modeliranje objekata.

7.7 Uporaba krivulja: TrueType fontovi

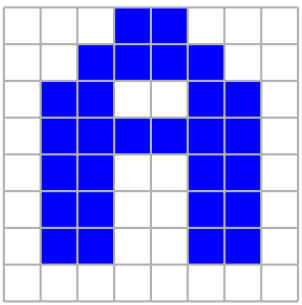
Krivulje se u računalnoj grafici koriste na mnoštvo mjesta i danas su u svakodnevnoj uporabi. Jedan od najočitijih primjera jest specificiranje fontova. U početcima razvoja računalne tehnologije za prikaz znakova su se koristili rasterski fontovi: u memoriji računala postojalo je mjesto gdje je za svaki znak bila smještena bitovna maska (primjerice: 8×8 slikovnih elemenata) koja je određivala koji će slikovni elementi biti upaljeni prilikom prikaza tog znaka. Kako su rezolucije ekrana u to vrijeme bile niske i kako je uobičajen način interakcije s

računalom bio kroz terminal fiksne veličine (npr. 25 redaka s 80 stupaca), veličina jednog znaka bila je fiksna pa je bilo smisleno potrebne podatke čuvati u memoriji na taj način.

Primjerice, nekada vrlo popularno računalo *Commodore 64* u tekstovnom načinu rada nudio je prikaz 25 redaka od kojih je svaki mogao pohraniti 40 znakova. Kako je grafička rezolucija bila 320×200, jedan znak na ekranu prikazivao se u polju od $320/40 = 8$ puta $200/25 = 8$, tj. 8×8 slikovnih elemenata. Kodna stranica koju je koristio *Commodore 64* definirala je 256 znakova (slova, interpunkcija, znamenke, grafički simboli i slično) pa je definicija maski ukupno zauzimala 8 okteta po znaku $\times$ 256 znakova što je ukupno 2 kB. Masku za prikaz slova *A* činilo je sljedećih 8 okteta: 24, 60, 102, 126, 102, 102, 102, 0. Zapišemo li te oktete binarno i posložimo jedan ispod drugoga, dobivamo uzorak slikovnih elemenata koji odgovara prikazu ovog slova.

```
00011000   ...11...
00111100   ..1111..
01100110   .11..11.
01111110   .111111.
01100110   .11..11.
01100110   .11..11.
01100110   .11..11.
00000000   ........
```

Na ekranu ovo bi rezultiralo prikazom danim na slici 7.9.

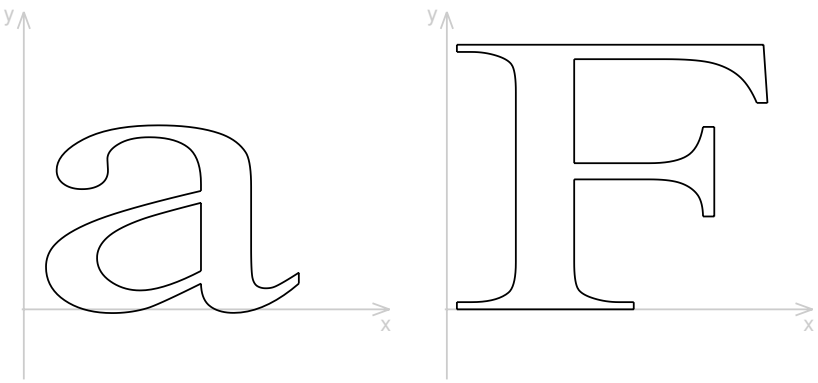


Slika 7.9: Prikaz slova *A* definiranog rasterskom maskom

Razvojem tehnologije prikaznih jedinica te pisača rasterski fontovi postali su neprikladni. Temeljni problem ovakvih fontova je loše skaliranje: ovisno o odabranoj skali, prikaz može postati vrlo nazubljen ili se pak, kod umanjivanja, mogu izgubiti važni detalji. Stoga su razvijeni fontovi koji "naputak" za generiranje slike znaka čuvaju u obliku vektorske grafike koja omogućava skaliranje bez gubitka kvalitete. Jedna od takvih tehnologija jesu *TrueType* fontovi.

7.7.1 Glyphovi

TrueType specifikacija definira *glyph* kao opis prikaza znaka. *Glyph* pri tome može biti jednostavan ili složen. Jednostavni *glyphovi* sastoje se od definicija jedne ili više zatvorenih kontura. Svaka je kontura pri tome sastavljena od niza segmenata ili kvadratnih Bézierovih krivulja. Primjer *glyphova* koji u fontu *Times New Roman* predstavljaju slova *a* i *F* dani su na slici 7.10.



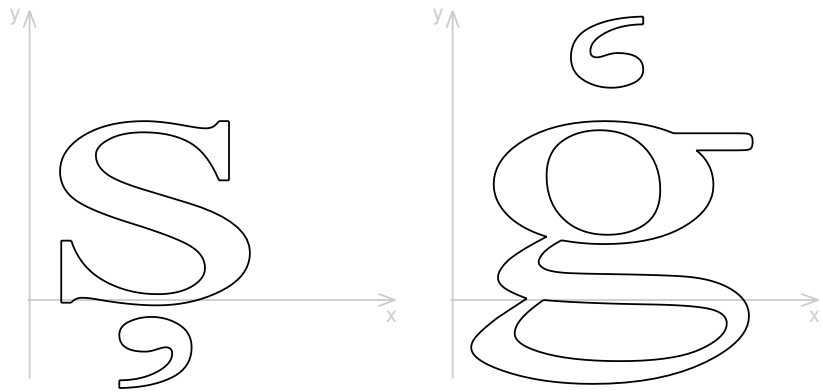
(a) *Glyph* za slovo *a* (b) *Glyph* za slovo *F*

Slika 7.10: *Glyphovi* za odabrana slova

Glyph za slovo *F* (slika 7.10b) sastoji se samo od jedne konture. *Glyph* za slovo *a* (slika 7.10a) sastoji se od dvije konture: jedna kontura opisuje vanjsku granicu slike slova dok druga definira prazninu koja se nalazi u trbuhu slova *a*.

Složeni *glyphovi* su *glyphovi* koji su sastavljeni od drugih *glyphova*. U `.ttf` datoteci za svaki složeni *glyph* definiran je popis *glyphova* od kojih je isti sastavljen te transformacije koje treba primijeniti na svaku komponentu kako bi se izgradio cjeloviti prikaz složenog *glypha*. Primjer dvaju složenih *glyphova* prikazan je na slici 7.11. *Glyph* prikazan na slici 7.11a sastavljen je od *glypha* za znak *s* te *glypha* koji je sličan znaku *zareza* i koji je iz ishodišta translatiran udesno i prema dolje. *Glyph* prikazan na slici 7.11b sastavljen je od *glypha* za znak *g* te istog onog *glypha* koji je sličan znaku *zareza* i koji je također prikazan na slici 7.11a. Međutim, prilikom sastavljanja konačnog prikaza ovaj je *glyph* sada najprije skaliran po x i y osi s faktorom -1 (čime je postignuto zrcaljenje oko x i y osi) i potom je translatiran gore i desno.

Za pozicioniranje *glyphova* koji su dijelovi složenih *glyphova*, specifikacija *TrueType* omogućava pohranu informacija temeljem kojih je moguće obaviti translaciju, uniformno skaliranje, zasebno skaliranje po x i y osima te u najopćenitijem slučaju proizvoljnu transformaciju koja se može opisati matricom 2×2 plus translacija. Iako specifikacija svaki od ovih slučajeva zapisuje na drugačiji način (kako



(a) (b)

Slika 7.11: Primjer složenih *glyphova*.

bi se uštedio prostor u `.ttf` datoteci), svaki od ovih slučajeva može se svesti na općenitu transformaciju u homogenom prostoru oblika:

$$\begin{bmatrix} s_x & a & 0 \\ b & s_y & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

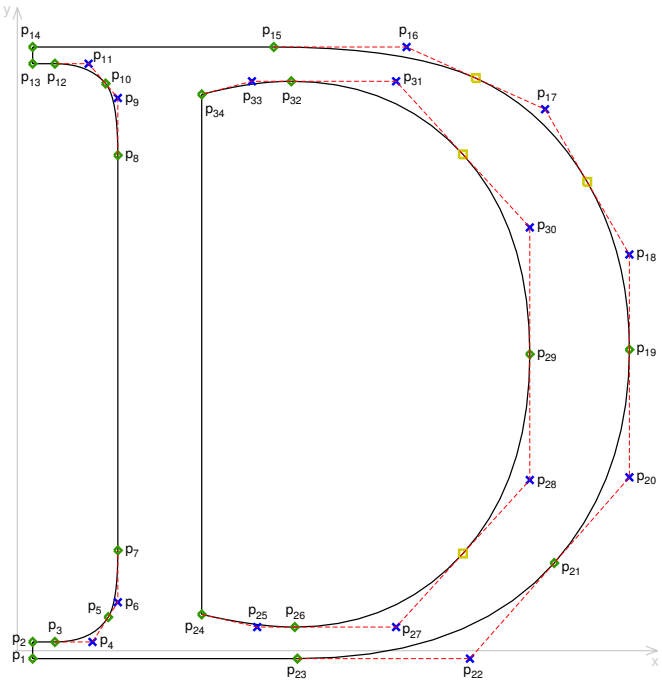
uz konvenciju množenja točke s transformacijskom matricom.

Razlog za uvođenje složenih *glyphova* također leži u uštedi na prostoru (isti se oblici ne definiraju više puta u datoteci) ali i postizanju konzistentnosti u prikazu: kako postoji više znakova odnosno simbola koji dijele određene elemente prikaza, takvi se elementi opišu jednom kao zaseban *glyph* i potom koriste na svim mjestima na kojima je to potrebno. U slučaju da je potrebno korigirati definiciju nekog *glypha*, s obzirom da se elementi prikaza dijele, korekciju je potrebno provesti samo na jednom mjestu i ona će automatski biti vidljiva uz sve *glyphove* koji koriste korigirani *glyph*.

7.7.2 Definiranje jednostavnog *glypha*

Jednostavan *glyph* definiran je jednom ili više kontura. Svaka kontura definirana je popisom točaka pri čemu svaka točka dodatno ima pridruženu zastavicu koja određuje vrstu točke. Postoje dvije vrste točaka: točke koje su na konturi (tzv. *on*-točke) te točke koje nisu na konturi (tzv. *off*-točke). Pogledajmo definiciju *glypha* koji je pridružen velikom slovu *D* u fontu *Times New Roman*, a koji je prikazan na slici 7.12.

Definicija ovog *glypha* sadrži 34 točke koje su na slici označene s p_1 do p_{34}. Točke p_1 do p_3 su *on*-točka, p_4 je *off*-točka, p_5 je *on*-točka, p_6 je *off*-točka i tako



Slika 7.12: *Glyph* pridružen slovu *D* s označenim točkama i njihovim statusima. Zeleni rombovi predstavljaju *on*-točke, plavi križići *off*-točke a žućkasti kvadratići *virtualne* točke

dalje. Statuse točaka redom od prve do zadnje možemo prikazati bit-vektorom:
`11101011010111100010101101001001O1`
gdje 1 označava *on*-točku a 0 označava *off*-točku. Prikazani *Glyph* sastoji se od dvije zatvorene konture. Prvu konturu čine točke p_1 do p_{23} dok drugu čine točke p_{24} do p_{34}. Status točaka važan je prilikom njihovog prikaza. Naime, specifikacija definira sljedeća pravila.

1. Slijed od dvije *on*-točke smatra se definicijom linijskog segmenta.

2. Slijed od *on*-točke, *off*-točke pa *on*-točke smatra se definicijom kvadratne aproksimacijske Bézierove krivulje za koju spomenute tri točke čine kontrolni poligon.

3. U slučaju da postoji slijed od dvije ili više *off*-točaka, između svake dvije *off*-točke umeće se virtualna *on*-točka koja se nalazi točno na polovištu spojnice tih dviju *off*-točaka. Primjerice, ako imamo pet točaka: $\vec{t}_1$, $\vec{t}_2$, $\vec{t}_3$, $\vec{t}_4$ i $\vec{t}_5$ od koji su samo $\vec{t}_1$ i $\vec{t}_5$ *on*-točke dok su ostale *off*-točke, time bi bile definirane tri kvadratne aproksimacijske Bézierove krivulje. Ako uve-

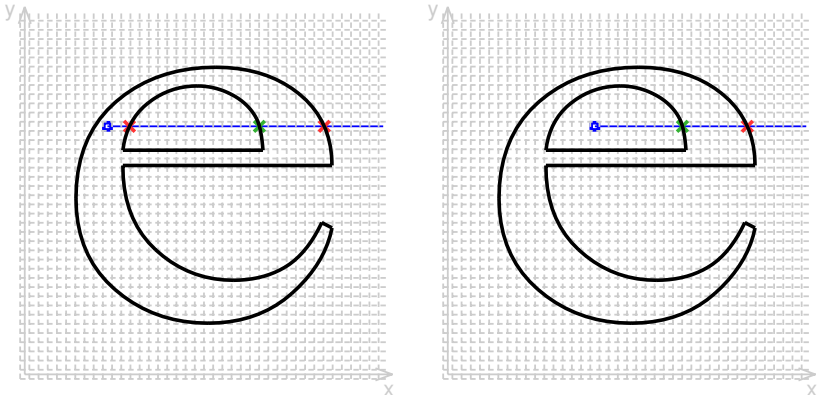
deno oznake $\vec{v}_{23} = \frac{\vec{t}_2+\vec{t}_3}{2}$ te $\vec{v}_{34} = \frac{\vec{t}_3+\vec{t}_4}{2}$, kontrolni poligon prve Bézierove krivulje bi bio određen točkama $(\vec{t}_1, \vec{t}_2, \vec{v}_{23})$, kontrolni poligon druge Bézierove krivulje bi bio određen točkama $(\vec{v}_{23}, \vec{t}_3, \vec{v}_{34})$ a kontrolni poligon treće Bézierove krivulje bio bi određen točkama $(\vec{v}_{34}, \vec{t}_4, \vec{t}_5)$.

Primjenu ovih pravila možemo vidjeti direktno na slici 7.12. Točke p_1 i p_2 su *on*-točke pa su spojene linijskim segmentom; p_2 i p_3 je opet slijed od dvije *on*-točke pa je i između njih povučena linija. Točke p_3, p_4 i p_5 čine slijed *on*-točke, *off*-točke i *on*-točke pa je nacrtana kvadratna Bézierova krivulja (a crvenom crtkanom linijom dodatno je prikazan i kontrolni poligon). Točke p_5, p_6 i p_7 također čine jednak slijed pa je i tu nacrtana kvadratna Bézierova krivulja. Točke p_7 i p_8 su obje *on*-točke pa su spojene linijom, itd. Situaciju opisanu pravilom (3) možemo vidjeti na primjeru slijeda točaka p_{15} do p_{19} od kojih su p_{15} i p_{19} *on*-točke a p_{16} do p_{18} *off*-točke. Na slijedovima p_{16}-p_{17} i p_{17}-p_{18} ubačene su dvije virtualne točke čime su ukupno definirane tri kvadratne aproksimacijske Bézierove krivulje. Jednostavniji primjer možemo vidjeti na unutarnjoj konturi gdje imamo slijed točaka p_{26} do p_{29} koji sadrži dvije unutarnje *off*-točke pa time definira dvije kvadratne aproksimacijske Bézierove krivulje. Uočite također da su konture zatvorene: zadnja točka prve konture je p_{23} koja je *on*-točka; kako ona s prvom točkom konture (p_1) čini slijed dvije *on*-točke, između njih je povučena linija. Da točka nije bila *on*-točka, primjenom prethodno navedenih pravila utvrdili bismo način na koji konturu potrebno zatvoriti.

7.7.3 Popunjavanje *glypha*

Prilikom prikaza na ekranu (ili prilikom ispisa na pisaču) slova uobičajeno nisu prikazana samo obrubom već je njihovo "tijelo" popunjeno. Da bismo pojasnili kako se radi popunjavanje, najprije je potrebno uočiti pravilnost koja mora biti poštivana prilikom definiranja kontura. Specifikacija zahtjeva da točke konture budu definirane tako da se prilikom obilaska konture od prve točke prema zadnjoj "tijelo" nalazi s desne strane. Pogledate li konture na slici 7.12, uočit ćete da je to poštivano. Vanjska kontura je definirana u smjeru kazaljke na satu kako bi prilikom "hodanja" po toj konturi unutrašnjost slova uvijek bila hodaču s desne strane. Unutrašnja kontura zadana je u smjeru suprotnom od smjera kazaljke na satu iz istog razloga. Način utvrđivanja smjera iz kojeg zraka probada linijski segment odnosno segment kvadratne aproksimacijske Bézierove krivulje (ulazi li zraka s lijeve strane a izlazi s desne ili obratno) opisan je u dodatku B u potpoglavlju B.1.

Postupak popunjavanja unutrašnjosti slova može se raditi na više načina. Jedan od načina koji se koristi jest pristup kod kojeg se za svaki slikovni element ispuca zraka (bilo vodoravna, bilo okomita) te se broji koliko puta ta zraka presječe konturu i kako. Slika 7.13 ilustrira postupak za dva slikovna elementa.



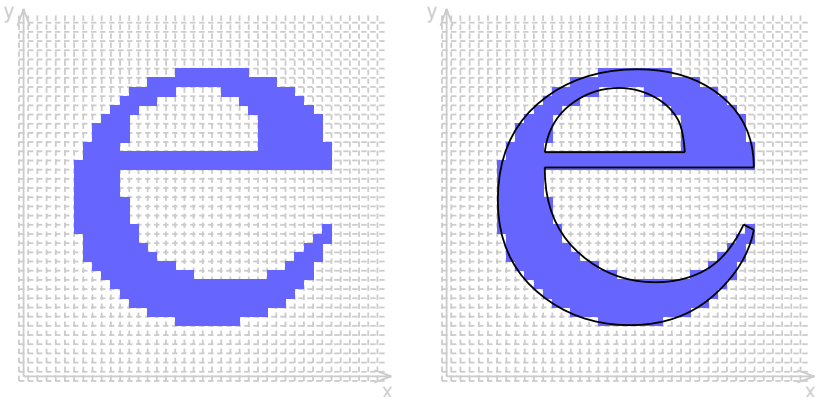
(a) (b)

Slika 7.13: Postupak popunjavanja unutrašnjosti slova.

Na slici 7.13a ispucana zraka najprije probada konturu s desne strane i izlazi na lijevu stranu (prvo sjecište, označeno crvenim križićem), potom probada konturu s lijeve strane i izlazi na desnu stranu (drugo sjecište, označeno zelenim križićem) i konačno probada konturu s desne strane i izlazi na lijevu stranu (treće sjecište, označeno crvenim križićem). Sjecišta u kojima zraka probada konturu s lijeve strane su ulazna; ona u kojima konturu probada s desne strane su izlazna. U slučaju da postoje dva sjecišta s dijelom konture koji je zadan segmentom kvadratne aproksimacijske Bézierove krivulje, oba sjecišta se odbacuju jer situacija odgovara ili ulasku zrake u unutrašnjost slova i potom njezinom izlasku, ili izlasku zrake iz unutrašnjosti slova i potom njezinom ulasku: kako god, ništa se ne mijenja i zraka na kraju ostaje s iste strane gdje je i bila. Kako je broj ulaznih i izlaznih sjecišta na slici 7.13a različit, zaključujemo da slikovni element iz kojega je zraka ispucana pripada unutrašnjosti slova i njega ćemo obojati. Slika 7.13b prikazuje situaciju u kojoj je broj ulaznih i izlaznih sjecišta jednak (jedno je ulazno i jedno izlazno sjecište): stoga je zaključak da slikovni element iz kojeg je zraka ispucana mora biti izvan tijela slova pa se on neće popuniti.

Konačan rezultat popunjavanja unutrašnjosti slova uz prikladnu rezoluciju rastera (jedan slikovni element je jedan kvadratić prikazane mreže) prikazan je na slici 7.14. Opisani postupak daje manje-više prihvatljiv rezultat što je prikazano na slici 7.14a. Rezultat popunjavanja ali uz superponiran obrub kako bi se vidjelo koji su slikovni elementi popunjeni s obzirom na konture slova dan je na slici 7.14b.

Smanjivanjem rezolucije rastera mogu se pojaviti problemi prilikom popunjavanja i to je ilustrirano na slici 7.15. U prikazanom scenariju srednji vodoravni dio slova *e* nije prikazan jer niti jedan centar slikovnog elementa nije pao u prostor ograničen prikazanim konturama. Međutim, od fontova očekujemo da se ispravno



(a) Uz finiju rezoluciju.

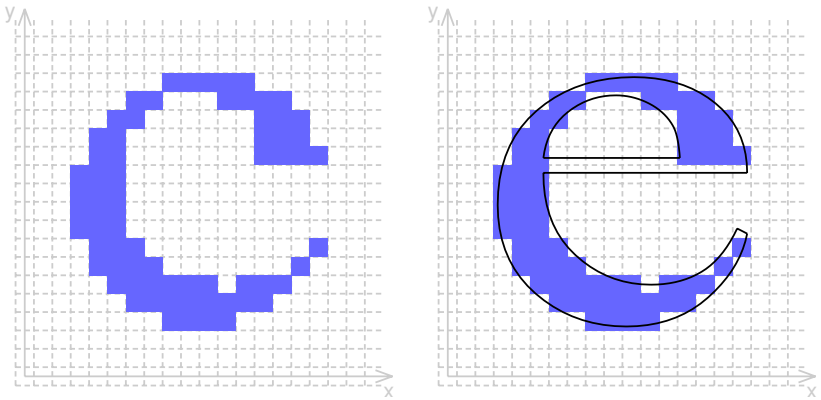
(b) Uz finiju rezoluciju i superponiran obrub.

Slika 7.14: Rezultat popunjavanja unutrašnjosti slova uz dovoljnu rezoluciju rastera.

prikazuju na širokom spektru različitih rezolucija; otvorite primjerice *LibreOffice Writer*, upišite slovo *e* i zatim mijenjajte veličinu fonta kojom se to slovo prikazuje: 20pt, 12pt, 8pt, 6pt – slovo će i dalje ostati čitljivo iako svaka sljedeća veličina ima sve manje i manje slikovnih elemenata kroz koje treba prikazati slovo.

Jedno moguće rješenje problema ilustriranog na slici 7.15a bi bilo točke koje definiraju segmente ovog vodoravnog dijela malo translatirati po vertikali ili prema gore, ili prema dolje, čime bismo dobili jedan red slikovnih elemenata čiji bi centri upali u unutrašnjost slova pa bi ih algoritam obojao. Specifikacija *TrueType* omogućava upravo takve manipulacije: specifikacija definira virtualni stroj (njegovu arhitekturu te podržane instrukcije) i dizajnerima fontova omogućava pisanje programa koji se mogu pridružiti svakom *glyphu* tako da program koji koristi *TrueType* font i koji ga želi prikazati najprije sam skalira i translatira *glyph* za ciljno mjesto u rasteru, potom nad konačnim nizom točaka koje čine konture tako skaliranog *glypha* pokrene u fontu pohranjeni program koji će napraviti još sitna poravnavanja i korekcije u pozicijama točaka i tek potom nad tako podešenim konturama pokrene podprogram za popunjavanje unutrašnjosti *glypha*. Ovaj postupak ponavlja se za svako slovo koje je potrebno prikazati.

Na modernim verzijama operacijskog sustava *Microsoft Windows* programerima su stoga dostupne gotove funkcije za rasterizaciju *TrueType* fontova pri čemu je implementacija virtualnog stroja ugrađena u sam operacijski sustav. Programeru su dostupne funkcije kojima bira font i željenu veličinu te kojima zadaje poziciju na ekranu na kojoj treba rasterizirati znak – cjelokupni daljnji postupak implementira sam operacijski sustav.



(a) Uz grublju rezoluciju.

(b) Uz grublju rezoluciju i superponiran obrub.

Slika 7.15: Rezultat popunjavanja unutrašnjosti slova uz nedovoljnu rezoluciju rastera.

Specifikacija *TrueType* definira još nekoliko naprednih operacija poput *kerninga*; međutim, zainteresiranog se čitatelja na ovom mjestu upućuje na daljnje samostalno istraživanje.

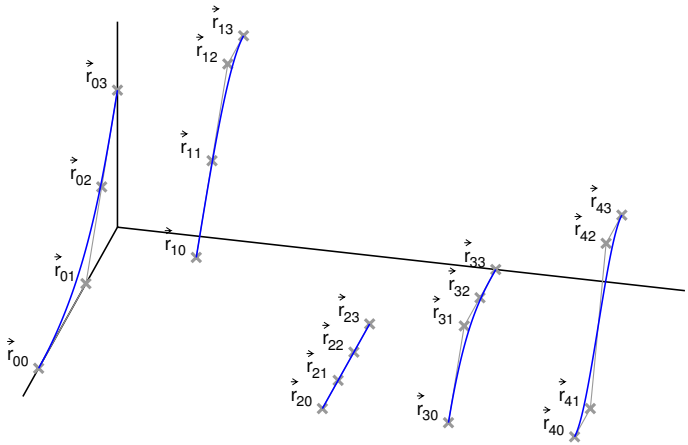
7.8 Površine Béziera

Kroz najveći dio ovog poglavlja razrađivane su različite vrste parametarskih krivulja. Krivulje se, međutim, mogu poopćiti na analitički zadane parametarske površine (u matematici se koristi termin *ploha*), čime se dobivaju sve prednosti vektorske reprezentacije objekata: takvim površinama moći ćemo opisivati plašt tijela na način koji će nam omogućiti skaliranje bez gubitka kvalitete slike.

U okviru ovog potpoglavlja pogledat ćemo kako se površine konstruiraju proširenjem Bézierovih krivulja. Pretpostavimo da je u prostoru zadano $m+1$ aproksimacijskih Bézierovih krivulja, svaka navođenjem $n+1$ točke kontrolnog poligona. Primjer za $m+1 = 5$ i $n+1 = 4$ prikazan je na slici 7.16 gdje su krivulje prikazane plavom bojom.

Označimo točke i-te krivulje s $\vec{d}_i(v)$ gdje je v parametar. Općenit prikaz točaka aproksimacijske Bézierove krivulje preko Bernsteinovih težinskih polinoma dan je izrazom (7.7) koji ćemo ovdje napisati uporabom oznaka prilagođenih ovom primjeru:

$$\vec{d}_i(v) = \sum_{j=0}^{n} b_{j,n}(v) \cdot \vec{r}_{ij}$$



Slika 7.16: 5 aproksimacijskih Bézierovih krivulja

gdje su $b_{j,n}(v)$ Bernsteinove težinske funkcije određene izrazom (7.8). Primjer na slici 7.16 prikazuje 5 aproksimacijskih Bézierovih krivulja 3 stupnja (svaka je zadana s 4 točke). Tako je kontrolni poligon prve krivulje određen točkama $\vec{r}_{00}$, $\vec{r}_{01}$, $\vec{r}_{02}$, $\vec{r}_{03}$, $\vec{r}_{04}$; kontrolni poligon druge krivulje određen je točkama $\vec{r}_{10}$, $\vec{r}_{11}$, $\vec{r}_{12}$, $\vec{r}_{13}$, $\vec{r}_{14}$, i tako dalje redom.

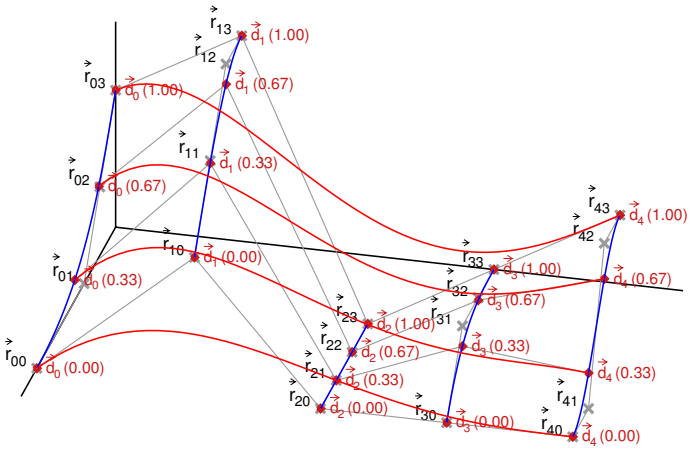
Fiksirajmo sada v na neku konkretnu vrijednost. Tada na prvoj krivulji imamo točku $\vec{d}_0(v)$, na drugoj točku $\vec{d}_1(v)$, na trećoj točku $\vec{d}_2(v)$, na četvrtoj točku $\vec{d}_3(v)$ i na petoj točku $\vec{d}_4(v)$. Tih pet točaka (odnosno općenito $m+1$ jer ih ima onoliko koliko imamo izvorno zadanih krivulja) možemo zamisliti kao novi kontrolni poligon koji određuje novu aproksimacijsku Bézierovu krivulju, ovog puta stupnja m. Ilustracija je dana na slici 7.17 gdje su označene točke i kontrolni poligoni koji se dobiju za $v = 0$, za $v = 0.33$, za $v = 0.67$ te za $v = 1$. Svaka od ovih krivulja prikazana je crvenom bojom.

Kontinuiranim (beskonačno finim) mijenjanjem parametra v od 0 do 1 dobit ćemo beskonačno mnogo kontrolnih poligona s pripadnim aproksimacijskim Bézierovim krivuljama: točke tih krivulja u prostoru će činiti površinu. Aproksimacijska Bézierova krivulja čiji je kontrolni poligon određen točkama $d_0(v)$ do $d_m(v)$ za neki fiksirani v dana je izrazom:

$$\vec{p}(u) = \sum_{i=0}^{m} b_{i,m}(u) \cdot \vec{d}_i(v).$$

Tretiramo li i v kao slobodan parametar te uvrštavanjem izraza za $\vec{d}_i(v)$ dobivamo konačni izraz za sve točke Bézierove površine (ili takozvane Bézierove krpice):

$$\vec{p}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{i,m}(u) \cdot b_{j,n}(v) \cdot \vec{r}_{ij}. \tag{7.41}$$



Slika 7.17: 4 aproksimacijske Bézierove krivulje čiji su kontrolni poligoni definirani točkama na aproksimacijskim Bézierovim krivuljama

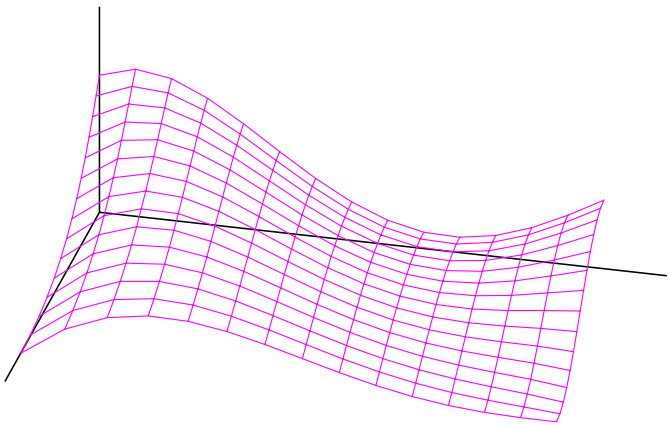
Već smo kod Bézierove krivulje pokazali da se njezin zapis dade lijepo prikazati matrično. Slično vrijedi i u slučaju Bézierove površine gdje možemo pisati:

$$\vec{p}(u,v) = \begin{bmatrix} b_{0,m}(u) & b_{1,m}(u) & \dots & b_{m,m}(u) \end{bmatrix} \cdot \begin{bmatrix} \vec{p}_{0,0} & \cdots & \vec{p}_{0,n} \\ \vec{p}_{1,0} & \cdots & \vec{p}_{1,n} \\ \vdots & \cdots & \vdots \\ \vec{p}_{m,0} & \cdots & \vec{p}_{m,n} \end{bmatrix} \cdot \begin{bmatrix} b_{0,n}(v) \\ b_{1,n}(v) \\ \vdots \\ b_{n,n}(v) \end{bmatrix}. \tag{7.42}$$

Lijeva jednoretčana matrica je matrica Bernsteinovih težinskih funkcija stupnja m dok je desna jednostupčana matrica matrica Bernsteinovih težinskih funkcija stupnja n. Središnja matrica je matrica dimenzija $m+1$ redak puta $n+1$ stupac, odnosno matrica koja ima onoliko redaka koliko ima izvorno zadanih Bézierovih krivulja te onoliko stupaca koliko ima točaka po svakoj zadanoj Bézierovoj krivulji. Treba uočiti da su elementi ove matrice vektori a ne skalari, pa je izraz nemoguće koristiti uz uobičajene matematičke biblioteke koje omogućavaju rad s vektorima i matricama čiji su elementi isključivo skalari. Izraz (7.42) može se međutim razbiti u tri izraza: ako se u središnju matricu uvrste samo x-koordinate radij vektora $\vec{r}_{ij}$, tada će čitav umnožak rezultirati matricom 1×1 – matricom koja zapravo predstavlja skalar: x-koordinatu tražene točke površine. Na isti se način u središnju matricu može ubaciti samo y-koordinata pa izračunati umnožak i potom z koordinata pa izračunati umnožak; time će se dobiti x, y i z koordinata točke površine koja odgovara zadanim u i v.

Vizualizacija Bézierove površine najčešće se radi poligonizacijom (a u tim slučajevima najčešće razbijanjem u trokute). Parametri u i v se mijenjaju u diskretnim koracima od 0 do $1 - \Delta u$ odnosno od 0 do $1 - \Delta v$ gdje je s Δu

označen korak po u a s Δv korak po v. Za svaku se vrijednost (u, v) računaju četiri točke na Bézierovoj površini: $\vec{p}(u, v)$, $\vec{p}(u + \Delta u, v)$, $\vec{p}(u + \Delta u, v + \Delta v)$ i $\vec{p}(u, v + \Delta v)$, čime se dobije četverokutna krpica površine. Jedan primjer takve diskretizacije, gdje su prikazane krpice, prikazan je na slici 7.18.



Slika 7.18: Prikaz Bézierove površine diskretnim krpicama

Želimo li dobiti trokute, kao točke prvog tokuta mogu se uzeti točke $\vec{p}(u, v)$, $\vec{p}(u + \Delta u, v)$ i $\vec{p}(u + \Delta u, v + \Delta v)$ a kao vrhovi drugog trokuta mogu se uzeti točke $\vec{p}(u, v)$, $\vec{p}(u + \Delta u, v + \Delta v)$, $\vec{p}(u, v + \Delta v)$. Uz konzistentno zadane Bézierove površine, ovako dobiveni trokuti bit će uvijek iste orijentacije, ta nakon što napravimo generiranje trokuta, možemo koristiti uobičajene algoritme za uklanjanje stražnjih trokuta o čemu će biti više riječi u poglavlju 8.

Vratimo se još nakratko na izraz (7.42). Lijeva jednoretčana matrica Bernsteinovih težinskih funkcija stupnja m te desna jednostupčana matrica Bernsteinovih težinskih funkcija stupnja n mogu se raspisati kao matrica potencija parametra puta matrica koeficijenata $\mathbf{B}$ kako smo to pokazali kod Bézierove krivulje. S obzirom da ovdje radimo s krivuljama različitih stupnjeva, uvest ćemo oznaku $\mathbf{B}_k$ gdje k označava stupanj krivulje kojemu odgovara ova matrica. Prisjetimo se, vrijedi:

$$\begin{bmatrix} b_{0,m}(u) & b_{1,m}(u) & \dots & b_{m,m}(u) \end{bmatrix} = \begin{bmatrix} u^m & u^{m-1} & \dots & u & 1 \end{bmatrix} \cdot \mathbf{B}_m$$

$$\begin{bmatrix} b_{0,n}(v) & b_{1,n}(v) & \dots & b_{n,n}(v) \end{bmatrix} = \begin{bmatrix} v^n & v^{n-1} & \dots & v & 1 \end{bmatrix} \cdot \mathbf{B}_n.$$

Uvrštavanjem u izraz (7.42) dobivamo:

$$\vec{p}(u,v) = \begin{bmatrix} u^m & u^{m-1} & \dots & u & 1 \end{bmatrix} \cdot \mathbf{B}_m \cdot \begin{bmatrix} \vec{p}_{0,0} & \cdots & \vec{p}_{0,n} \\ \vec{p}_{1,0} & \cdots & \vec{p}_{1,n} \\ \vdots & \cdots & \vdots \\ \vec{p}_{m,0} & \cdots & \vec{p}_{m,n} \end{bmatrix} \cdot \mathbf{B}_n^T \begin{bmatrix} v^n \\ v^{n-1} \\ \vdots \\ v \\ 1 \end{bmatrix}. \quad (7.43)$$

Prethodni izraz još ćemo kraće pisati ovako:

$$\vec{p}(u,v) = \mathbf{U}_m \cdot \mathbf{B}_m \cdot \mathbf{P} \cdot \mathbf{B}_n^T \cdot \mathbf{V}_n^T. \quad (7.44)$$

Pri tome $\mathbf{U}_m$ označava jednoretčanu matricu potencija parametra u do m-te potencije, $\mathbf{B}_m$ označava pripadnu matricu koeficijenata koja pripada raspisu Bernsteinovih polinoma m-tog stupnja, $\mathbf{P}$ je matrica zadanih točaka, $\mathbf{B}_n$ označava matricu koeficijenata koja pripada raspisu Bernsteinovih polinoma n-tog stupnja i $\mathbf{V}_n$ označava jednostupčanu matricu potencija parametra v do n-te potencije.

U računalnoj grafici vrlo je česta uporaba Bézierovih površina $m \times n = 3 \times 3$ koje su zadane s četiri Bézierove krivulje od kojih je svaka zadana s četiri točke kontrolnog poligona. Ovakva površina naziva se još i *bikubična*. U tom slučaju imamo 16 zadanih točaka a matrice $\mathbf{B}_m = \mathbf{B}_n = \mathbf{B}_3$ pa u tom slučaju izraz (7.43) prelazi u:

$$\vec{p}(u,v) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \vec{p}_{0,0} & \vec{p}_{0,1} & \vec{p}_{0,2} & \vec{p}_{0,3} \\ \vec{p}_{1,0} & \vec{p}_{1,1} & \vec{p}_{1,2} & \vec{p}_{1,3} \\ \vec{p}_{2,0} & \vec{p}_{2,1} & \vec{p}_{2,2} & \vec{p}_{2,3} \\ \vec{p}_{3,0} & \vec{p}_{3,1} & \vec{p}_{3,2} & \vec{p}_{3,3} \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}. \quad (7.45)$$

Prilikom uporabe ovog izraza tipično će se izračun rastaviti u tri koraka: najprije će se u središnju matricu uvrstiti x-koordinate točaka pa izračunati izraz i time dobiti x-koordinata točke površine nakon ćega će se isto ponoviti za y-koordinatu i z-koordinatu.

7.8.1 Određivanje normala

Prilikom postupaka uklanjanja stražnjih poligona te prilikom postupaka sjenčanja trebat ćemo i vektore normala u izračunatim točkama. Normalu ravnine već smo naučili izračunati i ona je za ravninu nepromjenjiva. Bézierova površina,

međutim, može biti zakrivljena, pa se normala iz točke u točku može mijenjati. Stoga je postupak određivanja normale sljedeći (uočite da su točke Bézierove površine određene kao funkcija od parametara u i v):

1. U zadanoj točki odrediti parcijalnu derivaciju funkcije površine s obzirom na parametar u. Rezultat je tangencijalni vektor u smjeru u.

$$\begin{aligned}\vec{t}_u &= \frac{\partial \vec{p}(u,v)}{\partial u} \\ &= \frac{\partial}{\partial u}\left(\mathbf{U}_m \cdot \mathbf{B}_m \cdot \mathbf{P} \cdot \mathbf{B}_n^T \cdot \mathbf{V}_n^T\right) \\ &= \left(\frac{\partial \mathbf{U}_m}{\partial u} \cdot \mathbf{B}_m \cdot \mathbf{P} \cdot \mathbf{B}_n^T \cdot \mathbf{V}_n^T\right)\end{aligned}$$

   Primjerice, ako je

$$U_3 = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

   tada je

$$\frac{\partial \mathbf{U}_3}{\partial u} = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix}$$

   odnosno svaki se član matrice $\mathbf{U}$ nezavisno derivira.

2. U zadanoj točki odrediti parcijalnu derivaciju funkcije površine s obzirom na parametar v. Rezultat je tangencijalni vektor u smjeru v.

$$\begin{aligned}\vec{t}_v &= \frac{\partial \vec{p}(u,v)}{\partial v} \\ &= \frac{\partial}{\partial v}\left(\mathbf{U}_m \cdot \mathbf{B}_m \cdot \mathbf{P} \cdot \mathbf{B}_n^T \cdot \mathbf{V}_n^T\right) \\ &= \left(\mathbf{U}_m \cdot \mathbf{B}_m \cdot \mathbf{P} \cdot \mathbf{B}_n^T \cdot \frac{\partial \mathbf{V}_n}{\partial v}^T\right)\end{aligned}$$

   Primjerice, ako je

$$V_3 = \begin{bmatrix} v^3 & v^2 & v & 1 \end{bmatrix}$$

   tada je

$$\frac{\partial \mathbf{V}_3}{\partial v} = \begin{bmatrix} 3v^2 & 2v & 1 & 0 \end{bmatrix}$$

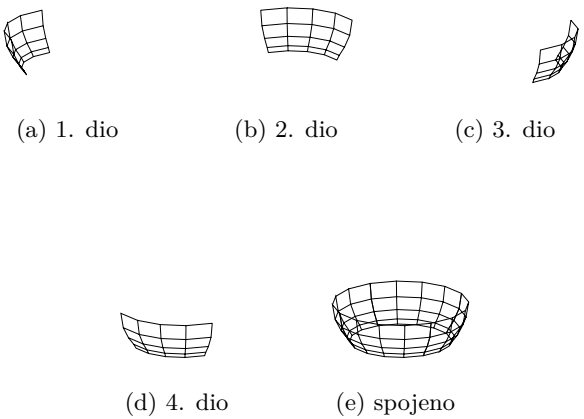
   odnosno svaki se član matrice $\mathbf{V}$ nezavisno derivira.

3. Normala u točki određenoj s (u, v) računa se kao vektorski produkt pripadnih tangencijalnih vektora:

$$\vec{n} = \vec{t}_u \times \vec{t}_v.$$

7.8.2 Modeliranje plašta tijela Bézierovim površinama

Jedan od često korištenih modela u računalnoj grafici je čajnik. Za modeliranje čajnika korištene su bikubične Bézierove krpice zadane kontrolnim poligonom sastavljenim od 4×4 kontrolne točke. Slika 7.19 prikazuje kako je donji dio plašta čajnika sklopljen od četiri Bézierove površine.

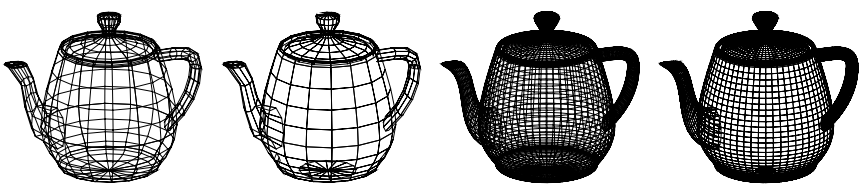


(a) 1. dio (b) 2. dio (c) 3. dio

(d) 4. dio (e) spojeno

Slika 7.19: Dio plašta čajnika izrađen od četiri Bézierove površine.

Svaka Bézierova površina na slici 7.19 prikazana je žičanim modelom koji je nastao uzorkovanjem parametara u i v u pet točaka čime je dobiven relativno grub prikaz modela čajnika. Cjelokupni čajnik prikazan uz različite gustoće uzorkovanja prikazan je na slici 7.20.



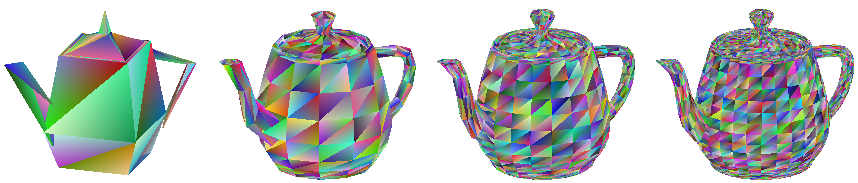
(a) 5 uzoraka po krpici (b) 5 uzoraka po krpici (c) 15 uzoraka po krpici (d) 15 uzoraka po krpici

Slika 7.20: Čajnik izrađen od 32 Bézierove površine.

Slike 7.20a i 7.20c pri tome prikazuju sve krpice pa se u prikazu vide i prednji i stražnji dijelovi čajnika. Prikaz koji je očišćen od krpica za koje je utvrđeno da su stražnje prikazan je na slikama 7.20b i 7.20d.

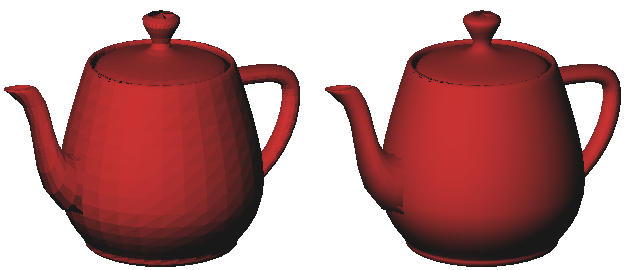
Prikaz čajnika uz generiranje mreže trokuta iz površine i bojanjem svakog trokuta slučajno odabranom bojom dan je na slici 7.21. Slika 7.21a prikazuje situaciju u kojoj je svaka površina po u i v uzorkovana na dva mjesta čime je dobiven vrlo grub prikaz čajnika. Postupno finija uzorkovanja prikazana su na slikama 7.21b, 7.21c i 7.21d.

Konačno, prikaz čajnika koji je dobiven podjelom Bézierovih površina u trokute i potom obojan postupkom sjenčanja dan je na slici 7.22. Više o postupcima sjenčanja bit će riječi u poglavlju 9; ovdje treba samo zapamtiti da se modeli definirani Bézierovim površinama mogu poligonizirati i potom koristiti kroz uobičajene algoritme sjenčanja.



(a) 2 uzorka po krpici (b) 4 uzorka po krpici (c) 6 uzoraka po krpici (d) 8 uzoraka po krpici

Slika 7.21: Čajnik izrađen od 32 Bézierove površine uzorkovan uz različite stupnjeve detaljnosti i obojan slučajno odabranim bojama.



(a) Konstantno sjenčanje (b) Gourardovo sjenčanje

Slika 7.22: Čajnik izrađen od 32 Bézierove površine i osjenčan.

7.8.3 Prednosti i svojstva Bézierovih površina

Čest način pohrane modela objekata u igrama su različite kolekcije poligona. Štoviše, kako bi se osigurao što brži prikaz scene, za isti model postoji više kolekcija poligona, svaka dobivena uz različitu razinu detalja. Tako dugo dok je objekt dovoljno daleko od igrača, za prikaz se koristi model s najmanjom razinom detalja a kako objekt postaje sve bliži, bira se model koji ima sve više detalja. Ovakav način pohrane podataka o objektima može iziskivati veliku količinu memoriju jer je za isti objekt potrebno čuvati više modela. Opisuje li se objekt Bézierovim površinama, modeli s različitom razinom detalja mogu se generirati u letu po potrebi – količina detalja ovisit će o gustoći uzorkovanja parametara u i v. Također, čitav prikaz je memorijski vrlo kompaktan – sve što je potrebno jest pohraniti točke kontrolnog poligona. Činjenica da se količina detalja može dinamički određivati pogodna je i za igre koje moraju moći raditi i na brzim i na sporijim računalima: na sporijim računalima igra može naprosto koristiti grublje uzorkovanje površine i time stvarati manju količinu grafičkih primitiva koje je potrebno obraditi.

Od svojstava Bézierovih površina nabrojat ćemo samo najvažnija.

1. $\vec{p}(u,v)$ prolazi kroz četiri ugla zadane mreže točaka: $\vec{p}(0,0) = \vec{r}_{00}$, $\vec{p}(1,0) = \vec{r}_{m0}$, $\vec{p}(0,1) = \vec{r}_{0n}$ i $\vec{p}(1,1) = \vec{r}_{mn}$. U općem slučaju kroz unutarnje točke mreže površina ne mora prolaziti.

2. $b_{i,m}(u)$ i $b_{j,n}(v)$ su nenegativni za sve m, n, i, j uz u i v unutar raspona $[0,1]$.

3. $\sum_{i=0}^{m} \sum_{j=0}^{n} b_{i,m}(u) \cdot b_{j,n}(v) = 1$ uz u i v unutar raspona $[0,1]$.

4. Bézierova krpica $\vec{p}(u,v)$ leži unutar konveksne ljuske koju definira mreža kontrolnih točaka.

5. *Invarijantnost na afine transformacije.* Površina koju dobijemo ako na sve točke Bézierove površine primijenimo afinu transformaciju ista je kao i površina koju dobijemo ako je računamo iz točaka koje dobijemo tako da afinu transformaciju primijenimo samo na točke kontrolne mreže.

6. Fiksiranjem jednog od parametara u $\vec{p}(u,v)$ dobiva se Bézierova krivulja – ovo je trivijalno vidljivo iz (7.41).

7. Postoje četiri rubne Bézierove krivulje koje omeđuju površinu: $\vec{p}(0,v)$, $\vec{p}(1,v)$, $\vec{p}(u,0)$ i $\vec{p}(u,1)$.

7.9 Ponavljanje

1. Što znači da je krivulja zadana eksplicitnom jednadžbom, implicitnom jednadžbom odnosno parametarskim zapisom? U čemu su razlike?
2. Kako klasificiramo krivulje? Koja su poželjna svojstva?
3. Pojasnite pojam C-neprekidnost.
4. Ako znamo točke kontrolnog poligona, kako je definirana Bézierova krivulja? Kojeg je ona stupnja? Kako glase pripadne bazne funkcije, a kako izgleda matrični zapis ove krivulje?
5. Ako znamo kroz koje točke Bézierova krivulja mora proći, na koji način možemo doći do njezinog matričnog zapisa?
6. Što su Bernsteinove težinske funkcije, a što Bézierove? Kako glase zapisi Bézierove krivulje koji koriste jedne odnosno druge težinske funkcije?
7. Kako se za neku konkretnu vrijednost parametra t može konstrukcijom odrediti pripadna točka kroz koju će proći Bézierova krivulja za tu vrijednost parametra ako je zadan kontrolni poligon?
8. Kako glasi matrični zapis parametarskog prikaza krivulje pomoću polinoma?
9. Kako glasi matrični zapis parametarskog prikaza krivulje pomoću razlomljenih funkcija?
10. Gdje (odnosno za što) se koriste Bézierove krivulje kod TrueType fontova?
11. Kako su definirane Bézierove površine?

Poglavlje 8

Uklanjanje skrivenih linija i površina

8.1 Uvod

Objekte scene opisujemo matematičkim primitivima: vrhovima, linijama, poligonima, funkcijama i sl. Rad s ovakvim opisom postaje računski zahtjevan kako scena raste i kako detaljnost pojedinih objekata raste. Iako broj poligona koji u jedinici vremena grafička kartica može prikazati, kroz nekoliko posljednjih godina, raste nevjerojatnom brzinom, rastu i naša očekivanja na vjernost prikaza i različite učinke koje nam prikaz omogućava. Na ovaj način scena postaje sve složenija a time i vrijeme potrebno da se ostvari njen prikaz. Da bi se iscrtavanje scene maksimalno ubrzalo, treba iz postupka iscrtavanja izbaciti sve radnje koje su nepotrebne (a takvih ima). Npr. ukoliko u scenu stavimo kocku, gdjegod da se nalazio promatrač, nikada mu neće biti vidljive sve stranice kocke. "Nevidljive" stranice zapravo su stražnje stranice kocke gledano iz smjera promatrača. Stražnje stranice ne treba niti sjenčati niti preslikavati teksturu na njih jer ionako nisu vidljive i trebaju biti uklonjene (engl. *backface culling*). Kocka predstavlja trivijalan primjer, no treba imati na umu da će objekt koji ima nekoliko milijuna poligona, uz uklanjanje samo stražnjih poligona, imati oko pedeset posto manje poligona koje treba prikazati, čime ćemo bitno rasteretiti grafički protočni sustav (engl. *graphics rendering pipeline*). Scenu također treba promatrati u cjelini, a iscrtavanje svih objekata i dijelova objekata – vidljivih i nevidljivih – nepotrebno će preopteretiti prikaz, kao na slikama 8.1 i 8.2. Na slici 8.1 je vidljivo da je prisutno puno stražnjih poligona: za svaki stup barem ih je pola stražnjih, tj. koji nisu vidljivi. Stupovima je zaklonjen i veliki dio scene koji je iza njih i nije vidljiv. A ako pogledamo i same stupove, međusobno se zaklanjaju, pa sagledavši sve skupa imamo veliku količinu poligona koja će se iscrtavati, a neće biti vidljiva

Slika 8.1: Primjer scene s puno objekata na kojoj je prisutna velika količina objekata (npr. stupovi u pozadini) i poligona koji se ne vide ili su djelomično zaklonjeni. Slika je iz igre The Talos Principle tvrtke CroTeam.

ili će se djelomično zaklanjati. Na slici 8.2 prikazana je velika otvorena scena gdje je također prisutan jako veliki broj poligona koji se djelomično ili u cijelosti zaklanjaju.

No, treba biti oprezan u odlučivanju kada su pojedine plohe zaista nevidljive, odnosno kada se smiju ukloniti. Ako u sceni imamo zrcala ili prozirne objekte tada se može desiti da neke dijelove objekata vidimo indirektno i u tom slučaju te poligone ne smijemo ukloniti. To može biti odraz na podu ili na površini vode, pa će neki dijelovi, iako su stražnji za promatrača, biti vidljivi na zrcalnoj površini ili kroz proziran objekt. Drugo na što treba paziti je ako poligone smijemo ukloniti, na kojem mjestu u protočnom sustavu to treba učiniti. Proračun

Slika 8.2: Primjer velike otvorene scene s puno poligona. Svaki grm predstavljen je poluprozirnim poligonom koji je potrebno iscrtati.

normala u vrhovima zahtijeva poznavanje normala susjednih poligona kako bi ih mogli odrediti. Ako uklonimo nevidljive poligone za promatrača uklonit ćemo i poligone koji diraju vrhove na silueti objekta. Učinimo li to prije izračunavanja normala u vrhovima, za točke na rubnim dijelovima objekta nećemo imati potrebne poligone, odnosno njihove normale, potrebne za određivanje normala u vrhovima.

Općenito, cilj algoritama za uklanjanje skrivenih linija i površina (engl. *hidden surface removal*) je ukloniti što je moguće prije i što je moguće više toga što je nepotrebno u grafičkom protočnom sustavu. Stoga se različite provjere rade na raznim mjestima i u pravilu na više mjesta u protočnom sustavu. Provjere koje nisu složene i nemaju zahtjevan izračun, a uklanjaju dijelove koji sigurno nisu vidljivi, obavljaju se ranije, kako bi do vremenski zahtjevnijih algoritama došao što manji broj nepotrebnih poligona. Uz sve provjere koje načinimo do konačnog koraka iscrtavanja, još uvijek dolazi u pravilu više pristupa iscrtavanju slikovnih elemenata nego što će u konačnici biti vidljivo. Tada promatramo koliko tog "viška" dolazi do kraja protočnog sustava (engl. *overdraw*). Procjenjuje se da je taj broj u aplikacijama kao što su računalne igre, otprilike tri, uz sve prethodne provjere i uklanjanje skrivenih dijelova. Odnosno, za iscrtavanje svakog slikovnog elementa na zaslonu bit će tri pokušaja iscrtavanja kako bi se iscrtao slikovni element koji će se u konačnici vidjeti. To se, naravno, obavlja prije konačnog osvježavanja iz slikovne memorije na zaslon.

Neke provjere koje ćemo ovdje objasniti imaju sklopovsku podršku jer značajno pridonose rasterećenju prikaza i k tome se obavljaju vrlo brzo, a neke su dane kao ideje koje se mogu koristiti i u različitim drugim kontekstima. Na primjer, kod detekcije kolizije javljaju se slični problemi kao i kod uklanjanja skrivenih linija i površina. Kod detekcije kolizije potrebno je odrediti prodire li jedan objekt u drugi. Kada i gdje dolazi do kontakta, potrebno je odrediti kako bi mogli načiniti međusobnu reakciju pri sudaru dva objekata. Ovi problemi su slični, pa su i algoritmi koji se koriste za uklanjanje skrivenih linija i površine, odnosno načini razmišljanja primjenjivi. Postupci odsijecanja koji se obavljaju obzirom na volumen pogleda ili otvor prikaza, također daju važne ideje i upotrebivi su u raznim drugim kontekstima. Bačene sjene, ako ih želimo realizirati u stvarnom vremenu također vuku bitne koncepte koji su se izvorno primjenjivali na problematiku uklanjanja skrivenih linija i površina. Vrijedi, naravno, i obrnuto nove ideje koje se javljaju pri razvoju algoritama detekcije sudara ili ostvarivanja sjene mogu biti upotrebive u postupcima uklanjanja skrivenih linija i površina. Primjer je ideja koja je razvijena za postupak ostvarivanja sjena. Objekti se dijele na one koji su zaklanjajući i objekte koji su zaklonjeni, odnosno objekte u području sjene i nisu vidljivi za izvor. Način određivanja zaklonjenih objekata primjenjiv je i u postupcima određivanja skrivenih linija i površina.

S obzirom da se radi o cijelom nizu algoritama koji se koriste u ove svrhe, možemo ih podijeliti na algoritme koji rade u prostoru scene – trodimenzionalnom

prostoru – i algoritme koji rade u prostoru projekcije, odnosno u dvodimenzionalnom prostoru. Valja primijetiti i da, iako ćemo načiniti ovu podjelu, pojedini algoritmi koji su definirani u dvodimenzionalnom prostoru proširivi su i na trodimenzionalan prostor. Također, možemo primijetiti da su neki postupci vrlo jednostavni, no mogu ukloniti ne mali broj objekata ili njihovih dijelova, pa ih je stoga poželjno što prije i češće koristiti.

Sve skupa, zajedno gledano, algoritmi uklanjanja skrivenih linija i površina troše neko vrijeme, no njihova primjena utječe na uštedu zbog smanjenja broja objekata ili njihovih dijelova u konačnom prikazu. U ovom poglavlju način ćemo pregled kako vrlo jednostavnih, tako i složenih algoritama koji troše vrijeme i memoriju za pohranu struktura podataka. Utrošak vremena na različite provjere mora biti isplativ prema učinku koji se time postiže. Funkcija cilja je povećati broj iscrtanih slika u jedinici vremena (*fps*), pa je to obično glavni kriteriju u ocjeni koji će algoritam i gdje biti korišten, uz prihvatljiv utrošak za strukture podataka.

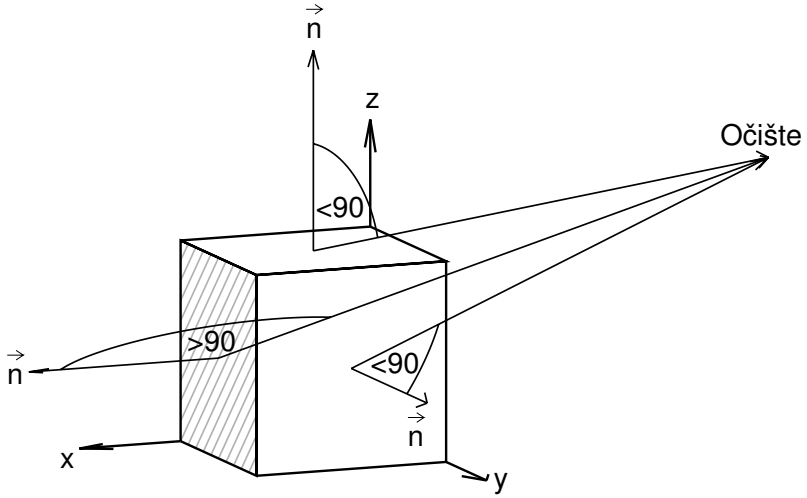
8.2 Uklanjanje stražnjih poligona – provjera normale

Ovaj postupak pomoći će nam da otkrijemo koji su poligoni tijela stražnji u odnosu na promatrača. Pretpostavka je da je tijelo opisano nizom poligona, i da je tijelo konveksno. Nadalje, pretpostavlja se da su vrhovi poligona zadani u smjeru suprotnom od smjera kazaljke na satu ako gledamo poligon iz točke koja se nalazi izvan tijela. Ovo će za posljedicu imati da sve normale poligona gledaju iz tijela prema van. Tada se odluka je li poligon prednji ili stražnji može donijeti na temelju kuta što ga *normala poligona* zatvara s *vektorom usmjerenim prema gledatelju*. Slika 8.3. demonstrira takvo zaključivanje.

Potrebno je promatrati kut što ga zatvara vektor normale poligona i vektor iz središta poligona prema promatraču. Tada vrijedi:

- poligon je stražnji, ako je $\vec{N}_P \cdot \vec{N} < 0$,
- poligon je prednji (vidljiv), ako je $\vec{N}_P \cdot \vec{N} > 0$, te
- poligon je degenerirao u liniju (za promarača) ako je $\vec{N}_P \cdot \vec{N} = 0$.

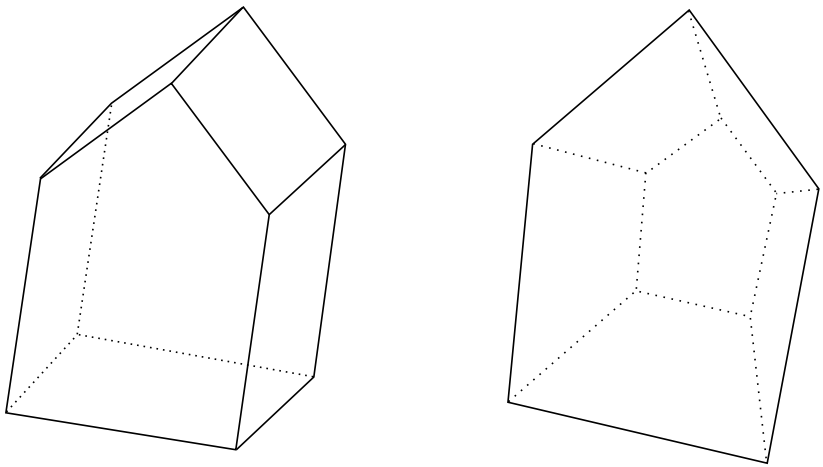
Uklanjanje stražnjih poligona možemo načiniti korištenjem još jedne jednostavnije provjere. Položaj promatrača možemo uvrstiti u jednadžbu ravnine koja sadrži pojedini poligon. Ako je točka promatrača "iznad" taj poligon je vidljiv, a ako je "ispod" poligon nije vidljiv. Ova provjera ima manje računskih operacija, a u osnovi radimo istu stvar kao i kod provjere kuta između vektora normale i vektora prema očištu.



Slika 8.3: Uklanjanje poligona na osnovi kuta između vektora normale poligona i vektora prema promatraču

Pri uklanjanju stražnjih poligona važno je obratiti pažnju na vrstu projekcije koja se obavlja. Ako se obavlja ortografska projekcija, neće biti problema, no ako se koristi perspektivna projekcija treba biti oprezan. Na slici 8.4 prikazan je objekt uz ortografsku projekciju lijevo i perspektivnu desno, za isti položaj promatrača. Crtkano su prikazane linije koje čine bridove stražnjih poligona i ne bi trebale biti vidljive. Treba primijetiti da su poligoni koji su skriveni u jednom i drugom slučaju različiti. Ako ne vodimo računa o transformaciji normala poligona pri perspektivnoj projekciji krov kućice i lijeva strana u projekciji perspektivnom projekcijom bit će prikazani iako nisu vidljivi u toj projekciji. Znači, prilikom korištenja perspektivne projekcije moramo voditi računa da transformaciji podvrgnemo vektore normala koje smo izračunali na početku ili da nakon obavljanja projekcije u transformiranom prostoru računamo jednadžbe ravnina na osnovi transformiranih vrhova. Kako često želimo mijenjati položaj očišta, u prvom pristupu imat ćemo manji broj računskih operacija.

Postupak uklanjanja stražnjih poligona može raditi i u prostoru projekcije. Ovaj pristup temelji se na provjeri orijentacije poligona u 2D prostoru projekcije. Možemo primijetiti da je, za poligone koji u trodimenzionalnom prostoru imaju redoslijed obilaženja vrhova u smjeru suprotnom od smjera kazaljke na satu gledano izvan objekta, u prostoru projekcije ta orijentacija sačuvana za poligone okrenute prema promatraču, a obrnuta je za promatraču skrivene poligone. Ako koristimo ovu provjeru ne moramo paziti na vrstu projekcije jer će i kod ortografske i kod perspektivne projekcije navedeno svojstvo vrijediti.



Slika 8.4: Ortografska i perspektivna projekcija

Postupak uklanjanja stražnjih poligona obično je sklopovski podržan. Standardom *OpenGL* ova je operacija podržana, i kako bi je aktivirali, potrebno je zadati niz naredbi prikazanih u kodu u nastavku.

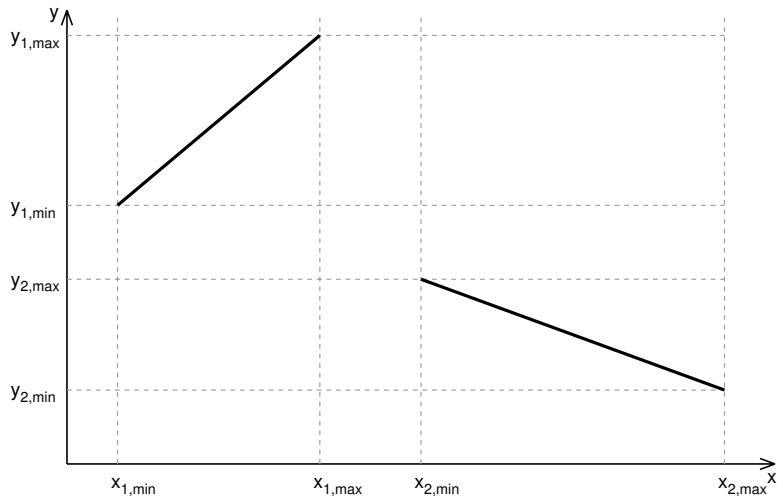
```
glFrontFace(GL_CCW);      // ili GL_CW
glEnable(GL_CULL_FACE); // glDisable(GL_CULL_FACE);
glCullFace(GL_BACK);      // ili GL_FRONT ili GL_FRONT_AND_BACK
```

Prvo moramo odrediti koju orijentaciju imaju poligoni koje smatramo prednjima. Nakon toga moramo omogućiti uklanjanje skrivenih poligona, te odabrati koju orijentaciju želimo koristiti pri uklanjanju.

Uporabom konstante GL_FRONT_AND_BACK možemo ukloniti i prednje i stražnje poligone. Ova operacija ima smisla ako u sceni imamo i linijske segmente koji na ovaj način neće biti uklonjeni pa će biti istaknuti.

8.3 Minimaks provjere

Minimaks provjera služi nam kao brza provjera da li se dva objekta *ne preklapaju*. Prethodna rečenica zvuči malo čudno, ali ispravno karakterizira postupak. Naime, minimaks provjerama možemo utvrditi je li sigurno da se dva objekta ne preklapaju. Ukoliko pak utvrdimo da postoji mogućnost preklapanja objekata, tada daljnje provjere treba obavljati drugim (tipično kompleksnijim i sporijim) algoritmima. Evo ideje. Svaki objekt u sustavu prikaza ima svoje minimalne i maksimalne koordinate kroz koje se proteže. Usporedbom tih vrijednosti možemo donijeti neke zaključke. Primjer je prikazan na slici 8.5 Provjerava se preklapanja dvaju segmenata linije. Linije su uzete radi jednostavnosti, no postupak je identičan za bilo kakve druge objekte.



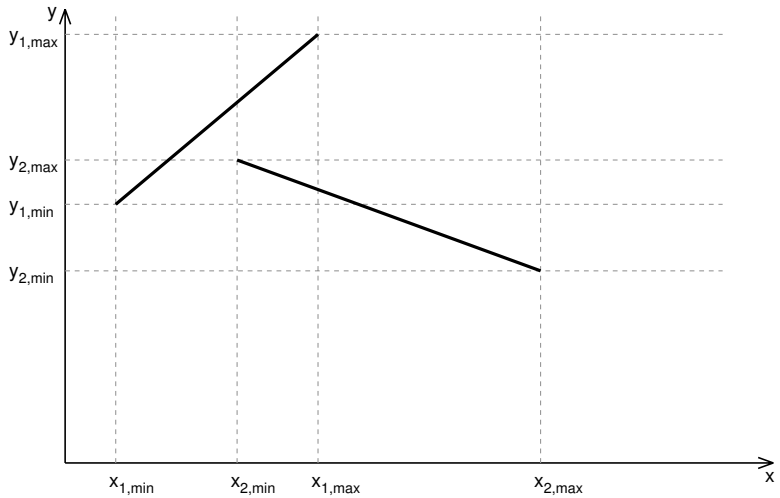
Slika 8.5: Primjer min-maks provjere na dvije dužine

Postavlja se pitanje, kada možemo biti sigurni da se objekti ne preklapaju? Četiri su takva slučaja. Objekti se sigurno ne preklapaju, ako je:

- $x_{1,max} < x_{2,min}$ ili
- $x_{2,max} < x_{1,min}$ ili
- $y_{1,max} < y_{2,min}$ ili
- $y_{2,max} < y_{1,min}$.

Ako je zadovoljen bilo koji od ova četiri uvjeta, tada smo sigurni da se objekti ne preklapaju. Ako niti jedan od ta četiri uvjeta nije ispunjen, tada treba izvršiti daljnje provjere. Naime, i dalje je moguće da se objekti ne preklapaju, a da niti jedan od prethodnih uvjeta nije ispunjen. Ovakav primjer ilustriran je na slici 8.6, gdje se objekti i dalje ne preklapaju, no minimaks provjerom to ne možemo utvrditi. Minimaks provjera u ovom slučaju rezultirat će potrebom za dodatnim provjerama, jer se kvadratna područja u kojima leže objekti preklapaju.

Valja primjetiti kako je osnova minimaks provjere zapravo provjera koja gleda preklapaju li se projekcije objekata na os x, ili na osi y. U općem slučaju to može biti bilo koja os, odnosno pravac. Koordinatne osi posebno su pogodne zbog jednostavnosti primjene. Kada provjeravamo preklapanje po obje osi x i y u stvari provjeravamo da li se preklapaju pravokutnici koji obuhvaćaju promatrane objekte. Ako se pravokutnici ne preklapaju tada zaključujemo da se objekti sigurno ne preklapaju, odnosno da siguno postoji pravac koji je između promatranih

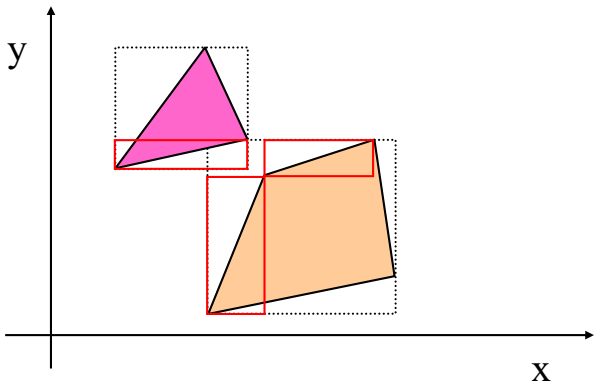


Slika 8.6: Primjer kada se dvije dužine potencijalno preklapaju

objekata. Kod složenijih objekata postupak možemo hijerarhijski primijeniti. Na primjer, provjeravamo li za dva poligona da li se preklapaju te utvrdimo da se potencijalno preklapaju, provjeru možemo primijeniti na međusoban odnos svaka dva brida koji sačinjavaju poligon. Ovaj slučaj je prikazan na slici 8.7 gdje se poligoni potencijalno preklapaju, ali usporedbom bridova potom zaključujemo da se niti jedan brid prvog objekta sigurno ne preklapa niti s jednim bridom drugog objekta. Iz toga možemo zaključiti da su dva poligona ili bez preklapanja, ili je jedan u cijelosti sadržan unutar drugoga. Točan odnos potrebno je dodatno provjeriti.

Opisana ideja proširiva je na trodimenzionalni prostor gdje se provjerava preklapanje omeđujućih kvadara. *Omeđujući kvadri* su minimalni kvadri koji u cijelosti obuhvaćaju objekt. Ovaj postupak koristi se pri detekciji kolizije dva objekta i naziva se *provjera omeđujućih kvadara poravnatih s koordinatnim osima* (engl. *AABB axes aligned bounding box*). Nedostatak ovog postupka je u tome što ako je objekt uzak i dugačak, kvadar koji ga obuhvaća zahvaća velik dio prostora, posebno kada je objekt položen dijagonalno, pa je procjena preklapanja objekata u tom slučaju vrlo loša.

Želimo li napraviti ispitivanje složenijih objekata uporabom ovog algoritma, najprije ćemo složenije objekte obuhvatiti kvadrima. Provjeru jesu li objekti u koliziji najprije ćemo obaviti nad kvadrima jer je to bitno jednostavnije. Tek ako se objekti potencijalno preklapaju idemo u vremenski zahtjevnije, odnosno skuplje provjere. Objekti kojima obuhvaćamo naše složene objekte mogu biti i kugle,



Slika 8.7: Poligoni se potencijalno preklapaju, ali niti jedan brid prvog objekta sigurno se ne preklapa s niti jednim bridom drugog objekta

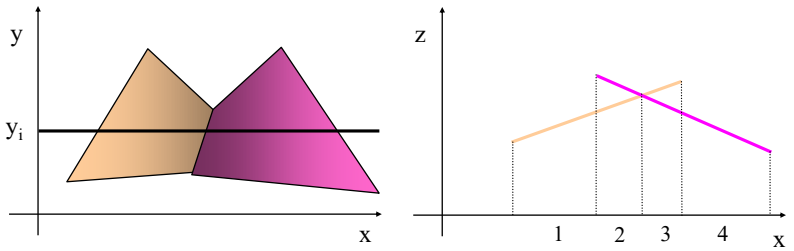
koje se često koriste zbog jednostavnosti provjere. Za dvije kugle dovoljno je provjeriti udaljenost od njihovih središta; kako znamo radijuse tih kugli, trivijalno je provjeriti preklapaju li se potencijalno objekti koje te kugle obuhvaćaju.

8.4 Postupak Watkinsa

Postupak koji je osmislio Watkins često se referencira i kao *algoritam linije pretrage* (engl. *scan-line algorithm*), odnosno nastavlja se na ovaj algoritam. Algoritam radi u prostoru projekcije. Osnovna ideja proizlazi iz postupka rasterizacije poligona liniju po liniju, tako da je cilj iskoristiti informaciju koja čini kontekst i proširiti je duž linije pretrage, a isto tako i duž bridova koji čine poligone.

Poligone promatramo u ravnini projekcije; neka to za potrebe primjera bude xy-ravnina. Krenemo li promatrati za jednu liniju pretrage, potrebno je odrediti koji su dijelovi vidljivi. Na slici 8.8 prikazana su dva poligona koja se po dubini sijeku. Trenutno promatrana linija pretrage je označena s y_i. Primijetite da je u 3D prostoru linija pretrage zapravo ravnina $y = y_i$, tj. ravnina paralelna s xz-ravninom podignuta na visinu y_i od ishodišta (u skladu sa slikom 8.8). Za promatranu liniju želimo odrediti vidljivost pojedinih poligona duž te linije. Desno je prikazan presjek kroz promatrane poligone po dubini, odnosno po z-osi. Za trenutno promatranu liniju pretrage y_i desna slika predstavlja presjek dva promatrana poligona ravninom koja je okomita na xy-ravninu. U presjeku vidimo da se poligoni sijeku po dubini, odnosno duž z-osi. Ovdje je cilj odrediti raspone uzorka. *Raspon uzorka* definiramo kao dio linije pretrage na kojoj se ne može dogoditi promjena vidljivosti. Raspon uzorka je dio linije koji zadovoljava sljedeće uvjete:

1. broj segmenata u rasponu uzorka konstantan je i veći ili jednak jedan, te



Slika 8.8: Linija pretrage (lijevo) i rasponi uzorka (desno) u postupku Watkinsa

2. presjeci poligona s ravninom $y = y_i$ ne sijeku se unutar raspona uzorka (svakim sjecištem u presjeku započinje novi raspon uzorka).

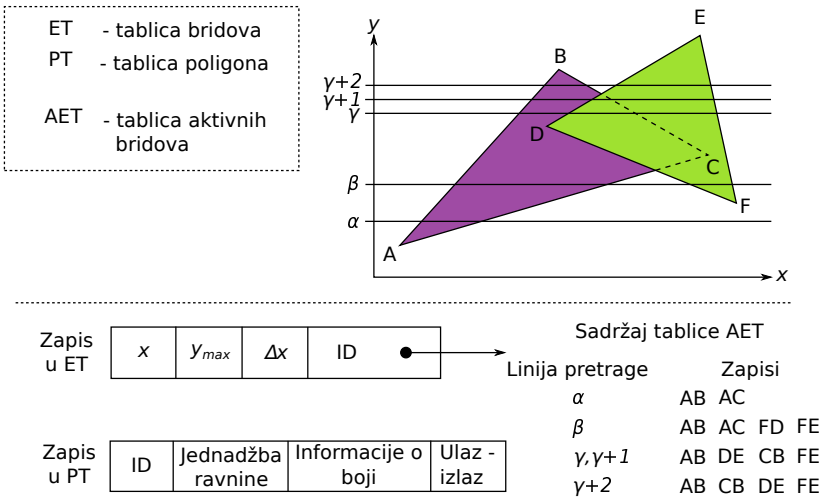
U primjeru na slici 8.8 desno označeni su rasponi uzorka. Promatrajući s lijeva na desno označen je raspon uzorka (a) dobiven na osnovi uvjeta (1) jer tu počinje prvi poligon promatrano s lijeva. U daljnjem dijelu vidimo da se poligoni po dubini sijeku pa su prema uvjetu (2) dobiveni rasponi uzorka (b) i (c), nakon toga završava prvi poligon s lijeva što određuje početak raspona uzorka označenog s (d). Kraj drugog poligona određuje završetak raspona uzorka (d).

Nakon određivanja raspona uzorka potrebno je odrediti vidljivost pojedinog raspona. Vidljivost se određuje na osnovi udaljenosti od promatrača. Ako je promatrač na slici 8.8 desno postavljen iz negativnog smjera z-osi na rasponu uzorka (a) i (b) bit će vidljiv prvi poligon, a na rasponu uzorka (c) i (d) drugi poligon.

Kada nije dostupna sklopovska grafička podrška za uklanjanje skrivenih linija i površina, ovaj algoritam je često bio korišten u programskoj izvedbi ostvarivanja prikaza (engl. *software renderers*). Vidimo da je osnovna ideja u tome da se ispitivanje ne provodi točku po točku, već je cilj iskoristiti kontekst, odnosno odrediti raspone za koje vrijedi svojstvo vidljivosti. Prilikom određivanja točku po točku dolazi do dodatnog opterećivanja dohvatom podataka, određivanja vidljivosti, slanjem podataka za svaku točku. Na starijim sustavima gdje nije postojala sklopovska podrška za grafiku, to je bilo toliko izraženo da je grupiranje podataka u veće cjeline (raspone) i grupiranje linija pretrage u blokove imalo drastičan učinak na brzinu ostvarivanja prikaza. Danas, na ugrađenim sustavima ovi koncepti ponovo imaju smisla, ali i proširenje koncepta na više dimenzija.

Sljedeći korak u razvoju algoritma je proširenje konteksta na susjedne linije pretrage i poligone koji čine prikaz. Stvara se tablica bridova (ET, engl. *Edge Table*), za sve ne-horizontalne bridove (slika 8.9). Pojedini zapis brida, sortiran po rastućoj y-koordinati, sadrži informaciju o:

- x-koordinati kraja koji ima manju y-koordinatu,
- y-koordinati drugog kraja brida,



Slika 8.9: Podatkovne strukture u postupku Watkinsa

- promjena po x, odnosno Δx promjena pri prelasku u sljedeću liniju pretrage, (recipročna vrijednost nagiba brida), te
- identifikator koji određuje kojem poligonu brid pripada.

Tablica poligona (PT, engl. *Polygon Table*) sadrži informaciju o:

- koeficijentima jednadžbe ravnine,
- zastavica koja određuje je li poligon aktivan (je li brid aktivan), te
- druge podatke, odnosno atribute kao što je boja, tekstura i sl.

Na slici 8.9 prikazana su dva projicirana poligona, a skriveni dijelovi prikazani su crtkano. Gradi se tablica aktivnih bridova (AET, engl. *Active Edge Table*) koja je prikazana na slici. Za liniju pretrage $y = \alpha$, aktivni su bridovi AB i AC jer tim redom slijede x-koordinate sjecišta s tim bridovima. Nailaskom na brid AB postavlja se zastavica poligona ABC, a prolaskom kroz brid AC se spušta. Gledano po y-koordinati to se ne mijenja sve do točke F. Za liniju pretrage $y = \beta$, aktivni su AB, AC, FD i FE. U ovom dijelu uvjek je aktivan u jednom trenutku samo jedan poligon i u prvom dijelu algoritma gdje određujemo raspone uzorka nije potreno ispitivati vidljivost već možemo koristiti informaciju s prethodne linije pretrage. Na slici nije posebno istaknut prijelaz kada se bridovi AC i FD sijeku, i time zamijene redoslijed u AET. Od tog trenutka kao i za $y = \gamma$, u jednom dijelu raspona bit će aktivna oba poligona istovremeno, gdje će onda ispitivanje z-koordinate odlučiti što je bliže. Ako se poligoni probadaju, to se može utvrditi na početku i linija koja određuje njihov presjek uvodi se kao poseban brid.

U nastavku ćemo dati malo i nešto detaljniji pseudokod ovog algoritma. Krenimo s potrebnim podatkovnim strukturama. Pri razmišljanju o ovom algoritmu potrebno je zapamtiti da algoritam scenu iscrtava jednu po jednu vodoravnu liniju, počevši od $y = y_{min}$ pa do $y = y_{max}$.

```
1  struct triangle_data_str;
2
3  // struktura za pamcenje jednog brida
4  typedef struct {
5      point3d *v1; // prvi vrh brida
6      point3d *v2; // drugi vrh brida
7      struct triangle_data_str *p_trokut; // pokazivac na trokut
8      int y; // minimalna y koordinata brida
9      int dy; // koliko puta ga linija pretrage sijece
10     double x, dx; // presjek s linijom pretrage te prirast x-a
11 } edge_data;
12
13 // struktura za pamcenje trokuta
14 typedef struct triangle_data_str {
15     int y; // minimalna y koordinata poligona
16     int dy; // koliko puta ga linija pretrage sijece
17     double a,b,c,d; // koeficijenti ravnine
18     f_rgb_color boja; // boja trokuta
19     bool aktivan; // je li aktivan
20     edge_data bridovi[3]; // bridovi
21 } triangle_data;
22
23 // podatci potrebni za Watkinsov algoritam
24 typedef struct {
25     list aktivni_trokuti; // popis aktivnih trokuta
26     list aktivni_bridovi; // popis aktivnih bridova
27     list** buckets;  // polje lista trokuta koji pocinju na y-u
28     double intervalL, intervalD; // Interval koji se ispituje
29     int xmin, xmax;       // raspon x koordinate ekrana
30     int ymin, ymax;       // raspon y koordinate ekrana
31     f_rgb_color boja_pozadine;     // boja pozadine
32 } watkins_data;
```

Struktura edge_data služi za pamćenje podataka o jednom bridu. Brid je određen točkama v1 i v2. Element p_trokut je pokazivač na trokut kojemu ovaj brid pripada (za potrebe algoritma pretpostavimo da se bridovi ne dijele). Kako bismo definirali sadržaj varijabli x, y, dx i dy, promotrimo točke v1 i v2, i pretpostavimo da vrijedi da nakon projekcije u xy-ravninu v1 ima manju y-koordinatu od v2. U slučaju da obje točke imaju jednaku y-koordinatu, tada vrijedi da je x-koordinata točke v1 nakon projekcije u xy-ravninu manja ili jednaka x-koordinati točke v2. Ako ovo nije zadovoljeno, dovoljno je zamijeniti točke v1 i v2. Na početku se element y postavlja na y-koordinatu točke v1 (niže točke), a element x na x-koordinatu točke v1. dx je tada $(v_{2,x} - v_{1,x})/(v_{2,y} - v_{1,y})$, odnosno što govori kako treba promijeniti sadržaj varijable x ako se y poveća za jedan (pri tome su x i y

koordinate one koje se dobiju nakon projekcije tih točaka u xy-ravninu). Element dy postavlja se na vrijednost $v_{2,y} - v_{1,y}$, što nam govori kroz koliko se različitih y-a (dakle, kroz koliko linija pretrage) proteže sam brid.

Struktura triangle_data sadrži podatke o jednom trokutu (algoritam je trivijalno proširiv na konveksne poligone). Element y sadrži y-koordinatu dna poligona, a element dy predstavlja visinu poligona (tj. broj linija pretrage koje prelaze preko poligona); ovaj podatak dobijemo kao razliku maksimalne i minimalne y-koordinate iz skupa vrhova poligona nakon projekcije u xy-ravninu. Elementi a, b, c i d predstavljaju koeficijente implicitnog oblika jednadžbe ravnine u kojoj leži trokut. Element boja predstavlja boju kojom je potrebno iscrtati trokut. Element aktivan koristit će se prilikom rada algoritma pa će biti pojašnjen uskoro. Konačno na kraju strukture nalazi se polje podataka o bridovima trokuta (prethodno opisana struktura edge_data).

Konačno, struktura watkins_data sadrži sve što je potrebno za rad algoritma. Element aktivni_trokuti je lista trokuta preko kojih prelazi trenutna linija pretrage. Slično, element aktivni_bridovi je lista bridova trokuta iz liste aktivni_trokuti preko kojih prelazi trenutna linija pretrage. Naime, kako bismo izbjegli potrebu da neprestano prolazimo kroz kompletan popis trokuta i bridova, i provjeravamo da li ih trenutna linija pretrage $y = y_i$ siječe ili ih možemo zanemariti, koristi se pametniji pristup. Već smo objasnili da se za svaki trokut pamti y-koordinata dna trokuta, a za svaki brid y-koordinata "nižeg" vrha. Da bismo ovo iskoristili, definirali smo pomoćno polje buckets koje ima onoliko elemenata koliko ima različitih y-vrijednosti između y_{min} i y_{max} (dakle, po jedan element za svaku liniju pretrage). i-ti element tada je lista svih trokuta koji počinju na $y = i$. Jednom kada se trokut doda u listu aktivnih trokuta, svakim podizanjem linije pretrage vrijednost njegove varijable dy umanjivat ćemo za jedan – trokut se izbacuje iz liste aktivnih trokuta onog trenutka kada mu dy padne na 0. Slično, za svaku novu liniju pretrage proći ćemo kroz popis bridova aktivnih trokuta i one bridove koji započinju na trenutnoj liniji pretrage dodat ćemo u listu aktivnih bridova. Svakim podizanjem linije pretrage bridovima iz liste aktivnih bridova umanjivat ćemo njihov dy i kada on padne na 0, brid ćemo izbaciti iz popisa aktivnih bridova.

Uočimo da uporabom ovakvog pristupa nikada ne moramo računati sjecište brida i trenutne linije pretrage. Naime, kada linija pretrage prvi puta dođe do brida, znamo da ga siječe u pohranjenjoj točki x (koju čuva struktura brida – to je bila x-koordinata nižeg od vrhova brida). Svakim podizanjem linije pretrage x brida povećat ćemo za dx, i time će nam x opet predstavljati korektnu x-koordinatu sjecišta. Na ovaj način štedi se na broju operacija računskih izračuna.

Varijable intervalL i intervalD predstavljaju pomoćne varijable – x-koordinate lijevog i desnog segmenta (na liniji pretrage) koji treba iscrtati. xmin, xmax, ymin i ymax predstavljaju raspon x i y-koordinata zaslona a boja_pozadine čuva boju kojom će se crtati segmenti koji ne pripadaju niti jednom trokutu.

```
1  void initialize_data(watkins_data *w, tablica *svijet) {
2      // Isprazni liste aktivnih trokuta i bridova
3      list_init(&w->aktivni_trokuti);
4      list_init(&w->aktivni_bridovi);
5
6      // definiraj velicinu ekrana i boju pozadine
7      w->ymin = 0; w->ymax = 299;
8      w->xmin = 0; w->xmax = 299;
9      w->boja_pozadine.r = 1.0f;
10     w->boja_pozadine.g = 1.0f;
11     w->boja_pozadine.b = 0.0f;
12
13     // stvori i inicijaliziraj buckets:
14     int broj_scan_linija = w->ymax-w->ymin+1;
15     w->buckets = (list**)malloc(broj_scan_linija*sizeof(list*));
16     for(int i = 0; i < broj_scan_linija; i++) {
17         w->buckets[i] = NULL;
18     }
19
20     // Za svaki trokut:
21     for(Trokut t : svijet->trokuti()) {
22         // Popuni podatke o trokutu:
23         triangle_data *td = fillTriangleData(t);
24         addToBucket(td, w->buckets, td->y);
25     }
26 }
```

Metoda initialize_data kao argumente prima pokazivač na strukturu watkins_data te pokazivač na strukturu tablica koja sadrži informacije o objektima i koju nećemo detaljno prikazivati; u ostatku koda koristimo je kao da pišemo pseudokod. Metoda initialize_data resetira liste aktivnih trokuta i bridova i stvara polje buckets. Za svaki trokut stvara se njegov opisnik (podatak tipa triangle_data). Računaju se koeficijenti implicitnog oblika jednadžbe ravnine u kojoj poligon leži, utvrđuju se minimalna i maksimalna y-koordinata vrhova poligona temeljem kojih se postavljaju y i dy. Element aktivan postavlja se na **false**. Za svaki brid popunjava se struktura edge_data. Konačno, trokut se dodaje u odgovarajući pretinac polja buckets prema svojoj najmanjoj y-koordinati.

```
1  void watkins_algorithm(tablica *svijet) {
2      int y;
3      watkins_data w;
4
5      // obavi inicijalizaciju potrebnih podataka
6      initialize_data(&w, svijet);
7
```

```
8       // Idemo za svaku liniju pretrage odozdo prema gore
9       for(y = w.ymin; y <= w.ymax; y++) {
10          // Dodaj trokute koji pocinju u buckets[y] u aktivne
11          list* l = w.buckets[y - w.ymin];
12          if(l!=NULL) {
13              for(int i = 0; i < l->size; i++) {
14                  list_add(&w.aktivni_trokuti, list_get(l, i));
15              }
16          }
17
18          // Dodaj sve bridove aktivnih trokuta koji pocinju na
19          // y u aktivne bridove
20          for(int i = 0; i < w.aktivni_trokuti.size; i++) {
21              triangle_data *td = list_get(&w.aktivni_trokuti, i);
22              for(int j = 0; j < 3; j++) {
23                  if(td->bridovi[j].y == y) {
24                      list_add(&w.aktivni_bridovi, &td->bridovi[j]);
25                  }
26              }
27          }
28
29          // Sortiraj aktivne bridove po x-u
30          for(int i = 0; i < w.aktivni_bridovi.size; i++) {
31              for(int j = i+1; j < w.aktivni_bridovi.size; j++) {
32                  int xj = list_get(&w.aktivni_bridovi,j))->x;
33                  int xi = list_get(&w.aktivni_bridovi,i))->x;
34                  if(xj < xi) {
35                      list_swap(&w.aktivni_bridovi,i,j);
36                  }
37              }
38          }
39
40          // Obradi aktivne bridove
41          process_active_edges(&w, y);
42          // Azuriraj aktivne bridove
43          update_active_edges(&w);
44          // Azuriraj aktivne trokute
45          update_active_triangles(&w);
46      }
47      free_watkins_data(&w);
48  }
```

Metoda watkins_algorithm predstavlja glavnu metodu Watkinsovog algoritma. Nakon što se napravi inicijalizacija u kojoj se stvore sve potrebne podatkovne strukture, algoritam pomiče liniju pretrage od najmanjeg y-a do najvećeg dozvoljenog y-a. Za svaku liniju pretrage, u listu aktivnih trokuta dodaju se trokuti koji počinju na toj liniji pretrage (uporabom informacija iz polja buckets). Potom se prolazi kroz bridove aktivnih trokuta, i svi bridovi koji počinju na liniji pretrage dodaju se u listu aktivnih bridova. Lista aktivnih bridova potom se sortira prema trenutnoj x-koordinati sjecišta s linijom pretrage (pohranjeno u varijabli

x svakog brida). Ovo je važno jer se podizanjem linije pretrage može promijeniti redosljed sjecišta bridova i linije pretrage. Nakon što su aktivni bridovi sortirani, prelazi se na njihovu obradu (tu se obavlja crtanje vodoravnih segmenata), te potom na ažuriranje liste aktivnih bridova i trokuta. Krenimo od ove posljednje dvije metode.

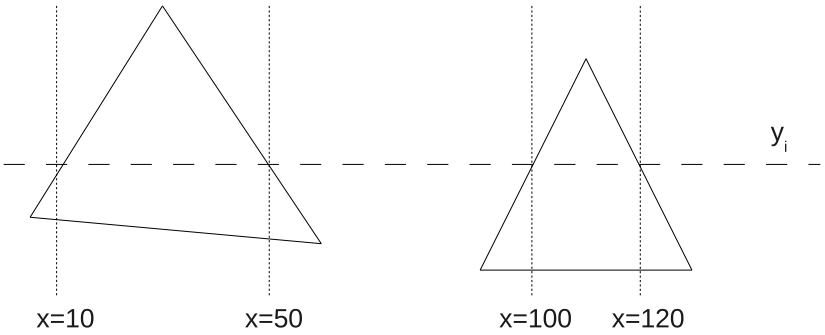
```
1  void update_active_edges(watkins_data *w) {
2      for(int i = 0; i < w->aktivni_bridovi.size; i++) {
3          edge_data *e = list_get(&w->aktivni_bridovi, i);
4          e->dy = e->dy - 1;
5          if(e->dy <= 0) {
6              list_delete(&w->aktivni_bridovi, i);
7              i--;
8          } else {
9              e->x = e->x + e->dx;
10         }
11     }
12 }
13
14 void update_active_triangles(watkins_data *w) {
15     for(int i = 0; i < w->aktivni_trokuti.size; i++) {
16         triangle_data *t = list_get(&w->aktivni_trokuti, i);
17         t->dy = t->dy - 1;
18         if(t->dy <= 0) {
19             list_delete(&w->aktivni_trokuti, i);
20             i--;
21         }
22     }
23 }
```

Obje metode, update_active_edges i update_active_triangles već smo načelno opisali: umanjuju dy za jedan, i ako se dođe na nulu, brid odnosno trokut izbacuju se iz liste aktivnih.

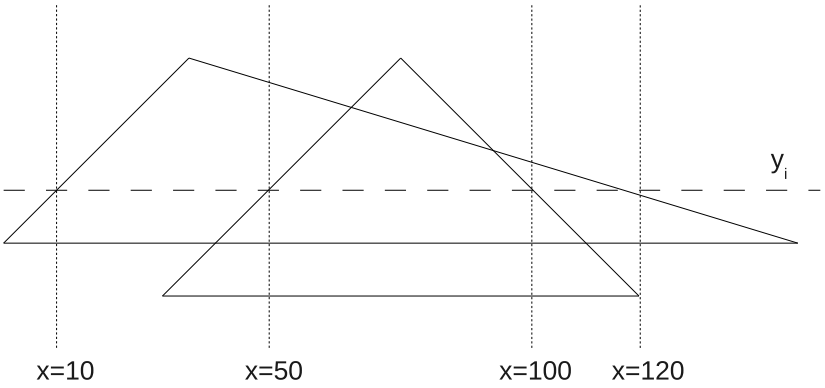
```
1  void process_active_edges(watkins_data *w, int y) {
2      f_rgb_color seg_boja; // boja intervala
3      int broj_trokuta; // koliko je trokuta u segmentu?
4      edge_data *e;      // brid
5      triangle_data *t; // trokut
6
7      broj_trokuta = 0;
8      w->intervalL = w->xmin; // pocetak intervala je na xmin
9
10     // Srusi zastavicu aktivan svih aktivnih trokuta
11     for(int i = 0; i < w->aktivni_trokuti.size; i++) {
12         t = list_get(&w->aktivni_trokuti, i);
13         t->aktivan = false;
14     }
15
16     // Idemo za svaki brid (sortirani su prema x-u sjecista
17     // s linijom pretrage)
```

```
18      for(int i = 0; i < w->aktivni_bridovi.size; i++) {
19          // Dohvati brid
20          e = (edge_data*)list_get(&w->aktivni_bridovi, i);
21          // Zatvori interval njegovim sjecistem
22          w->intervalD = e->x;
23          // Koliko je trokuta u podrucju tog segmenta?
24          switch(broj_trokuta) {
25              // ako nula, popuni pozadinom
26              case 0:
27                  seg_boja = w->boja_pozadine;
28                  break;
29              // ako je jedan, uzmi boju aktivnog trokuta
30              case 1:
31                  seg_boja = color_of_active_triangle(w);
32                  break;
33              // ako je vise, utvrdi boju najblizeg
34              default:
35                  seg_boja = color_for_segment(w, y);
36                  break;
37          }
38
39          // Promijeni zastavicu aktivan pripadnog trokuta
40          t = e->p_trokut;
41          t->aktivan = !t->aktivan;
42          // i azuriraj broj aktivnih trokuta
43          if(t->aktivan) {
44              broj_trokuta = broj_trokuta + 1;
45          } else {
46              broj_trokuta = broj_trokuta - 1;
47          }
48
49          // nacrtaj vodoravni segment linije pretrage
50          display_interval(w->intervalL, w->intervalD,
51              y, &seg_boja);
52
53          // zapocni novi interval s krajem trenutnog
54          w->intervalL = w->intervalD;
55      }
56
57      // ako je ostao neiscrtan kraj linije pretrage:
58      if(w->intervalL < w->xmax) {
59          display_interval(w->intervalL, w->xmax,
60              y, &w->boja_pozadine);
61      }
62  }
```

Metoda process_active_edges zadužena je za iscrtavanje vodoravnih segmenata jedne linije pretrage. Metoda promatra sjecišta s aktivnim bridovima i temeljem toga utvrđuje veličinu segmenta i njegovu boju. Primjerice, pretpostavimo da za trenutnu liniju pretrage postoje 4 aktivna brida, čija su sjecišta s linijom pretrage $x = 10$, $x = 50$, $x = 100$ i $x = 120$. Uz dimenzije ekrana po x-osi od $x = 0$ do



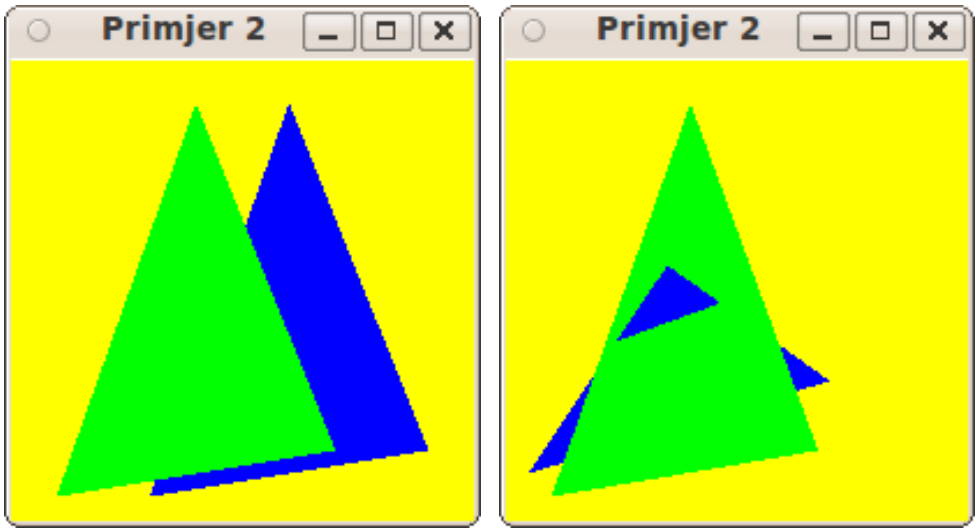
Slika 8.10: Watkinsov postupak: dva disjunktna trokuta



Slika 8.11: Watkinsov postupak: preklapajući trokuti

$x = 299$, očekujemo 5 segmenata: prvi S_1 od $x = 0$ do $x = 10$, drugi S_2 od $x = 10$ do $x = 50$, treći S_3 od $x = 50$ do $x = 100$, četvrti S_4 od $x = 100$ do $x = 120$ te konačno peti S_5 od $x = 120$ do $x = 299$. Segment S_1 bojat ćemo pozadinskom bojom - naime, on se proteže od ruba ekrana, pa do sjecišta linije pretrage s prvim aktivnim bridom; varijabla broj_trokuta u tom segmentu bit će 0. Nailaskom na prvo sjecište (dakle, obradom prvog aktivnog brida), njegov trokut također postaje aktivan – sljedeći segment prelazit će preko tog trokuta. Zbog toga ćemo i broj_trokuta povećati za 1. Obradom drugog brida, odnosno dolaskom do sjecišta $x = 50$, razmatramo novi interval: S_2. Ovdje su već moguća dva slučaja: ako taj brid pripada istom trokutu (kao i maloprije, vidi sliku 8.10), znači da tim sjecištem segment izlazi iz područja tog trokuta. U tom slučaju zastavicu aktivan treba spustiti i vrijednost varijable broj_trokuta smanjiti za 1. Međutim, ako brid pripada nekom drugom trokutu (vidi sliku 8.11), tada ili ulazimo i u njegovo područje – a već jesmo na području onog starog trokuta, čime i njegovu zastavicu aktivan treba postaviti na **true** i broj_trokuta uvećati za jedan, ili izlazimo iz njegovog područja, pa mu zastavicu aktivan treba postaviti na **false** i broj_trokuta umanjiti za jedan. U općem slučaju, dakako, moguće je da se linija pretrage nalazi na području niti jednog, jednog ili proizvoljno mnogo trokuta. Ako se linija pretrage nalazi na području primjerice dva trokuta, tada su opet moguća dva slučaja. U jednostavanom slučaju, jedan se trokut nalazi ispred drugog trokuta, i segment će biti bojan bojom onog koji je promatraču bliži. Međutim, moguć je slučaj da se ta dva trokuta u dijelu promatranog segmenta probadaju, pa segment treba podijeliti na onaj dio u kojem je vidljiv jedan trokut, te na ostatak u kojem je vidljiv drugi trokut. U općem slučaju, s više trokuta, ovih podjela može biti još i više. Pa imajući to u vidu, objasnimo što se događa u metodi process_active_edges. Slike 8.12a i 8.12b prikazuju rezultat iscrtavanja Watkinsovim postupkom.

Metoda započinje postavljanjem lijeve granice segmenta na x-koordinatu početka ekrana. Svim trokutima iz liste aktivnih trokuta zastavica aktivan se postavlja na **false**. broj_trokuta postavlja se na 0. Potom se gleda brid po brid iz liste aktivnih bridova. x-koordinata sjecišta promatranog brida s linijom pretrage zatvara trenutni segment. Promatra se broj_trokuta koji se nalazi ispod segmenta. Ako je to 0, bira se pozadinska boja za popunjavanje. Ako je to 1, pretražuje se lista aktivnih poligona, i vraća boja onog (jedinog) koji ima postavljenu zastavicu aktivan na **true**. Konačno, ako je to 2 ili više (slučaj **default**), poziva se pomoćna metoda koja će proanalizirati koji je slučaj nastupio, po potrebi će segment podijeliti na manje podsegmente i sve ih iscrtati osim zadnjeg, čiju će boju vratiti. Nakon toga, mijenjamo zastavicu aktivan trokuta kojemu pripada trenutni brid, te ako smo ga aktivirali, povećavamo, a ako smo ga deaktivirali, smanjujemo broj_trokuta. Potom crtamo segment utvrđenom bojom, i započinjemo novi segment od kraja trenutnog segmenta.



(a) jednostavniji slučaj (b) trokuti koji se probadaju

Slika 8.12: Watkinsov postupak

```
1  f_rgb_color color_of_active_triangle(watkins_data *w) {
2      for(int i = 0; i < w->aktivni_trokuti.size; i++) {
3          triangle_data *t = list_get(&w->aktivni_trokuti, i);
4          if(t->aktivan) {
5              return t->boja;
6          }
7      }
8      // Ovdje ne smijemo stici!
9      return w->boja_pozadine;
10 }
11
12 f_rgb_color color_for_segment(watkins_data *w, int y) {
13     // Stog dovoljno velik da primi sva sjecista...
14     int broj_sjecista =
15         w->aktivni_trokuti.size*w->aktivni_trokuti.size;
16     float* stog = (float*)malloc(broj_sjecista*sizeof(float));
17     int stog_size;
18     bool imam_presjek;
19     float x_presjeka;
20     f_rgb_color interval_boja;
21
22     x_presjeka = w->intervalL;
23     stog_size = 0;
24     while(1) {
25         checkForIntersectionsInSegment(
26             w, &imam_presjek, &x_presjeka, y);
27         if(imam_presjek) {
```

```
28              stog[stog_size++] = w->intervalD;
29              w->intervalD = x_presjeka;
30          } else {
31              interval_boja =
32                  color_of_min_active_z_value_triangle(
33                      w, (w->intervalL+w->intervalD)/2.0, y);
34              if(stog_size==0) {
35                  free(stog);
36                  return interval_boja;
37              }
38              display_interval(w->intervalL, w->intervalD,
39                  y, &interval_boja);
40              w->intervalL = w->intervalD;
41              w->intervalD = stog[--stog_size];
42          }
43      }
44  }
```

Metoda color_of_active_triangle vraća iz liste aktivnih trokuta boju trokuta koji ima postavljenu zastavicu aktivan na **true**. Pretpostavka je da takav postoji samo jedan. Metoda color_for_segment koristi se za analizu složenijih slučajeva, gdje se u području segmenta nalazi više trokuta. Ideja metode jest skraćivati segment koji je početno zadan varijablama w->intervalL i w->intervalD na manji segment, tako dugo dok se ne dođe do situacije da se u promatranom segmentu više ne događa probadanje trokuta – pa točno možemo utvrditi koji je trokut u tom segmentu iznad kojeg trokuta. Evo kako se to radi. Metoda ulazi u petlju i traži postoji li u trenutnom segmentu (što je na početku onaj originalni, no u sljedećem prolazu to već može biti i skraćeni segment) sjecište bridova aktivnih trokuta. Ako postoji, trenutni kraj segmenta gura na stog (kako bi se kasnije moglo nastaviti), a kao kraj trenutnog segmenta uzima to sjecište. Postupak se ponavlja s tako skraćenim segmentom.

Kada se utvrdi da u promatranom segmentu više nema sjecišta, kao boja za segment se uzima boja onog aktivnog trokuta iz liste aktivnih trokuta koji je u tom segmentu najbliži promatraču. Ovo se jednostavno ispita: naime, budući da znamo jednadžbu ravnine u kojoj trokut leži, kao x koordinatu uzmemo sredinu segmenta, y je zadan linijom pretrage, i iz jednadžbe ravnine očitamo pripadnu z-koordinatu. Ako je u tom trenutku stog prazan, vraćamo tu boju, i puštamo pozivatelja da iscrta taj segment. Ako stog nije prazan, tada metoda crta pronađeni podsegment, te započinje novi segment koji počinje na mjestu gdje je trenutni završio, a kraj se vadi sa stoga. S tim novim segmentom postupak se ponavlja. Metoda koja pronalazi boju najbližeg aktivnog trokuta prikazana je u nastavku.

```
1  f_rgb_color color_of_min_active_z_value_triangle(
2      watkins_data *w, float x, int y) {
3
4      f_rgb_color *boja = NULL;
```

```
5       double z = 0.0;
6
7       for(int i = 0; i < w->aktivni_trokuti.size; i++) {
8           triangle_data *t = list_get(&w->aktivni_trokuti, i);
9           if(t->aktivan) {
10              double z_trenutni = -(t->a*x+t->b*y+t->d)/t->c;
11              // gledam izishodista prema -z; veci z mi je blizi
12              if(boja==NULL || z_trenutni > z) {
13                  boja = &t->boja;
14                  z = z_trenutni;
15              }
16          }
17      }
18      if(boja==NULL) {
19          printf("Greska: nema boje.\n");
20      }
21      return boja==NULL ? w->boja_pozadine : *boja;
22  }
```

Konačno, metoda koja traži sjecišta opisana je u nastavku, u malo naglašenijem pseudokodu.

```
1   void checkForIntersectionsInSegment(
2       watkins_data *w, bool *imam_presjek,
3       float *x_presjeka, int y) {
4
5       za_svaki_aktivni_trokut t {
6           za_svaki_aktivni_brid t.b {
7               (x,z) = brid_point(t.b, y);
8           }
9       }
10
11      // sada (potencijalno) za svaki trokut imamo segment odreden
12      // s dvije tocke koje se nalaze na dva aktivna brida kroz
13      // koje prolazi linija pretrage
14
15      // pogledaj sjecista tih segmenata (voditi racuna da to nisu
16      // pravci, vec dijelovi pravaca
17
18      // ako postoji sjeciste cija je x-koordinata x_sjec i ako za
19      // nju vrijedi: w->intervalL < x_sjec < w->intervalD
20      if(postoji_opisano_sjeciste) {
21          *imam_presjek = true;
22          *x_presjeka = x_sjec;
23      } else {
24          *imam_presjek = false;
25          *x_presjeka = w->intervalL;
26      }
27  }
```

Metoda checkForIntersectionsInSegment programski je najzahtjevnija, no konceptualno je vrlo jednostavna. Zato je odabran opis koji je više kroz komentare a

manje programerski čist. Ideja je najprije pronaći za sve aktivne bridove točke u kojima ih linija pretrage siječe. Kako znamo y-koordinatu, x i z-koordinate možemo dobiti direktno iz parametarskog oblika jednadžbe brida. Za svaki trokut, općenito govoreći, postojat će ili nula, ili dva takva sjecišta. Ako radimo samo s aktivnim bridovima, onda ćemo sigurno imati dva sjecišta (oba uz isti y), i te dvije točke u xz-ravnini određuju segment pravca. Nakon što pronađemo sve segmente, kod za svaki par segmenata provjerava sijeku li se, i ako da, pada li x-koordinata u granice trenutnog segmenta koji želimo iscrtati. Ako takvo sjecište postoji, može se vratiti odmah prvo; naime, sjetimo se da će pozivatelj to sjecište iskoristiti da skrati interval, i potom na kraćem intervalu opet pozvati ovu provjeru. Za kraj je ostalo još prikazati i najjednostavniju metodu - onu koja uporabom *OpenGL*-a radi crtanje vodoravnog segmenta.

```
1  void display_interval(float x0, float x1,
2       float y, f_rgb_color *boja) {
3
4       glColor3f(boja->r, boja->g, boja->b);
5       glBegin(GL_LINE_STRIP);
6       glVertex2f(x0, y);
7       glVertex2f(x1, y);
8       glEnd();
9  }
```

Primjer: 13

Zadana su dva trokuta u 3D prostoru. Vrhovi prvog su: $A = (\ 20 \quad 10 \quad -10\ )$, $B = (\ 80 \quad 180 \quad -20\ )$ i $C = (\ 135 \quad 30 \quad -15\ )$; vrhovi drugog su: $D = (\ 10 \quad 20 \quad -30\ )$, $E = (\ 70 \quad 110 \quad -5\ )$ i $F = (\ 140 \quad 60 \quad -40\ )$. Prvi trokut je zelen, drugi je plavi. Boja pozadine je žuta. Provedite Watkinsov postupak za liniju pretrage y=80. Pretpostavite da se slika iscrtava na pravokutnom području (0,199) po obje koordinate.

Rješenje:

Kako se u primjeru zahtjeva izračun samo za jednu liniju, ne znamo kakvo je stanje u tablicama aktivnih bridova i trokuta – naime, postupak nismo provodili od početka. Pa idemo najprije rekonstruirati to stanje. Bridovi i trokuti s kojima radi algoritam, te početne vrijednosti koje se za njih pamte prikazane su u tablicama u nastavku.

Brid	y	dy	x	dx	Trokut
AB	10	170	20	0.352941	T1
BC	30	150	135	-0.366667	T1
CA	10	20	20	5.75	T1
DE	20	90	10	0.666667	T2
EF	60	50	140	-1.4	T2
FD	20	40	10	3.25	T2

Trokut	y	dy	a	b	c	d
T1	10	170	-650	-850	-18350	-162000
T2	20	90	-1900	3850	-9300	-337000

Za $y = 80$ vidimo da će oba trokuta biti u listi aktivnih trokuta (prvi počinje na $y = 10$ i proteže se sljedećih 170 linija pretrage, a drugi počinje na $y = 20$ i proteže se sljedećih 90 linija pretrage; dakle, na $y = 80$ oba su u listi aktivnih trokuta). Od bridova, u listi aktivnih bridova bit će AB, BC, DE i EF. Sjecišta tih bridova s linijom pretrage $y = 80$ prikazana su u sljedećoj tablici.

Brid	Pripada trokutu	$x_{sjec} = b.x + (y - b.y) \cdot b.dx$
AB	T1	$20 + (80 - 10) \cdot 0.352941 = 44.7059$
BC	T1	$135 + (80 - 30) \cdot -0.366667 = 116.6667$
DE	T2	$10 + (80 - 20) \cdot 0.666667 = 50.0000$
EF	T2	$140 + (80 - 60) \cdot -1.4 = 112.0000$

Alternativno, sjecišta x-koordinatu sjecišta mogli smo direktno računati iz parametarskog oblika jednadžbi bridova, tako da najprije temeljem poznatog y-a odredimo parametar, a zatim za taj parametar odredimo preostale koordinate. Nakon što smo definirali sadržaj liste aktivnih bridova, bridove je potrebno sortirati prema x-koordinati sjecišta. Tada dobivamo tablicu prikazanu u nastavku.

Brid	Pripada trokutu	x_{sjec}
AB	T1	44.7059
DE	T2	50.0000
EF	T2	112.0000
BC	T1	116.6667

Algoritam kreće s broj_trokuta=0 te intervalL=0.

Korak 1. Gleda se prvi brid (AB), i kraj segmenta se postavlja na intervalD=44.7059. Kako je broj_trokuta=0, za popunjavanje segmenta bira se pozadinska boja (žuta). Postavlja se zastavica T1.aktivan=**true** i varijabla broj_trokuta=1. Boja se interval $(0,\ 44.7059)$ žutom. Početak intervala postavlja se na intervalL=44.7059.

Korak 2. Gleda se drugi brid (DE), i kraj segmenta se postavlja na 50.0000; broj_trokuta=1 pa se traži jedini trokut koji ima postavljenu zastavicu aktivan (to će biti T1), i za popunjavanje se bira njegova boja (zelena). Postavlja se T2.aktivan=**true** i povećava se broj_trokuta=2. Boja se interval $(44.7059,\ 50.0000)$ zelenom. Početak intervala postavlja se na intervalL=50.0000.

Korak 3. Gleda se treći brid (EF). Kako je broj_trokuta=2, provjerava se postoji li u intervalu $x \in (50, 112)$ kakvo sjecište. Presjek prvog trokuta i ravnine $y = 80$ je segment pravca u xz-ravnini (44.705883,-14.117647)-(116.666664,-16.666666). Presjek drugog trokuta i ravnine $y = 80$ je segment pravca u xz-ravnini (50.000000,-13.333333)-(112.000000,-26.000000). Ova dva segmenta sijeku se u točki koja ima $x = 55.7547$. Kako je ovo unutar promatranog intervala, vraća se pronađeno sjecište. Zbog toga se aktualni kraj intervala gura na stog (112.0000), a kao novi kraj se postavlja pronađeno sjecište 55.754688. Ponovno se provjerava postoji li kakvo sjecište unutar tog skraćenog intervala, i odgovor je ne. Tada se kao testna točka uzima sredina intervala $(50 + 55.7547)/2 = 52.8773$. Za tu točku ($x = 52.8773$, $y = 80$) računa se pripadna z-koordinata u ravnini svakog od aktivnih trokuta, i vraća se boja onog koji je najbliži promatraču (trokut čija točka ima najveću z-koordinatu – promatrač je u ishodištu). U ovom slučaju to će biti plava boja. Kako stog nije prazan, crta se trenutni interval (50,55.7547) plavom bojom. Početak intervala postavlja se na trenutni kraj intervalL=55.7547, a kraj se vadi sa stoga intervalL=112.0000. Za zadani interval provjerava se postoji li koje sjecište u xz-ravnini koje pada unutar tog intervala (rubovi su isključeni). Kako takvo sjecište ne postoji (bridovi se sijeku u $x = 55.7547$), traži se trokut za koji je središnja točka intervala $x = (55.7547 + 112)/2 = 83.8774$, $y = 80$ najbliža promatraču (ima najveću z-koordinatu) i uzima se njegova boja i vraća (to će biti zelena). Ovime smo ponovno u glavnoj metodi koja sada vidi skraćeni segment (55.7547,112) i ima za njega zelenu boju. Kako trenutni brid pripada trokutu T2, mijenja se njegova zastavica T2.aktivan=**false** i umanjuje broj_trokuta=1. Boja se interval $(55.7547,\ 112.0000)$ zelenom. Početak intervala postavlja se na intervalL=112.0000.

Korak 4. Gleda se četvrti brid (BC), i kraj segmenta se postavlja na 116.6667; broj_trokuta=1 pa se traži jedini trokut koji ima postavljenu zastavicu aktivan (to će biti T1), i za popunjavanje se bira njegova boja (zelena). Mijenja se T1.aktivan=**false** i umanjuje se broj_trokuta=0. Boja se interval $(112.0000,\ 116.6667)$ zelenom. Početak intervala postavlja se na intervalL=116.6667.

Korak 5. Kako više nema bridova, izlazi se iz petlje. Kraj segmenta postavlja se na desni rub ekrana (199), i segment (116.6667, 199) boja se pozadinskom bojom (žutom). Ovime završava obrada te linije pretrage.

8.5 Z-spremnik

Z-spremnik (engl. *z-buffer*) je postupak koji provjeru preklapanja radi na razini slikovnih elemenata. Ideja je sljedeća. Slika se iscrtava na zaslon konačnih di-

menzija. Tada se prilikom iscrtavanja slikovnog elementa na zaslon u z-spremniku pamti udaljenost izvorne točke od promatrača (z-koordinata točke nakon projekcije). Npr. potrebno je iscrtati točku (x,y,z). Radi se projekcija točke i dobije se točka (x',y',z'). Na zaslon treba iscrtati slikovni element na poziciji (x',y'). No prije nego što nacrtamo taj slikovni element, potrebno je u z-spremniku pogledati je li na tu poziciju već nacrtan neki drugi slikovni element, i ako je, koliko je pripadna točka bila daleko od promatrača. Ako je nova točka bliža, tada skriva prethodno iscrtanu točku (jer je bliža promatraču), pa je crtamo i u z-spremnik upisujemo na mjesto (x',y') udaljenost te točke. Ako je naša točka dalja, tada je ne crtamo, i sadržaj z-spremnika ne mijenjamo.

U praktičnim izvedbama polje dimenzija rezolucija$_x$ × rezolucija$_y$ koristi se za implementaciju z-spremnika. Hoće li to biti polje tipa **short**, **int** ili nešto treće, ovisi o postavkama. Inicijalno se svi elementi polja postave na najveće vrijednosti (u smislu razmišljanja: do sada smo "iscrtavali" jedino točke u beskonačnosti koje su iza svih drugih). Prostor kojeg smo odredili prednjom i stražnjom ravninom odsijecanja određuje raspon koji će se preslikati na raspon z-spremnika. Ovdje treba pripaziti na moguće pogreške. Ako na primjer odredimo prednju ravninu odsijecanja na 1, stražnju na 1000, a naš objekt ima z koordinatu u rasponu od 1.2 - 1.3 očito je da nismo dobro odredili raspon spremnika i da će pogreške uslijed zaokruživanja utjecati na rezultat. Sklopovska podrška za z-spremnik već je dulje vrijeme nezaobilazna na računalima i svakako je treba koristiti. U OpenGL-u prvo je potrebno inicijalizirati spremnik, pa ako koristimo i dvostruki spremnik opisan u prvom poglavlju potrebno je:

```
1  glutInitDisplayMode(GLUT_DOUBLE|GLUT_DEPTH);
```

Dalje je potrebno omogućiti korištednje spremnika dubine i odabrati uvjet ispitivanja. Ako je inicijalno u spremniku najveća vrijednost tada ćemo provjeravati je li nešto bliže od te postavljene vrijednosti. Z-spremnik možemo koristiti i za razne druge namjene pa su i različiti uvjeti koji mogu biti postavljeni pri navedenoj provjeri. Treba još obratiti pažnju da prilikom brisanja spremnika s bojom obrišemo i z-spremnik. Ako to ne učinimo iscrtavanje pomaknutog objekta obavljat će se uz stare neispravne vrijednosti u z-spremniku pa će i prikaz animacije biti vrlo neobičan.

```
1  glEnable(GL_DEPTH_TEST);
2  glDepthFunc(GL_LEQUAL);
3  ...
4  void display()
5  { ...
6  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
7  ...
8  }
```

Najveći nedostatak ovakvog pristupa je u zahtjevima za memorijom. Ako radimo na nekom ugrađenom sustavu ili sustavu na kojem imamo memorijska

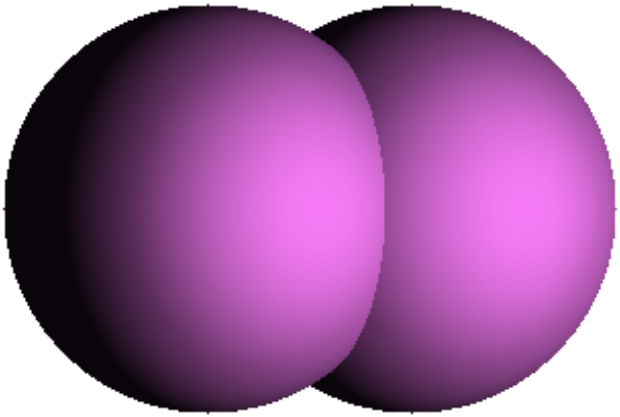
ograničenja posebnu pažnju morat ćemo posvetiti zadanim ograničenjima. Npr. odlučimo li se na prikaz u rezoluciji 1024×768 piksela, uz maksimalnu dubinu scene takvu da možemo koristiti tip **short** (2 okteta po podatku), potrebna količina memorije iznosi $1024 \cdot 768 \cdot 2 = 1572864$ okteta, što je poprilici 1.5 MB. Zahtjevi za memorijom mogu još porasti ako u z-spremnik odlučimo upisivati više informacija. Naime, ako ga koristimo za spremanje dubine, a točke iscrtavamo na zaslonu, tada ćemo kod vrlo kompleksnih scena kojima treba dosta vremena za iscrtavanje uočiti kako se neki pikseli pojavljuju, pa ih zamjenjuju drugi pikseli dobiveni od bližih objekata. Da bi se ovo izbjeglo, z-spremnik možemo koristiti tako da pamti i dubinu pojedine točke, i njezinu boju (samo se iscrtavanje u ovom koraku ne radi na zaslon). Jednom kada smo čitavu scenu iscrtali u z-spremnik, čitavu sliku možemo u jednom prolazu iz z-spremnika osvježavati na zaslonu. No ovakav postupak za svaku točku z-spremnika zahtjeva 2 okteta za dubinu + 3 okteta za boju što daje 5 okteta po točki, ili uz rezoluciju 1024×768 iznos od 3932160 okteta odnosno 3.75 MB.

Ove značajne memorijske zahtjeve možemo ublažiti višeprolaznim iscrtavanjem scene: npr. u prvom prolazu iscrtati ćemo gornju polovicu zaslona, a u drugom prolazu donju polovicu. Tada nam treba z-spremnik veličine za pola zaslona. Općenito, n-prolazno iscrtavanje zahtjeva samo n-ti dio z-spremnika koji bi trebao za jednoprolazno iscrtavanje, ali zato postupak traje n puta dulje. Razvijen je i *scan line z-buffer* algoritam koji iscrtava liniju po liniju zaslona. Međutim, efikasna primjena ovog algoritma zahtjeva velike izmjene u načinu pamćenja objekata.

Kako bismo ilustrirali uporabu Z-spremnika, u nastavku ćemo najprije pokazati dio programa koji će osjenčati dvije kugle koje se međusobno probadaju koristeći Phongov model i našu vlastitu implementaciju z-spremnika. Više o Phongovom modelu bit će riječi u sljedećem poglavlju. Rezultat rada prikazan je na slici 8.13 a izvorni kod naveden je u nastavku. Centar desne kugle nalazi se nešto ispod centra lijeve kugle. Konkretno, lijeva kugla ima centar u točki (150,150,0), dok desna kugla ima centar u točki (250,150,−50).

Ispis 8.1: Isječak bitni dijelova koda za vlastitu izvedbu z-spremnika s primjerom crtanja dvije kugle

```
1  // Struktura tocke / 3D vektora
2  typedef struct {
3      double x;
4      double y;
5      double z;
6  } Point3D;
7
8  // Normiranje vektora
9  void normalize(Point3D *pnt) {
10     double norm = sqrt(
11         pnt->x*pnt->x + pnt->y*pnt->y + pnt->z*pnt->z);
```



Slika 8.13: Dvije kugle prikazane uporabom z-spremnika

```
12      pnt->x = pnt->x / norm;
13      pnt->y = pnt->y / norm;
14      pnt->z = pnt->z / norm;
15  }
16
17  // Skalarni produkt dvaju vektora
18  double scalarProduct(Point3D *a, Point3D *b) {
19      return a->x*b->x + a->y*b->y + a->z*b->z;
20  }
21
22  // Pomice predanu tocku za trazeni vektor
23  double add(Point3D *p, Point3D *delta) {
24      p->x += delta->x;
25      p->y += delta->y;
26      p->z += delta->z;
27  }
28
29  // Struktura z-spremnika
30  typedef struct {
31      int w;
32      int h;
33      short *buffer;
34  } z_buffer;
35
36  // Konstruktor z-spremnika
37  z_buffer *new_z_buffer(int w, int h) {
```

```
38      z_buffer *z_buf = (z_buffer*)malloc(sizeof(z_buffer));
39      if(z_buf==NULL) return NULL;
40      z_buf->buffer = (short*)malloc(w*h*sizeof(short));
41      if(z_buf->buffer==NULL) {
42          free(z_buf);
43          return NULL;
44      }
45      z_buf->w = w;
46      z_buf->h = h;
47      return z_buf;
48  }
49
50  // Oslobadanje z-spremnika
51  void free_z_buffer(z_buffer *z_buf) {
52      free(z_buf->buffer);
53      free(z_buf);
54  }
55
56  // Brisanje sadrzaja z-spremnika
57  void clear_z_buffer(z_buffer *z_buf) {
58      int x,y;
59
60      for(y=0; y<z_buf->h; y++) {
61          for(x=0; x<z_buf->w; x++) {
62              z_buf->buffer[y*z_buf->w+x] = -32000;
63          }
64      }
65  }
66
67  // Provjera smije li se iscrtati predana tocka; ako da, njezina
68  // z-koordinata odmah ce se upisati i u sam spremnik.
69  bool can_update_z_buffer(z_buffer *z_buf, Point3D *p) {
70      int pomak = z_buf->w*zaokruzi(p->y) + zaokruzi(p->x);
71      int z = zaokruzi(p->z);
72      bool result = z > z_buf->buffer[pomak];
73      if(result) {
74          z_buf->buffer[pomak] = z;
75      }
76      return result;
77  }
78
79  // Metoda za crtanje kugle, uz pretpostavku da se promatra iz
80  // smjera pozitivne z-osi.
81  void bojajKuglu(const int R, Point3D *cntr, Point3D *l,
82          Point3D *v, z_buffer *z_buf) {
83      int x, y; // indeksi petlje
84      double Id, Is; // difuzna i reflektirajuca komponenta
85      float I; // ukupni intenzitet
86      double ln_scalar; // pomocna varijabla
87      Point3D p; // tocka na kugli
88      Point3D n; // normala u tocki p
```

```
89      Point3D r; // vektor reflektirane komponente
90
91      glBegin(GL_POINTS);
92      for( x=-R; x<=R; x++ ) {
93          for( y=-R; y<=R; y++ ) {
94              // odredi tocku ishodisne kugle
95              p.x = x; p.y = y;
96              p.z = R*R - (p.x*p.x + p.y*p.y);
97              if( p.z < 0 ) continue;
98              p.z = sqrt(p.z);
99              // odredi normalu
100             n = p;
101             normalize(&n);
102             // racunaj difuznu komponentu
103             ln_scalar = scalarProduct(l, &n);
104             Id = ln_scalar;
105             if( Id > 0.0 ) Id = 200*Id; else Id = 0.0;
106             // racunaj reflektirajucu zraku i komponentu
107             r.x=2*ln_scalar*n.x-l->x;
108             r.y=2*ln_scalar*n.y-l->y;
109             r.z=2*ln_scalar*n.z-l->z;
110             normalize(&r);
111             Is = scalarProduct(&r, v);
112             if( Is > 0.0 ) {
113                 Is = 45*pow(Is,1.1);
114             } else Is=0.0;
115             // ukupni intenzitet
116             I = (float)( (10.0 + Id + Is)/255.0 );
117             // translatiraj tocku obzirom na pravi centar
118             add(&p, cntr);
119             // nacrtaj ako smije
120             if(can_update_z_buffer(z_buf, &p)) {
121                 glColor3f((float)I, (float)(I/2), (float)I);
122                 glVertex2i(zaokruzi(p.x), zaokruzi(p.y));
123             }
124         }
125     }
126     glEnd();
127 }
128
129 void renderScene() {
130     const int R = 100;
131     Point3D l = {1, 0, 1}; // vektor prema izvoru
132     Point3D v = {0, 0, 1}; // vektor prema gledatelju
133     Point3D c0 = {150, 150, 0}; // centar prve kugle
134     Point3D c1 = {250, 150, -50}; // centar druge kugle
135
136     // normiranje vektora
137     normalize(&l);
138     normalize(&v);
139
```

```
140       // zauzmi spremnik
141       z_buffer *z_buf = new_z_buffer(400, 300);
142       if( z_buf==NULL ) {
143           printf("Nema dovoljno memorije.");
144           return;
145       }
146
147       // ocisti z-spremnik
148       clear_z_buffer(z_buf);
149
150       // nacrtaj obje kugle
151       glPointSize(1);
152       bojajKuglu(R, &c0, &l, &v, z_buf);
153       bojajKuglu(R, &c1, &l, &v, z_buf);
154
155       // oslobodi spremnik
156       free_z_buffer(z_buf);
157   }
```

Program je potrebno pratiti od metode renderScene. U toj metodi najprije definiramo vektore $\vec{L}$ (prema izvoru svjetla) i $\vec{V}$ (prema gledatelju), te centre obje kugle. Potom stvaramo strukturu z-spremnika, čistimo ga, pozivamo crtanje obje kugle i konačno oslobađamo z-spremnik. Svaki od ovih poslova obavlja po jedna pomoćna metoda koja je također prikazana u izvornom kodu. Pri implementaciji z-spremnika u ovoj funkciji točka je bliža promatraču što joj je z-koordinata veća (jer je promatrač na z-osi u pozitivnoj beskonačnosti). Zbog toga je inicijalno z-spremnik popunjen s negativnim vrijednostima.

Isti program u nastavku je prikazan uporabom OpenGL-ove implementacije z-spremnika. Rezultat toga je značajno pojednostavljenje koda. Međutim, uključivanje OpenGL-ove implementacije unijelo je promjene kroz čitav program, pa je zbog toga u ispisu 8.2 prikazan čitav kod. Promjene započinju od metode main gdje se može vidjeti promjena u inicijalizaciji, uključivanje podrške za z-spremnik te definiranje načina provođenja usporedbe z-koordinata. Kako bi se dobila ista slika kao i ona generirana ispisom 8.1, smjer gledanja je trebalo okrenuti iz $+z$ u $-z$. Stoga je u metodi display dodana matrica m koja okreće predznak z-koordinati, i nakon što je učitana jedinična matrica, ona je pomnožena tom matricom. U istoj metodi ažuriran je i poziv funkcije brisanja, koja sada briše i stanje z-spremnika. Promijenjena je i metoda reshape, točnije poziv metode glOrtho, koji sada sa zadnja dva parametra specificira z-koordinate bližeg i daljeg kraja kojim se definira volumen pogleda (vrijednosti su postavljene na 300 – bliži kraj, te -300 – dalji kraj).

Ispis 8.2: Crtanje dviju kugli uz z-spremnik OpenGL-a

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
```

```
4  #include <GL/glut.h>
5
6  void reshape(int width, int height);
7  void display();
8  void renderScene();
9
10 int main(int argc, char **argv) {
11     glutInit(&argc, argv);
12     glutInitDisplayMode(GLUT_DOUBLE|GLUT_DEPTH);
13     glutInitWindowSize(400, 300);
14     glutInitWindowPosition(0, 0);
15     glutCreateWindow("Primjer 9");
16     glutDisplayFunc(display);
17     glutReshapeFunc(reshape);
18     glEnable(GL_DEPTH_TEST);
19     glDepthFunc(GL_LEQUAL);
20     glutMainLoop();
21 }
22
23 void display() {
24     GLdouble m[] = {1,0,0,0,0,1,0,0,0,0,-1,0,0,0,0,1};
25     glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
26     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
27     glLoadIdentity();
28     glMultMatrixd(m);
29     renderScene();
30     glutSwapBuffers();
31 }
32
33 void reshape(int width, int height) {
34     glMatrixMode(GL_PROJECTION);
35     glLoadIdentity();
36     glOrtho(0, width-1, 0, height-1, 300, -300);
37     glViewport(0, 0, (GLsizei)width, (GLsizei)height);
38     glMatrixMode(GL_MODELVIEW);
39 }
40
41 int zaokruzi(double d) {
42     if(d>=0) return (int)(d+0.5);
43     return (int)(d-0.5);
44 }
45
46 // Struktura tocke / 3D vektora
47 typedef struct {
48     double x;
49     double y;
50     double z;
51 } Point3D;
52
53 // Normiranje vektora
54 void normalize(Point3D *pnt) {
```

```
55      double norm = sqrt(pnt->x*pnt->x + pnt->y*pnt->y
56          + pnt->z*pnt->z);
57      pnt->x = pnt->x / norm;
58      pnt->y = pnt->y / norm;
59      pnt->z = pnt->z / norm;
60  }
61
62  // Skalarni produkt dvaju vektora
63  double scalarProduct(Point3D *a, Point3D *b) {
64      return a->x*b->x + a->y*b->y + a->z*b->z;
65  }
66
67  // Pomice predanu tocku za trazeni vektor
68  double add(Point3D *p, Point3D *delta) {
69      p->x += delta->x;
70      p->y += delta->y;
71      p->z += delta->z;
72  }
73
74  // Metoda za crtanje kugle, uz pretpostavku da se promatra iz
75  // smjera pozitivne z-osi.
76  void bojajKuglu(const int R, Point3D *cntr, Point3D *l,
77          Point3D *v) {
78      int x, y; // indeksi petlje
79      double Id, Is; // difuzna i reflektirajuca komponenta
80      float I; // ukupni intenzitet
81      double ln_scalar; // pomocna varijabla
82      Point3D p; // tocka na kugli
83      Point3D n; // normala u tocki p
84      Point3D r; // vektor reflektirane komponente
85
86      glBegin(GL_POINTS);
87      for( x=-R; x<=R; x++ ) {
88          for( y=-R; y<=R; y++ ) {
89              // odredi tocku ishodisne kugle
90              p.x = x; p.y = y;
91              p.z = R*R - (p.x*p.x + p.y*p.y);
92              if( p.z < 0 ) continue;
93              p.z = sqrt(p.z);
94              // odredi normalu
95              n = p;
96              normalize(&n);
97              // racunaj difuznu komponentu
98              ln_scalar = scalarProduct(l, &n);
99              Id = ln_scalar;
100             if( Id > 0.0 ) Id = 200*Id; else Id = 0.0;
101             // racunaj reflektirajucu zraku i komponentu
102             r.x=2*ln_scalar*n.x-l->x;
103             r.y=2*ln_scalar*n.y-l->y;
104             r.z=2*ln_scalar*n.z-l->z;
105             normalize(&r);
```

```
106                 Is = scalarProduct(&r, v);
107                 if( Is > 0.0 ) {
108                     Is = 45*pow(Is,1.1);
109                 } else Is=0.0;
110                 // ukupni intenzitet
111                 I = (float)( (10.0 + Id + Is)/255.0 );
112                 // translatiraj tocku obzirom na pravi centar
113                 add(&p, cntr);
114                 // nacrtaj; z-spremnik ce provjeriti je li to OK
115                 glColor3f((float)I, (float)(I/2), (float)I);
116                 glVertex3i(zaokruzi(p.x), zaokruzi(p.y),
117                     zaokruzi(p.z));
118             }
119         }
120         glEnd();
121     }
122
123     void renderScene() {
124         const int R = 100;
125         Point3D l = {1, 0, 1}; // vektor prema izvoru
126         Point3D v = {0, 0, 1}; // vektor prema gledatelju
127         Point3D c0 = {150, 150, 0}; // centar prve kugle
128         Point3D c1 = {250, 150, -50}; // centar druge kugle
129
130         // normiranje vektora
131         normalize(&l);
132         normalize(&v);
133
134         // nacrtaj obje kugle
135         glPointSize(1);
136         bojajKuglu(R, &c0, &l, &v);
137         bojajKuglu(R, &c1, &l, &v);
138     }
```

8.6 Postupak Warnocka

Postupak Warnocka teži tome da se područje ispitivanja udaljenosti objekta do promatrača poveća. Kod postupka ispitivanja z-spremnika provjerava se jedan slikovni element, kod Watkinsovog postupak provjerava se ispitna linija, dok u postupku Warnocka područje želimo povećati na kvadrat za koji ćemo moći odlučiti što je vidljivo, odnosno općenito što se nalazi u njemu.

Postupak Warnocka započinje promatranjem čitavog zaslona i svih poligona scene. Početno područje zaslona nazivamo prozor. Za trenutni prozor ispitujemo vidljivosti i ako je potrebno, dijelimo ga u četiri podprozora i pripadne liste poligona. Podjelu rekuzivno dalje ponavljamo dok ne ostvarimo neki od postavljenih slučajeva. Na slici 8.14 prikazan je osnovni prozor i podijela na četiri jednaka dijela. Ako je pojedini dio dalje potrebno dijeliti tada to i učinimo sve do unapri-

jed zadane dubine rekurzije. U prikazanom primjeru gornji lijevi prozor se dalje dijeli, dok ostale nismo dijelili, a nakon te podjele gornji desni smo podijelili i time završili podjelu. Za dio prozora gdje smo rekurzijom došli do kraja koristit ćemo neki drugi algoritam odluke u prikazu poligona.

Mogući slučajevi koji određuju hoćemo li dalje ostvarivati podjelu su:

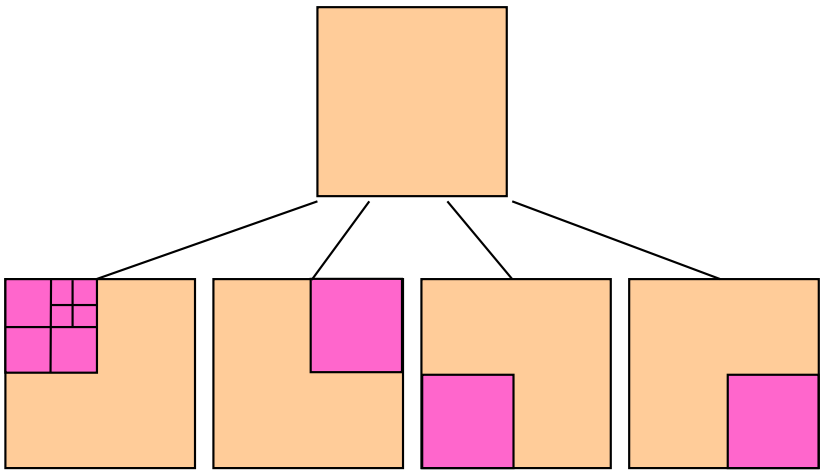
(1) poligon je izvan prozora pa ga uklonimo iz liste,

(2) samo jedan poligon siječe prozor ili je u prozoru,

(3) poligon (jedan) prekriva prozor,

(4) više poligona prekriva ili siječe prozor ali jedan od njih je kroz čitav prozor konzistentno ispred svih,

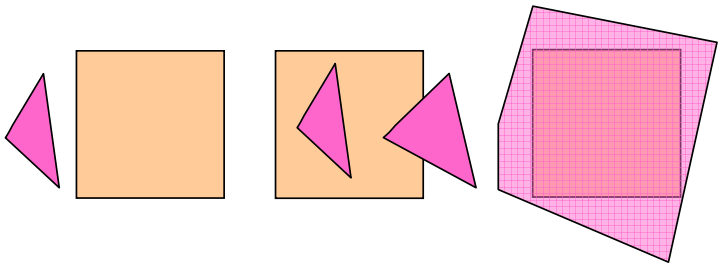
(0) scena nije jednostavna - dijelimo dalje.



Slika 8.14: Warnockov postupak: rekurzivna podjela prostora na četiri dijela

Znači, ako je poligon izvan prozora kojeg smo dobili podjelom, jasno je da ćemo ga ukloniti iz liste (1) 8.15. Ako imamo samo jedan poligon koji sječe prozor ili je u prozoru (2), tada je scena jednostavna i možemo ju prikazati za taj prozor, pa ćemo dani poligon i prikazati u prozoru. Treći slučaj je također jednostavan, a to je kada jedan poligon prekriva prozor, pa i njegova boja u konačnici dređuje prikaz. Četvrti slučaj je složeniji i potrebna je detaljnija analiza. U četvrtom slučaju, kada više poligona prekriva ili siječe prozor, potrebno je provjeriti postoji li jedan koji je najbliži promatraču i koji prekriva ostale. U tom slučaju prikazat ćemo taj najbliži, a ako to ne možemo jednoznačno utvrditi prelazimo na zadnji slučaj (0), kada scena nije jednostavna i moramo ju dalje dijeliti.

Primjer jednostavne scene u kojoj su prikazana dva poligona dan je na slici 8.16. Prvi poligon je u obliku kućice, a drugi je kvadratan. Nakon prve podjele,



Slika 8.15: Warnockov postupak mogući slučajevi: poligon je izvan prozora (1), poligon je u prozoru ili siječe prozor (2), poligon prekriva prozor (3)

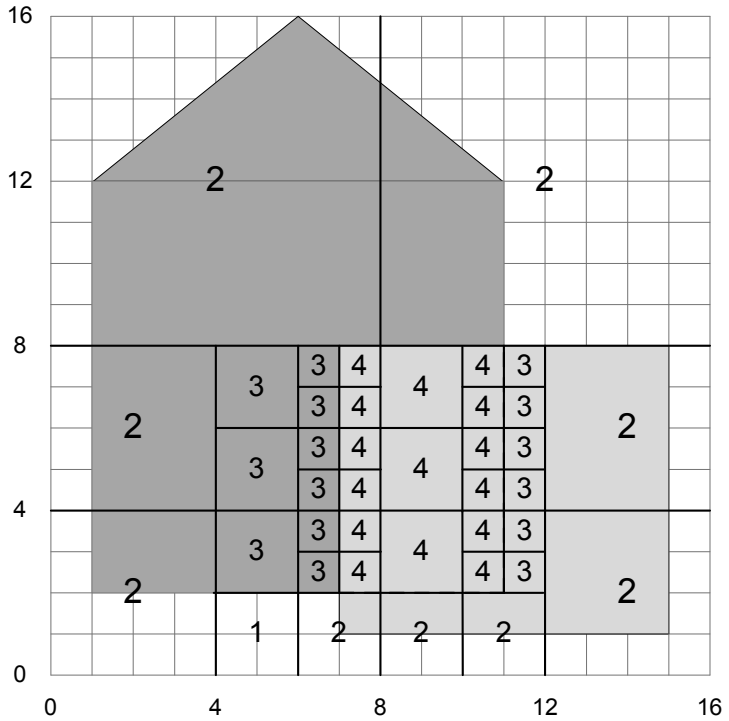
gornji lijevi prozor ($x \in [0, 8]$, $y \in [8, 16]$) i gornji desni prozor ($x \in [8, 16]$, $y \in [8, 16]$) spadaju pod slučaj (2) - kod oba prozora postoji samo jedan poligon koji siječe prozor. Ti se prozori tada mogu nacrtati crtanjem tog poligona (kućice).

Donji lijevi prozor ($x \in [0, 8]$, $y \in [0, 8]$) ćemo morati dijeliti dalje. Nakon te podjele, za gornji lijevi podprozor ($x \in [0, 4]$, $y \in [4, 8]$) možemo zaključiti da poligon siječe prozor pa podprozoru pridjeljujemo oznaku (2). Isto vrijedi i za donji lijevi podprozor ($x \in [0, 4]$, $y \in [0, 4]$). Za gornji desni ($x \in [4, 8]$, $y \in [4, 8]$) i donji lijevi ($x \in [4, 8]$, $y \in [0, 4]$) situacija je složena pa imamo slučaj (0) koji zahtijeva daljnje rekurzivno dijeljenje. Postupak ponavljamo i za donji desni prozor ($x \in [8, 16]$, $y \in [0, 8]$).

Idjea koja je ovdje prikazana i korištena za određivanje koji poligon je najbliži promatraču, može se na sličan način upotrijebiti i za stvaranje strukture četverostabla te oktalnog stabla.

8.7 Četvero i oktalno stablo

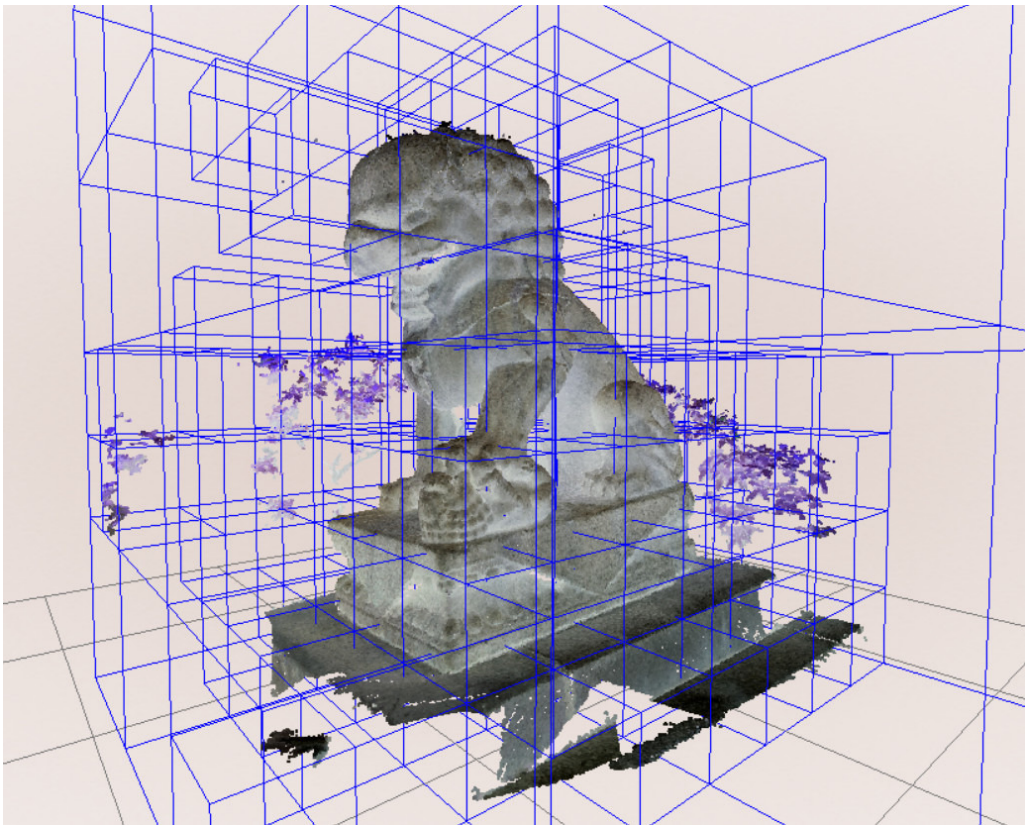
Strukturu stabla čini niz povezanih čvorova od kojih je korijen osnovni čvor, a grane ga povezuju s ostalim čvorovima koji se dalje mogu granati ili mogu biti završni čvorovi. U strukturi četverostabla (engl. *Quadtree*) primitive scene na početku dijelimo u četiri grane stvarajući stukturu stabla ovisno u kojem kvadrantu se primitivi nalaze. Podjelu rekurzivno ponavljamo do unaprijed zadane dubine rekurzije. Uvjet zaustavljanja podijele je i ako je broj primitiva u nekom čvoru manji od unaprijed zadanog broja. Primitivi u općem slučaju mogu biti točke, linije, poligoni, cijeli objekti ili dijelovi scene. Ako se primitiv nađe na rubu između dva susjedna dijela prostora, tada se zapisuje u oba podčvora ili se takvi primitivi čuvaju u čvoru kojeg dijelimo, te se dalje ne razvrstavaju, ovisno o tome za koju varijantu izgradnje stabla se odlučimo. Izgrađeno stablo služi za brzu pretragu prostora. Zanima li nas, na primjer, za zadanu točku je li u nekom pologonu, iz koordinata točke brzo ćemo odrediti u kojem podprozoru se nalazi i s kojim poligonima upoće trebamo raditi provjeru. Oktalno stablo (engl. *Octree*)



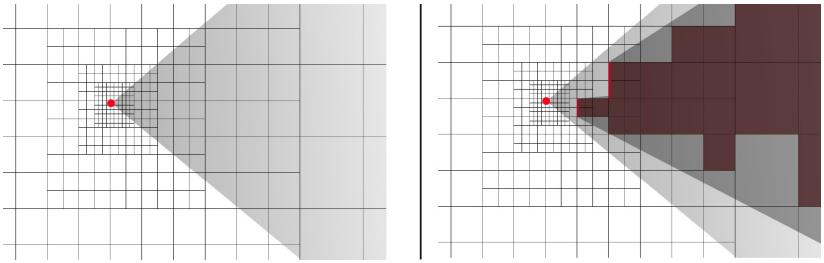
Slika 8.16: Warnockov postupak - primjer podjele za dva poligona gdje su označeni mogući slučajevi.

analogno je četverostablu, osim što svaki čvor sarži podjelu na osam grana koje dalje rekurzivno pratimo. Osim x i y koordinata promatramo i cijeli volumen scene odnosno z koordinatu. Slika 8.17 prikazuje primjer podjele prostora pri izgradnji oktalnog stabla. U prikazanom primjeru plavo su označeni pojedini oktanti. Oktanti koji sadrže više točaka rekurzivno se dijele na manje oktante.

Pogledajmo jedan primjer gdje je pogled na scenu odozgo. Prostor je podijeljen u četverostablo i pojedini čvorovi su nazvani ćelijama (Slika 8.18). Na lijevom dijelu slike prikazana je scena podjeljena u kvadrante ovisno o udaljenosti od promatrača, koji je označen crvenom točkom. Vidno polje promatrača označeno je svijetlo-sivom bojom. Na desnoj slici sadržana je ista scena i položaj promatrača ali su u scenu dodani i objekti. Crvenom linijom su označeni bridovi koji s obzirom na promatrača zaklanjaju druge objekte u sceni. Zagasito-crvenom bojom označene su ćelije koje neće biti prikazivane jer su zaklonjene (osim rubnih koje sadrže crveni brid). Time se značajno smanjuje broj ćelija koje je potrebno prikazivati čime se postupak prikaza scene ubrzava. Ovdje možemo lako uočiti da kod složenih scena, uz dobru podijelu možemo vrlo veliki dio scene ukloniti tako da ga uopće ne šaljemo grafičkom protočnom sustavu i time bitno ubrzamo izvođenje aplikacije. U prvom redu to su sve ćelije koje su van krnje piramide



Slika 8.17: Oktalno stablo. Prostor dijelimo u oktante ovisno o sadržaju pojedinih oktanata.



Slika 8.18: Prostor podijeljen na ćelije koje mogu biti prazne granične ili neprozirne. Iz crvene točke se promatra scene, crvenom linijom na desnoj slici su neprozirne ćelije koje zaklanjaju plave ćelije.

pogleda (engl. *frustum*), a dalje to je i većina zagasito-crvenih ćelija s obzirom da su zaklonjene. Znači u konačnici, samo će ćelije koje su u pramidi pogleda i nisu zaklonjene, biti proslijeđene protočnom sustavu.

Cijela scena se na početku izvođenja aplikacije iz glavne memorije računala prebacuje u memoriju grafičke kartice. Danas uobičajeno ima dovoljno memorije na grafičkoj kartici za ovakav način rada. Ako mijenjamo scenu onda obično pričekamo prebacivanje objekata i tekstura neke druge scene. U OpenGLu, kada želimo objekte jednokratno prebaciti u memoriju grafičke kartice i onda ih višekratno koristiti upotrijebit ćemo VBO (Vertex Buffer Object), odnosno više takvih objekata. Određivanje geometrijskih podataka koji su potrebni pri iscrtavanju (obično se obavlja na CPU) i slanje svaki puta nanovo u memoriju grafičke kartice ne bi bilo dobro riješenje. Izračunavanje koji dijelovi su zaklonjeni na osnovi sagrađenog stabla obično se radi na CPU, pa se samo određeni dijelovi koji su vidljivi, proslijeđuju na prikazivanje ali iz memorije grafičke kartice. U protočnom sustavu, dalje, niz poligona možemo ukloniti kao stražnje za što smo vidjeli da je potrebno svega nekoliko naredbi u OpenGLu, a obično pola poligona jesu stražnji. Programi za sjenčanje geometrije osim što omogućuju stvaranje nove geometrije omogućuju i uklanjanje postojeće geometrije iz protočnog sustava, pa ovdje onda možemo primijeniti razne algoriteme koji će zaključiti što se još može ukloniti prije faze rasterizacije, čime smanjujemo opterećenje i iscrtavanje zaklonjenih poligona (engl. *overdraw*).

Stvorene stukture podataka o sceni u obliku četvero ili oktalnog stabla koriste se u raznim primjenama gdje je potrebna pretraga ili analiza prostora scene. To je, na primjer, u postupcima detekcije kolizije, gdje je potrebno utvrditi koji objekt s kojim je potencijalno u koliziji. Za na primjer n objekata bit će potrebno provjeriti svaki objekt sa svakim je li u koliziji, odnosno bi će potreno $n(n-1)/2$ provjere i to će biti potrebno izračunavati za svaki okvir animacije koji se prikazuje, odnosno nekoliko desetaka puta u sekundi. Podjela svih objekata u skupine po oktantima koje su potencijalno u koliziji drastično će smanjiti

potreban broj ispitivanja. U postupku praćenja zrake, gdje je izraziti problem određivanja probodišta zrake i tijela bitno je što je moguće više smanjiti broj nepotrebnih ispitivanja, a organizacija prostora i podataka u tome znatno doprinosi.

8.8 Algoritam Cohen Sutherlanda

Osnovna primjena ovog algoritma je odsijecanje linija, odnosno poligona s obzirom na područje zaslona na kojem se ostvaruje prikaz. Ovaj algoritam omogućava nam da unutar zaslona definiramo dio podprostora unutar kojega će se obavljati iscrtavanje, dok će izvan tog podprostora iscrtavanje biti onemogućeno. Algoritam Cohen Sutherlanda pri tome pretpostavlja da će potprostor unutar kojega je dozvoljeno crtanje biti pravokutnog oblika. Primjerice, neka je zaslon dimenzija 1024×768; legalni raspon za x-koordinate tada je od 0 do 1023 a za y-koordinate od 0 do 767. Pretpostavimo sada da smo kao podprostor unutar kojeg je dozvoljeno crtanje definirali pravokutnik čiji si su dijagonalni vrhovi točke $(100, 100)$ i $(800, 500)$. To je pravokutno područje određeno s $x_{min} = 100$, $x_{max} = 800$, $y_{min} = 100$ te $y_{max} = 500$. Pitanje na koje treba odgovoriti jest kako osigurati da uz ovako definirani podprostor algoritmi za crtanje linije (poput Bresenhamovog algoritma) ne "pale" slikovne elemente koji se nalaze izvan definiranog podprostora. Jedno moguće rješenje je modificirati algoritam za crtanje linije tako da svaki puta kada treba obojati neki slikovni element najprije provjeri je li taj slikovni element u zadanim granicama, te ako nije, da preskoči naredbu kojom se postavlja boja slikovnog elementa. Međutim, to je vrlo neefikasno. Primjerice, uz prethodno definirati podprostor, poziv korisnika da kojim traži crtanje linije zadane s početnom točkom $(1, 1)$ i konačnom točkom $(1000, 1)$ pokušat će obojati 1000 slikovnih elemenata, za svaki će utvrditi da pripada izvan podprostora unutar kojeg je dozvoljeno crtanje, i na kraju neće obojati niti jedan slikovni element. Međutim, postupak će potrošiti jednako vremena kao i da je obojao sve slikovne elemente, i još gore, potrošit će dodatnu količinu vremena jer će za svaki slikovni element raditi provjeru treba li ga prikazati ili ne.

Algoritam Cohen Sutherlanda pokušava pristupiti ovom zadatku inteligentije. Svaku liniju koju treba nacrtati, korisnik zadaje s dvije točke – s početnom i konačnom točkom. Ideja ovog algoritma jest brzo provjeriti može li se čitav postupak crtanja linije naprosto preskočiti (jer je čitava linija izvan podprostora u kojem je crtanje dozvoljeno), te ako ne može, onda se po potrebi računaju nova početna i konačna točka koje određuju dio zadane linije i koje se u cijelosti nalaze unutar podprostora u kojem je dozvoljeno crtanje – tada se poziva klasični algoritam za brzo crtanje linija kojem se predaju tako nanovo izračunate točke i koji dalje više ništa ne treba provjeravati jer su predane točke sigurno unutar područja u kojem je iscrtavanje dozvoljeno.

Prvi korak algoritma je podjela prostora na devet dijelova i pridjeljivanje četverobitnog koda početnoj i konačnoj točki koja određuje liniju koju treba nacrtati. Za konkretnu točku, u pridruženoj kodnoj riječi ćemo prvi bit s lijeva postaviti u jedinicu ako je točka iznad y_{max}, a inače u nulu. Drugi bit ćemo postaviti u jedinicu ako je točka u dijelu prostora ispod y_{min}, inače u nulu. Sljedeći bit ćemo postaviti u jedinicu ako je točka desno od x_{max} i konačno zadnji bit ćemo postaviti u jedinicu ako je točka lijevo od x_{min} (Slika 8.19). Na ovaj način imat ćemo za svaku točku jednoznačno pridijeljen kod.

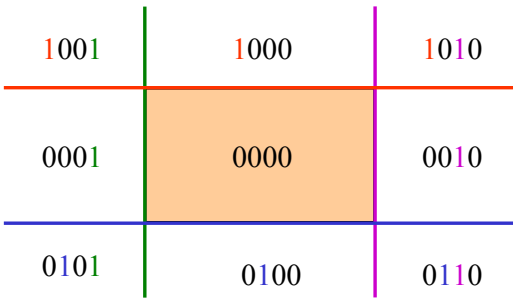
Tablica 8.1: Pridjeljivanje koda *abcd* točki $T(x, y)$ kod algoritma Cohen Sutherlanda

$$T(x,y) \to abcd \quad a = \begin{cases} 1, & y > y_{max} \\ 0, & \text{inače.} \end{cases}$$

$$b = \begin{cases} 1, & y < y_{min} \\ 0, & \text{inače.} \end{cases}$$

$$c = \begin{cases} 1, & x > x_{max} \\ 0, & \text{inače.} \end{cases}$$

$$d = \begin{cases} 1, & x < x_{min} \\ 0, & \text{inače.} \end{cases}$$

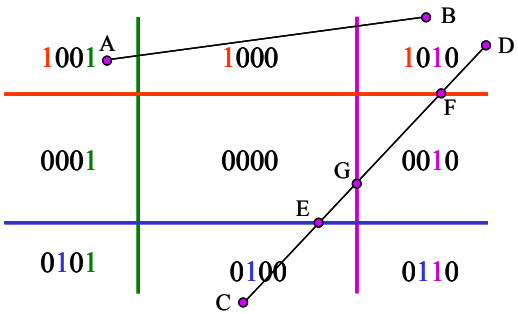


Slika 8.19: Podjela prostora na devet dijelova kod algoritma Cohen Sutherlanda.

Središnji dio koji čini dio zaslona unutar kojeg se obavlja crtanje ima pridijeljen kod 0000; naime ako točka pada u taj dio, njezina x-koordinata je veća ili jednaka x_{min} i manja ili jednaka x_{max}, a njezina y-koordinata je veća ili jednaka y_{min} i manja ili jednaka y_{max}; stoga će sva četiri bita biti postavljena na 0. Za promatranu točku $T(x, y)$ dovoljno je ispitati predznak $y_{max} - y$ da bi postavili najznačajniji bit koda, što je sklopovski jednostavna operacija. U ovom algo-

ritmu posebice treba razmišljati o sklopovskoj implementaciji, s obzirom da je to temeljni algoritam, pa ga svaki sustav treba podržavati i sklopovski implementirati.

Nakon što točkama promatrane dužine V_1, V_2 pridijelimo četverobitne kodove c_1 i c_2 krećemo s ispitivanjem. Prvi trivijalan slučaj je ako su obje točke unutar središnjeg prozora $c_1 = 0000$, $c_2 = 0000$ – odsijecanje nije potrebno i ta se linija može nacrtati. Ako to nije slučaj, provjerit ćemo rezultat logičke operacije c_1 AND c_2 između postavljenih bitova koda. Ako je taj rezultat različit od 0000 dužina trivijalno nije vidljiva (čitava je iznad, ispod, lijevo ili desno od zadanog podprostora) – crtanje linije se preskače. Možemo primijetiti da, ako su na primjer obje točke iznad y_{max}, očito je i cijela dužina iznad y_{max} pa će četverobitni kod biti 1000 AND 1000 = 1000, kao što je slučaj za dužinu AB na slici 8.20. Ako ni to nije ispunjeno, potreno je odsijecanje.



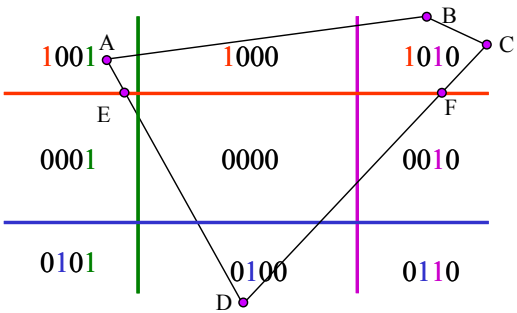
Slika 8.20: Primjer odsijecanja dužine kod algoritma Cohen Sutherlanda.

Odsijecanje se provodi prema unaprijed zadanom redoslijedu ispitivanja. Taj redoslijed može biti proizvoljni zadan. U primjeru koji slijedi, prvo ćemo provjeravati sve bitove postavljene u 1 za točku V_1, s lijeva na desno; potom prelazimo na sve bitove postavljene u 1 za točku V_2, također s lijeva na desno. Promotrimo primjer na slici 8.20 za dužinu CD. Točka C ima pridijeljen kod $c_1 = 0100$, a točka D kod $c_2 = 1010$. Kako je pri provjeri prvi bit koji je postavljen i na koji ćemo naići drugi bit s lijeva u c_1 tako je prva provjera s pravcem $y = y_{min}$. Potrebno je odrediti sjecište pravca određenog točkama CD i pravca $y = y_{min}$. Označimo to sjecište s E. Točki E ćemo ponovno pridijeliti kod, a kako smo upravo načinili ispitivanje s y_{min}, i točka se nalazi upravo na liniji y_{min} jedinica koju smo provjeravali se briše. Segment CE ćemo odbaciti, dok novu liniju koju dalje provjeravamo čini linija ED. Svi bitovi točke E su nula pa prelazimo na točku D. Postavljen je prvi bit s lijeva pa se obavlja provjera, odnosno traži se sjecište linije ED s pravcem y_{max}, odnosno točka F i konačno zadnji bit koji određuje provjeru s x_{max} daje točku G. Konačni segment koji treba nacrtati na kraju je određen točkama E i G – zovemo proceduru za crtanje segmenta EG.

Tijekom rada algoritma pojedine dijelove linije odbacujemo i dobivamo nove sve kraće segmente koji na kraju dovode do segmenta koji će se u stvarnosti poslati na crtanje. Međutim, izračun sjecišta sa pravcima y_{max}, y_{min}, x_{max} i x_{min} uvijek obavljamo s linijom koja je određena izvornim točkama – u našem primjeru CD. Naime, ako bismo svaki puta sjecišta računali s novoizvedenim točkama, moguća je pojava numeričkih pogrešaka koje bi na kraju mogle dovesti do pogrešno utvrđenih točaka između kojih treba obaviti crtanje. S druge pak strane, parametri pravca određenog točkama C i D se tijekom postupka ne mijenjaju pa ih je moguće izračunati unaprijed i dalje koristiti za sve izračune čime se dobiva i na preciznosti i na brzini.

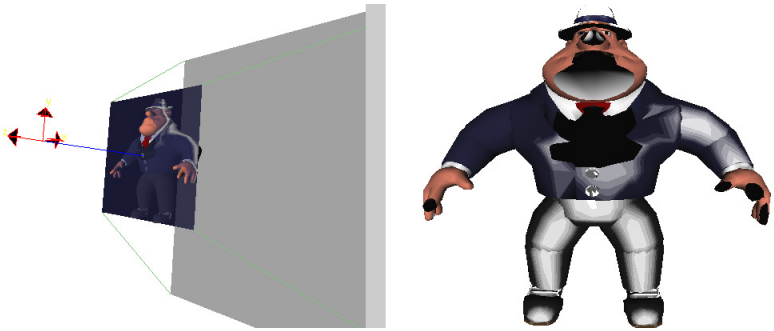
Možemo primijetiti da je moguć nepotreban izračun nekih točaka. U našem primjeru točka F ne čini konačno riješenje i izračun nije potreban, no to je nedostatak algoritma. Za drugačije odabran redoslijed točaka, na primjer DC i redoslijed dobivenih sjecišta bit će različit. Tako bi za segment DC dobili redom segmente: FC, GC, GE.

Odsijecanje poligona obzirom na promatrani prozor provodi se slično. Uspoređivanje pojedinih bridova poligona provodi se redom obzirom na pojedine pravce i čuva se lista bridova koji nakon takvog odsijecanja ostaju. Na slici 8.21 prikazano je odsijecanje poligona $ABCD$ obzirom na gornji pravac y_{max}, nakon čega ostaje poligon FDE koji se dalje sječe s ostalim linijama odsijecanja.



Slika 8.21: Primjer odsijecanja poligona kod algoritma Cohen Sutherlanda.

Opisani algoritam lako je proširiv na tri dimenzije, odnosno na volumen pogleda, koji može biti kvadar ili krnja piramida. U tom slučaju četverobitini kod potrebno je proširiti još s dva bita koja određuju je li nešto ispred prednje ravnine odsijecanja ili je iza stražnje ravnine odsijecanja. Postupak se provodi sklopovski i rezultat odsijecanja vidljiv je na slici 8.22. Lijevo je prikazan objekt, položaj kamere te prednja i stražnja ravnina odsijecanja. Prednja ravnina odsijecanja zahvaća dio objekta, odnosno siječe ga te će nakon provedenog postupka ostati rupa u objektu kroz koju se vidi stražnja strana objekta što je vidljivo na slici 8.22 desno.



Slika 8.22: Primjer odsijecanja obzirom na volumen pogleda kod algoritma Cohen Sutherlanda.

8.9 Algoritam Cyrus Beck

Algoritam Cyrus Becka omogućuje određivanje sjecišta linije, odnosno dužine s proizvoljnim konveksnim poliedrom. Algoritam je vrlo sličan algoritmu koji se koristi za popunjavanje konveksnog poligona koji smo prethodno opisali u potpoglavlju 4.2.4, osim što je proširen tako da radi u 3D prostoru. Pri tome, pojmovi koje smo koristili kod algoritma popunjavanja konveksnog poligona "lijevo sjecište" i "desno sjecište" ovdje su poopćeni u "potencijalno ulaznu" točku i "potencijalno izlaznu" točku.

Prvo je potrebno odrediti izraz za određivanje sjecišta pravca i poligona. Neka je pravac određen točkama dužine P_0P_1 (vidi sliku 8.23). Parametarska jednažba pravca kroz te dvije točke za parametar t je: $P(t) = t(P_1 - P_0) + P_0$. Neka je normala poligona $\vec{n}_i$ usmjerena prema vanjštini objekta. Odaberimo sada jednu točku poligona i označimo ju P_{Ei}. Obično se za ovu točku odabire proizvoljan vrh poligona. Sada promatramo vektor između odabranog vrha P_{Ei} i bilo koje točke na pravcu. Za kut između promatranog vektora i vektora normale poligona možemo zaključiti da je taj kut manji od 90^0 ako je točka na pravcu van promatranog poligona, odnosno skalarni produkt tih vektora je pozitivan. Na slici 8.23 prikazan je pogled okomito na ravninu u kojoj je poligon, pa nam je poligon zapravo degenerirao u jednu dužinu na koju je označena normala $\vec{n}_i$. Očito je da je kut između vektora normale tog poligona $\vec{n}_i$ i vektora izmedju točaka P_{Ei} i na primjer P_0 biti manji od 90^0. Promatramo li točku koja je u ravnini poligona, produkt promatranih vektora bit će nula, dok za točke koje su ispod ravnine poligona, kao što je na slici P_1, produkt će biti negativan. Nas zanima točka probodišta, odnosno slučaj kada je promatrani produkt nula, odnosno:

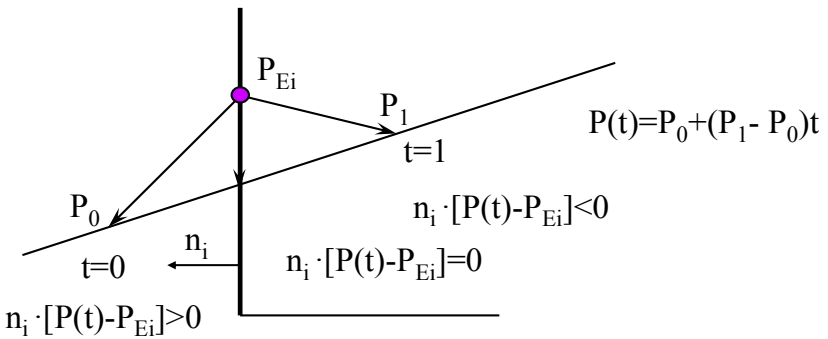
$$\vec{n} \cdot (P(t) - P_{Ei}) = 0$$

Uvrstimo li $P(t) = t(P_1 - P_0) + P_0$ (što je parametarska jednadžba pravca), za parametar t ćemo dobiti:

$$t = \frac{\vec{n}_i \cdot (P_0 - P_{Ei})}{-\vec{n}_i \cdot D,}$$

gdje je $D = P_1 - P_0$. I to je parametar t koji određuje probodište pravca i ravnine u kojoj leži promatrani poligon. Promotrimo detaljnije izraz za parametar koji smo upravo dobili. U brojniku i u nazivniku imamo skalarni produkt dva vektora, od kojih je jedan vektor normale $\vec{n}_i$. Drugim riječima imamo omjer predznačenih duljina projekcija vektora $P_0 - P_{Ei}$ i vektora $P_1 - P_0$ na vektor normale. Naveden omjer, odnosno parametar t može biti pozitivan nula i negativan, te nam određuje točku probodišta.

U izvedenoj formuli za t moramo još pripaziti da nema nepoželjnih slučajeva, odnosno dijeljenja s nulom. Nastupi li takav slučaj, obradu poligona treba preskočiti. Nazivnik može biti nula u nekoliko slučajeva. Najprije, bit će nula ako je vektor normale nul-vektor, što moramo provjeriti. To se može dogoditi jer zbog različitih transformacija, poligon čiju normalu računamo može degenerirati u dužinu čime računanje vektora normale postaje besmisleno, a računamo li normalu na uobičajen način kao vektorski produkt, dobit ćemo nul-vektor. Isto tako, ako poligoni postanu sićušni, zbog numeričke nepreciznosti moguće su degeneracije vektora normale u nul-vektor. Konačno, u zapisu objekata možemo imati poligone koji su degenerirali u dužinu, što obično u prikazu nećemo ni primijetiti, a normala će biti nula.



Slika 8.23: Određivanje probodišta pravca na kojem je dužina P_0P_1 i poligona čija je normala $\vec{n}_i$.

Sljedeći vektor koji nam može stvoriti probleme je vektor D odnosno $P_1 - P_0$. On će biti nula ako te dvije točke degeneriraju u jednu. Lijepo bi bilo da to ne učine, ali znamo da stvari u životu nisu idealne pa zbog prethodno opisanih razloga upravo to se može dogoditi. Treća je mogućnost da oba promatrana vektora nisu nul-vektori, ali da njihov skalarni produkt je. To će biti u slučaju

kada je promatrani pravac paralelan s ravninom u kojoj leži poligon. U tom slučaju, sječiste ili ne postoji (ako pravac ne leži u ravnini poligona) ili ih ima beskonačno mnogo (ako leži u ravnini poligona). Dijeljenje s nulom u tom slučaju moramo spriječiti tako da sjecište ne određujemo.

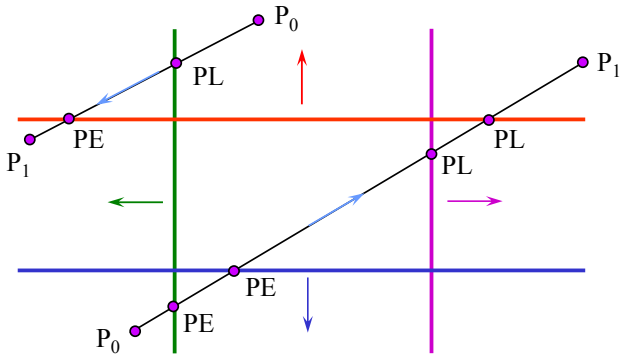
Još je zanimljiv slučaj ako je brojnik nula, odnosno ako je vektor $P_0 - P_{Ei}$ nula. To znači da su se odabrana točka poligona i prva točka promatrane dužine poklopili. Parametar t je tada nula, što je sasvim u redu jer je prva točka ujedino i sjecište, pa je u tom slučaju sve regularno.

Određivanje sjecišta pravca zadanog točkama i poligona može se svesti i na 2D slučaj gdje promatramo sjecište 2D pravca i poligona. Sve ostaje potpuno isto kao i u opisanom slučaju u 3D, osim što normala koju smo promatrali $\vec{n}_i$ je sada normala brida poligona. Sliku 8.23 sada možemo promatrati u 2D i po istom postupku odrediti sjecište s pravcem.

Sljedeći korak je određivanje potencijalno ulaznih i potencijalno izlaznih točaka odnosno vrijednosti parametra t za koje postoji sjecište te vrsta sjecišta (ulazno ili izlazno); u tu svrhu promotrimo sliku 8.23. Označimo potencijalno ulazno sjecište s PE (engl. *entering*), a potencijalno izlazno s PL (engl. *leaving*). Postavimo inicijalne vrijednosti $PE = 0$ i $PL = 1$. Je li neko sjecište potencijalno ulazno, određeno je kutem između vektora normale brida $\vec{n}$ (promatramo li 2D slučaj) i vektora pravca $D = P_1 - P_0$. Promatrat ćemo produkt $\vec{n} \cdot D$. Primijetimo da smo ovaj produkt ionako već računali pri određivanju parametra t pa se izračun pokazuje višestruko korisnim. Ako je ovaj produkt negativan, odnosno kut između vektora normale i pravca je veći od 90ř, sjecište je potencijalno ulazno ($PE = t$), dok je u suprotnom potencijalno izlazno ($PL = t$). Na ovaj način sva sjecišta imamo razvrstana kao PE ili kao PL. Izbor najvećeg PE i najmanjeg PL dat će traženo odsijecanje (razmislite zašto), ali samo u slučaju da za te PE i PL vrijedi $PE < PL$; ako je $PE > PL$, linija je izvan područja odsijecanja. Kako smo na početku inicijalizirali $PE = 0$ i $PL = 1$, imat ćemo odsiječenu dužinu s obzirom na zadani konveksni poligon. Ako želimo načiniti odsijecanje pravca, a ne dužine, vrijednosti ćemo inicijalizirati na $PE = VrloMaliBroj$ i $PL = VrloVelikiBroj$.

8.10 Binarna podjela prostora BSP

Izgradnja stabla kroz koje organiziramo scenu u pojedine djelove značajno može doprinjeti ubrzanju postupaka kojima pretražujemo takve scenu. Jedna od najzančajnijih takvih organizacija je BSP (engl. *Binary Space Partitioning*) odnosno Binarna podjela prostora. Osnovna ideja izgradnje BSP stabla je rekurzivna binarna podjela prostora. Pogledajmo prvo problem u 2D prostoru. Neka početna scena sarži proizvoljan broj bridova ili geometrijskih likova koji se sastoje od bridova. Početni korak je da odaberemo bilo koji brid i pravcem na kojem leži taj



Slika 8.24: Razvrstavanje sjecišta na PE i PL.

brid binarno podijelimo prostor na dio koji je iznad promatranog pravca i dio koji je ispod. Slučaj kada je točka točno na promatranom pravcu može se pridružiti jednom od ova dva osnovna slučaja. Istovremeno s ovom podjelom gradimo stablo i definiramo početni čvor (korijen) stabla. U taj čvor, i bilo koji čvor koji ćemo kasnije napraviti, spremimo oznaku brida kojeg promatramo (ID brida), jednadžbu pravca na kojem je promatrani brid kojim smo ostvarili podjelu, te načinimo dva pokazivača. Jedan pokazivač pokazuje na dio stabla koji je vezan uz prostor 'iznad', a drugi uz dio prostora 'ispod' promatranog pravca. U shemi koju koristimo za prikaz stabla prvi pokazivač označimo s $+$, a drugi s $-$. Ako smo ovom početnom podjelom presjekli bilo koji drugi brid tada od takvog brida u stvari radimo dva nova od kojih je jedan dio iznad a drugi ispod početnog pravca. Ovom početnom podjelom razvrstamo i sve ostale bridove scene uključujući i ove nove dobivene dijeljenjem presiječenih bridova u dvije skupine. To su skupine bridova iznad početnog pravca i skupina bridova ispod početnog pravca.

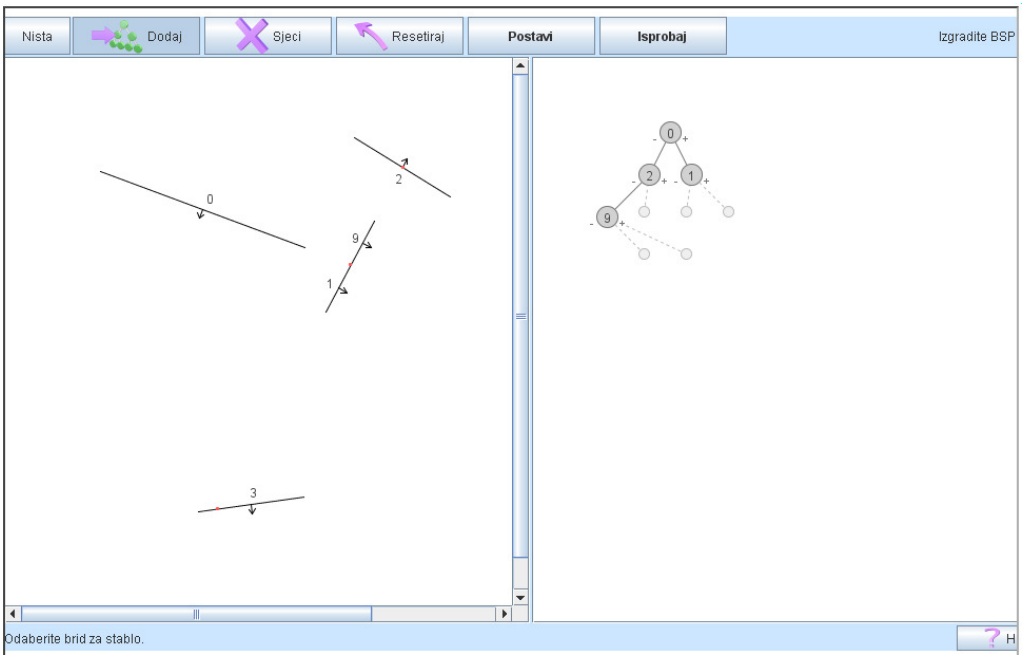
Na slici 8.25 možemo vidjeti da je odabran prvo brid 0, te je postavljan desno na istoj slici kao korijen stabla. Brid koji se nalazi neposredno desno-dolje od promatranog brida presječen je na dva dijela od kojih je jedan označen s 1, a drugi s 9. Kako će se ovi bridovi naći na različitim mjestima u stablu i njihova će jednadžba pravca biti zapravo dva puta zapisana. Već nakon ove prve podjele znamo da će se bridovi 2 i 9 nalaziti s lijeve strane stabla zato što su 'ispod' brida 0, a bridovi 1 i 3 bit će s desne strane, gdje je i oznaka podstabla $+$. Brojevi pridruženi pojedinim oznakama bridova ovdje su proizvoljni (nisu bitni). Ovaj osnovni princip dijeljenja se dalje nastavlja s tim da sada više ne promatramo cijelu ravninu, već promatramo odvojeno dvije poluravnine koje smo dobili prvom podjelom. Za lijevo podstablo i daljnu podjelu odabran je brid 2, a brid 9 se nalazi s njegove 'negativne strane', pa su tako i smješteni u stablu. U desnom podstablu odabran je brid 1, za koji vidimo da će pravac na kojem leži presjeći brid 3 u

crvenoj točkici na bridu. U ovom primjeru postupak nije dovršen. Potrebno bi bilo još ostvariti podjelu brida 3 na dva dijela i jedan od njih postaviti s pozitivne, a drugi s negativne strane brida 1. Time bi izgrađeno stablo bilo potpuno.

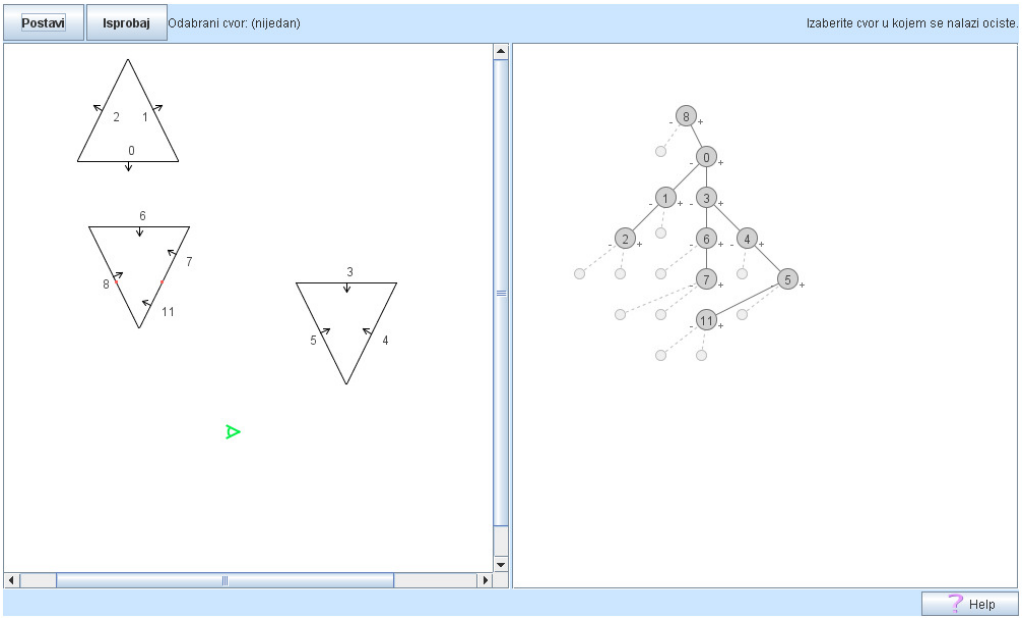
Možemo primijetiti da će odabir redoslijeda bridova utjecati na izgradnju stabla. Biramo li bridove koji se nalaze posred scene, stablo će biti balansirano, dok ako bridove biramo s rubnih dijelova, stablo će postati lista. Naravno, poželjno je da stablo bude balansirano.

Kada jednom scenu podijelimo i načinimo pripadno stablo u prostoru smo dobili niz konveksnih podprostora koji pripadaju završnim čvorovima stabla. Za odbir pravaca kojima obavljamo podjelu nije nužan uvijet da na njima jesu bridovi, već možemo odabrati i proizvoljne pravce u prostoru. Bridove biramo jer obično uzrokuju najmanje dodatnih presijecanja. Specijalan slučaj može nastupiti ukoliko je brid kojeg trebamo razvrstati već na pravcu koji nam dijeli prostor, no tada je njegova jednadžba već ionako u zapisana u promatranom čvoru, pa je potrebno zapisati i oznaku toga brida.

Čemu sad služi stablo koje smo napravili? Jedna od osnovnih primjena je za određivanje u kojem od podprostora se nalazi točka $V(x, y, z)$. Na prvi pogled, ovo djeluje malo zbunjujuće, pa naravno da vidimo gdje se nalazi točka. No objasniti to računalu i nije baš tako jednostavno, posebno za veće scene. Is-



Slika 8.25: Izgradnja BSP stabla. Postupak nije dovršen, brid 3 još treba podijeliti i smjestiti u stablo http://www.zemris.fer.hr/predmeti/irg/BSP/.



Slika 8.26: Traženje pozicije točke (očišta) u BSP stablu i pripadne pozicije u sceni http://www.zemris.fer.hr/predmeti/irg/BSP/.

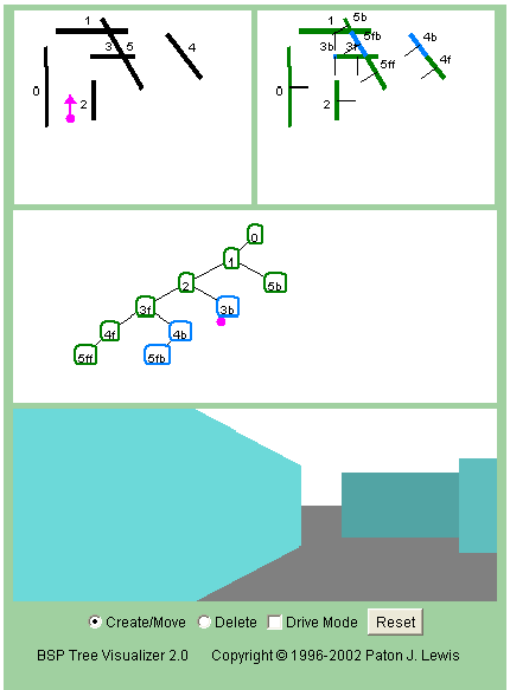
pitivanje gdje se nalazi točka korištenjem stabla je jednostavno. Uvrštavanjem koordinata točke u pohranjenu jednadžbu pravca na kojem leži brid u pojedinom čvoru dat će odgovor je li točka 'ispod' ili 'iznad' pojedinog pravca pa ovisno o tom rezultatu krećemo na lijevu ili desnu stranu podstabla, sve dok ne dođemo do podprostora gdje se točka doista nalazi.

Na slici 8.26 točka čiji položaj ispitujemo je očište i označena je malim zelenim okom na slici. Potrebno je redom uvrštavati koordinate točke u jednadžbe pravaca zapisanih u stablu. Evo primjera. Uvrstimo najprije koordinate promatrane točke u jednadžbu pravca zapisanu u korijenu stabla, odnosno u jednadžbu u čvoru '8'; rezultat će biti pozitivan pa zaključimo da je točka iznad pravca (pa u stablu slijedimo desnu granu). Potom koordinate točke uvrštavamo u jednadžbu pravca zapisanog u čvoru '0'; rezultat će biti pozitivan pa opet idemo u desno podstablo. Postupak nastavljamo stoga s čvorom '3' (dobivamo pozitivan rezultat pa idemo desno), s čvorom '4' (dobivamo pozitivan rezultat pa idemo desno), s čvorom '5' (dobivamo negativan rezultat pa idemo lijevo), te konačno s čvorom '11' pri čemu dobivamo negativan rezultat; stoga je konačni čvor određen kao lijevo dijete čvora '11'.

Na slici bi mogli i iscrtati konveksni poligon koji predstavlja cijelo područje koje pripada tom čvoru i vidjeli bi da je doista promatrana točka u tom poligonu. U općem sličaju, ako imamo stotinjak poligona kakvi su na slici 8.26, rezultat

će biti neki od listova stabla, no nas obično u konačnici zanima je li to neki od poligona ili se radi o prostoru oko poligona, što nas obično zanima kod detekcije kolizije. Pri tome se uporabom navedenog postupka svi objekti razmjeste u podprostore određene izgrađenim stablom pa se pri ispitivanju kolizija radi ispitivanje samo između objekata koji su u istom podprostoru; time se mogu dobiti značajna ubrzanja. Općenito gledano, broj ispitivanja ovisit će o tome koliko je dobro balansirano stablo i naravno, kolika je scena.

Prelazak u 3D prostor samo proširuje razmatranje na poligone umijesto bridova i pripadne ravnine umijesto pravaca, koje opet prostor dijele na dva dijela. Pripadno stablo će biti binarno stablo, a dijelovi prostora će biti konveksni poliedri. Primjer je prikazan na slici 8.27 gdje je scena sačinjena od niza 'zidova' prikazanih na slici gore lijevo. Na istoj slici gore desno su prikazani podijeljeni bridovi. Treba primijetiti da redoslijed odabira bridova pri izgradnji stabla određuje koji će bridovi biti dijeljeni. Pripadno stablo je prikazano ispod, a 3D prikaz prostora koji u konačnici izgleda kao labirint, je prikazan dolje. Pomicanjem očišta bit će moguća interaktivna šetnja kroz scenu. Osnovni koncept izgradnje i primjene BSP stabla proširiv je i dalje na više dimenzije.



Slika 8.27: Primjer 3D scene sačinjene od niza 'zidova' koji predstavljaju prostor scene. Izvor: http://symbolcraft.com/graphics/bsp/index.php

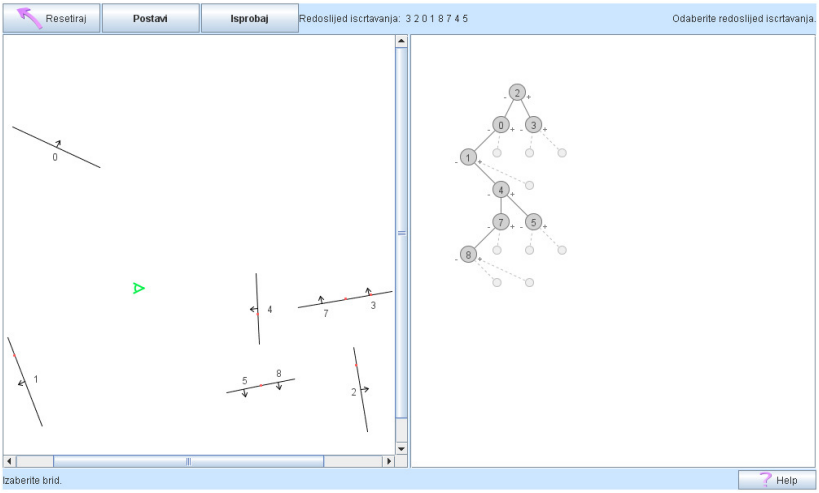
Sljedeća primjena sagrađenog BSP stabla je za sortiranje poligona s obzirom na udaljenosti od očišta od najdaljeg prema najbližem BTF (Back To Front). Pokušate li ovo napraviti korištenjem nekog od uobičajenih načina sortiranja, vrlo brzo ćete spoznati da je problem vrlo nezgodan. Promatrano u 2D prostoru, jasno je da svaki brid ima po dvije točke od kojih svaka ima po dvije koordinate što treba uzeti u razmatranje, a i da se promatrani bridovi mogu sjeći. Čaki i uspješno rješenje sortiranja bridova na klasičan način u pravilu će dati prilično spora rješenja. Zato je za primjenu sortiranja bridova (poligona) upotreba BSP stabla vrlo značajna. Postupak sortiranja se temelji na provjeri određivanja položaja očišta u stablu, slično kao kod upravo opisanog postupka određivanja točke u stablu (Slika 8.26), i postupanja prema sljedećem algoritmu.
Pri tome obilaženje znači i iscratvanje pripadnih bridova. Detaljniji algoritam prikazan je u nastavku.

1. Ako je očište s pozitivne strane obiđi negativnu stranu, pa čvor, pa pozitivnu stranu.

2. Ako je očište s negativne strane obiđi pozitivnu stranu, pa čvor, pa negativnu stranu.

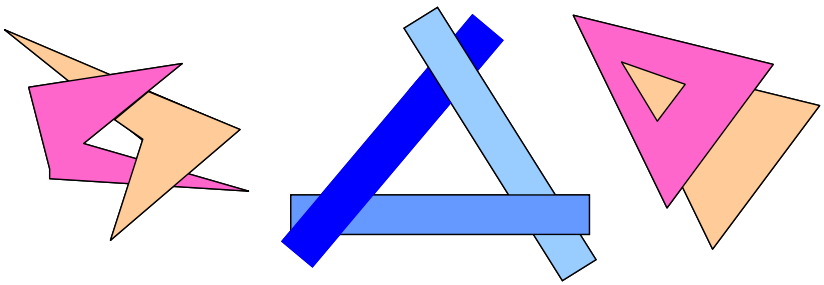
```
BSP_display(BSP_tree tree) {
  if (!EMPTY(tree)) {
    if (observer is located on front of root) {
                    BSP_display(tree->backChild);
                    displayPolygon(tree->root);
                    BSP_display(tree->frontChild);
    } else {
                    BSP_display(tree->frontChild);
                    displayPolygon(tree->root);
                    BSP_display(tree->backChild);
    }
  }
}
```

Primjer sortiranja poligona prikazan je na slici 8.28. Krećemo s pozicijom očišta i korjenom stabla, te zaključujemo da se očište nalazi s negativne strane brida 2, te moramo prvo obići suprotnu stranu, znači čvor 3 pa čvor 2, pa tek onda negativnu stranu. Time će iscrtani bridovi biti 3 pa 2. Dalje posjećujemo čvor 0, a očište je s negativne strane. Obilazimo pozitivnu stranu koja je prazna, čvor 0 i pripadni brid, te dalje negativnu stranu gdje je čvor 1. Slično kao maloprije očište je s negativne strane pa je sljedeći iscrtani brid 1, a posjećeni čvor 4. Za čvor 4 očište je s pozitivne strane pa idemo do čvora 7 gdje je očište isto s pozitivne strane što nas vodi na iscrtavanje čvora 8, povratak na 7 i njegovo iscrtavanje, povratak na 4 i njegovo isctravanje, te konačno do čvora 5 čime je postupak završen. Konačan redoslijed iscrtanih bridova je 32018745.



Slika 8.28: Sortiranje poligona obzirom na udaljenost od očišta.

Zanimljivo je da će navedeni algoritam uspješno prikazati i slučajeve koji su problematični za većinu ostalih algoritama za uklanjanje skrivenih linija i površina prikazanih na slici 8.29. Kada bi htjeli ove poligone iscrtati jedan po jedan od najdaljeg prema najbližem, ne bi mogli odrediti redoslijed kojim je to potrebno učiniti, a da scena bude ispravno prikazana. BSP algoritam, će upravo zbog dijeljenja bridova, odnosno poligona koje je sastavni dio algoritma ispravno prikazati bilo koju od prikazanih scena.



Slika 8.29: Tipični problematični slučajevi kod postupaka uklanjanja skrivenih linija i površina.

8.11 Ponavljanje

1. Na koji se način obavlja detekcija (a time omogućava i uklanjanje) stražnjih poligona u prostoru scene? Navedite dva načina.
2. Što su i čemu služe minimaks provjere?
3. Čemu služi postupak Watkinsa?
4. Što je, kako se koristi i čemu služi Z-spremnik?
5. Čemu služi postupak Warnocka?
6. Usporedite Z-spremnik, postupak Watkinsa te postupak Warnocka.
7. Čemu služe četvero i oktalna stabla?
8. Čemu služi i kako se provodi algoritam Cohen Sutherlanda?
9. Čemu služi i kako se provodi algoritam Cyrus Beck?
10. Čemu služi kako se gradi BSP?

Poglavlje 9

Osvjetljavanje

9.1 Osvjetljavanje

9.1.1 Uvod

U prethodnim poglavljima naučili smo dovoljno da znamo kako iscrtati žičani model objekta na zaslonu. Sada je došlo vrijeme da se upoznamo s načinima kako u scenu uvesti svjetlosne izvore i analizirati njihov utjecaj na objekte. U prirodi, interakcija svjetlosti s površinom izuzetno je složena i stvaranje modela koji bi u potpunosti obuhvatio sve aspekte te interakcije ne bi bilo izvedivo. Stoga su razvijeni različiti modeli koji obuhvaćaju pojedine aspekte u većoj ili manjoj mjeri.

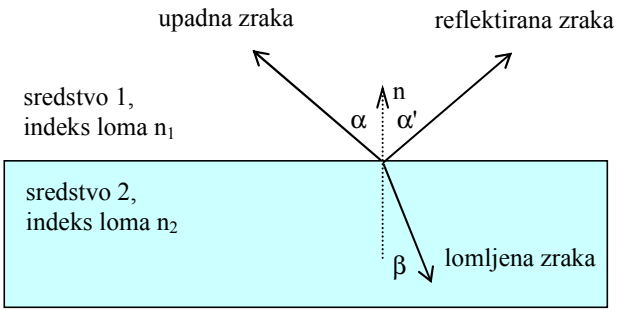
9.1.2 Fizikalni model svjetlosti

Postoje različite interpretacije, razvijane tijekom stoljeća, koje opisuju propagaciju svjetlosti kroz medij i interakciju s površinom. Tri su osnovne interpretacije koje se temelje na: geometrijskoj optici, fizikalnoj optici i kvantnoj optici.

U okviru **geometrijske optike** svjetlost se predstavlja skupom svjetlosnih zraka. Geometrijska optika se temelji na četiri osnovna zakona, a to su zakoni o pravocrtnom širenju svjetlosti, neovisnost svjetlosnih snopova, zakon obijanja (refleksije) i zakon loma (refrakcije) svjetlosti (slika 9.1). Zakon refleksije kaže da je kut između upadne zrake i normale n jednak kutu između reflektirane zrake i normale ($\alpha = \alpha'$). A zakon refrakcije (Snellov zakon) se odnosi na kut između lomljene zrake i normale (β) obzirom na upadni kut zrake (α) i vrijedi:

$$\frac{\sin(\alpha)}{\sin(\beta)} = \frac{v_1}{v_2} = \frac{n_2}{n_1}, \tag{9.1}$$

gdje su n_1 i n_2 indeksi loma pojedinih sredstava a v_1 i v_2 brzine širenja vala u pojedinim sredstvima. Prelaskom iz jednog sredstva u drugo mijenja se brzina širenja vala, a kako je indeks loma definiran kao omjer brzine svjetlosti naprema brzini širenja vala u sredstvu ($n = c/v$), vrijedi dani zakon.

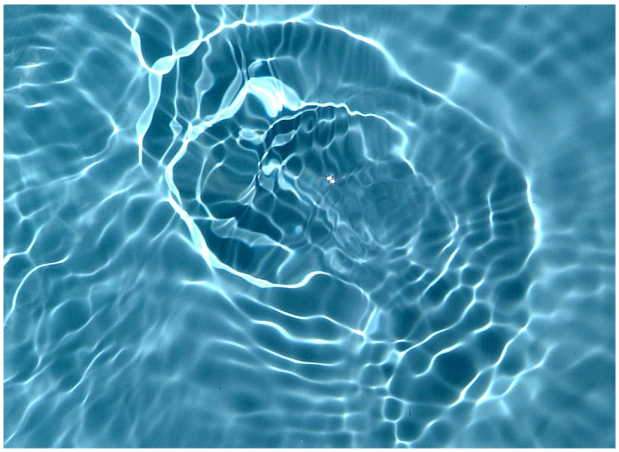


Slika 9.1: Refleksija i refrakcija svjetlosti na površini.

Ove osnove se koriste u proučavanju ponašanja leća, zrcala i općenito optičkih sustava. Navedeni zakoni nam predstavljaju osnovu u svakom modelu kojim opisujemo ponašanje svjetlosti.

Svjetlost u okviru **fizikalne optike** je shvaćena kao elektromagnetsko zračenje, odnosno širenje svjetlosti shvaća se kao širenje progresivnog, odnosno transverzalnog vala. Ovo znači da se kod svjetlosti električno i magnetsko polje vala mijenjaju periodički u svim smjerovima okomitim na smjer gibanja vala koji se širi. Pojave kao što su difrakcija (ogib svjetlosti) i interferencija ne mogu se objasniti geometrijskom optikom, već se objašnjavaju fizikalnom optikom. Osnovni princip rada LCD monitora također se zasniva na polarizirajućim svojstvima materijala i svjetlosti kao elektromagnetskom valu. Sunčeva svjetlost je val polariziran u svim smjerovima, odnosno titranje vala je prisutno u svim smjerovima, dok polarizirajući slojevi u LCD-u propuštaju samo horizontalno ili vertikalno polariziranu komponentu. Propuštanjem ili blokiranjem polarizirane svjetlosti postiže se vidljivost pojedinog slikovnog elementa. Šareni uzorci na mjehuriću sapunice također su primjer iz stvarnog života gdje valna priroda i interferencija svjetlosti dolazi do izražaja. Lijep vizualni učinak koji na određenom mjestu stvara fronta svjetlosnog vala zove se kaustika. Učinak kaustuke vidljiv je, na primjer, na dnu bazena ili mora za sunčanog vremena (Slika 9.2).

Treći aspekt ponašanja svjetlosti temelji se na čestičnoj prirodi svjetlosti, odnosno **kvantnoj optici**. Ovaj aspekt ponašanja svjetlosti vidjiv je kada čestična priroda svjetlosti temeljena na česticama zvanim *fotoni* dolazi do izražaja. Pojedina čestica nosi kvant energije koji je jednak hf, gdje je h Planckova konstanta, a f frekvencija svjetosti. Ovaj koncept posebno nam je zanimljiv kod razumijevanja subtraktivnog sustava boja, o kojem će biti riječ kasnije. U ovom sustavu boja iz skupa frekvencija prisutnih u dolaznoj svjetlosti na neku obojenu površinu,



Slika 9.2: Primjer učinka kaustike na površini vode.

neke će frekvencije biti absorbirane ("oduzete"), dok će ostale biti reflektirane. Boja površine, odnosno atomi koji ju čine sposobni su apsorbirati samo određene frekvencije te je time određeno i što će biti reflektirano, odnosno koju ćemo boju u konačnici vidjeti na danoj plohi.

Ovo su samo neki primjeri zanimljivih učinaka koje ćemo primijetiti promatrajući svjetlost, no i na vrlo jednostavnim površinama svjetlost koja dolazi, ovisno o valnim duljinama, raspršit će se različito (anizotropno) u pojedinim smjerovima, prodrijet će do neke dubine u površinu i tek će se onda reflektirati što je tipično za ljudsku kožu. Tako da postavljeni fizikalni zakoni daju tek osnovu ponašanja svjetlosti a u izgradnji modela kojim želimo vjerno reproducirati svu pojavnost njena ponašanja u svijetu koji nas okružuje susrećemo se s vrlo izazovnim zadatkom. Problem u izgradnji modela ne stvara samo interakcija svjetlosti s jednom površinom, već svaka površina u prostoriji svjetlost koju dolazi do nje i apsorbira i reflektira do bilo koje druge površine u prostoriji bilo direktno bilo indirektno, te se tu susrećemo s rekurzivnim problemom. Međusobna interakcija svjetlosti nad objektima prisutna je ukoliko u sceni imamo više od jednog objekta ili ako je objekt konkavan. Naime, dio svjetlosti koji osvjetljava objekt A dolazi do objekta B (bilo refleksijom, bilo raspršenjem i sl.) i osvjetljava ga. Objekt B opet dio te svjetlosti odbija prema objektu A; objekt A opet dio šalje objektu B, i – vjerojatno ste shvatili. Zbog ove kompleksnosti uvode se modeli koji u određenoj mjeri uvažavaju pojedine fizikalne zakone i interakcije objekata, te daju koliko-toliko zadovoljavajuće rezultate.

Modeli kojima se aproksimira ponašanje svjetlosti u računalnog grafici temelje se na iznesenim fizikalnim osnovama i u većoj ili manjoj mjeri obuhvaćaju

navedena fizikalna ponašanja. Vremenom su razvijani sve složeniji modeli kojima se aproksimira ponašanje svjetlosti, a oni koji nisu u početku bili ostvarivi u stvarnom vremenu razvojem računalne opreme to postaju.

No, odnekud moramo početi, pa to je najjednostavniji model i prva aproksimacija složenog ponašanja koje smo opisali. Općenito, **modelom osvjetljenja** određujemo kako izračunati intenzitet svjetlosti u promatranoj točki, a **postupkom sjenčanja** određujemo kako osjenčati površine uz odabrani model osvjetljenja.

Promatrano s aspekta fizikalne optike svjetlost je samo jedan dio spektra zračenja, te i za nju vrijedi općenito:

$$\text{Upadna svjetlost} = \sum \begin{matrix} \text{reflektirana} \\ \text{rasprsena} \\ \text{apsorbirana} \\ \text{transmitirana} \end{matrix}$$

Svaka od ovih komponenti ovisna je o samim fizikalnim svojstvima materijala, o grubоći površine, o valnoj duljini same svijetlosti i još mnoštvu faktora. Među najjednostavnije modele spada i Phongov model osvjetljavanja koji ćemo obraditi u nastavku. Ovaj model predstavlja prvu aproksimaciju i još uvijek je najkorišteniji model, a zove se još i empirijski model.

9.2 Phongov model osvjetljenja

9.2.1 Općenito o modelu

Model pretpostavlja da se cjelokupno osvjetljenje objekta, i svakog njegovog djelića, može opisati kao linearna kombinacija triju komponenti:

- ambijentna komponenta,
- difuzna komponenta, te
- zrcalna komponenta.

Uz to, svjetlosni izvori s kojima ovaj model barata uglavnom su točkasti. Kako u takvom scenariju sva svjetlost dopire iz jedne ili više točaka, govorimo o jednom ili više točkastih izvora.

9.2.2 Ambijentna komponenta

Ambijentna komponenta rezultat je interakcije svjetlosti među svim objektima u sceni opisane u uvodu. Znači, pretpostavlja se da je reflektirana svjetlost koja dolazi do neke površine od svih okolnih površina konstantnog iznosa. Površine

koje nisu pod izravnim utjecajem svjetlosti bit će u stvarnosti različito osvijetljene. Ako zavirimo ispod stola i promotrimo donju stranu radne plohe, sigurno će biti manje rasvijetljena nego na primjer bočna strana stola, uz uvijet da obje plohe nisu pod izravnim utjecajem svjetlosti. No u ovom modelu, za ambijentnu komponentu uzima se konstantna:

$$I_g = k_a \cdot I_a, \tag{9.2}$$

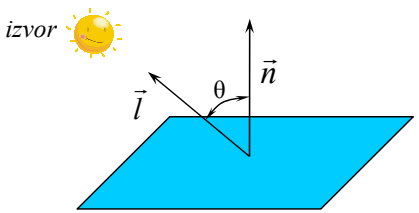
gdje je k_a koeficijent između 0 i 1 ($0 \leq k_a \leq 1$) i daje za ovu komponentu koliko svjetlosti I_a će biti apsorbirano, a koliko reflektirano. Usprkos ovako gruboj aproksimaciji, rezultati su prihvatljivi. Ambijentna komponenta osigurava da površine koje su stražnje u odnosu na izvor svjetlosti ne budu potpuno crne, kakve bi inače bile da ove komponente nema.

9.2.3 Difuzna komponenta

Iz fizike je poznato da intenzitet svjetlosti koji reflektira elementarna površina tijela ovisi o kutu pod kojim upada svjetlost u odnosu na normalu te elementarne površine (Lambetov zakon). Pri tome, prema slici 9.3 vrijedi:

$$I_d = I_i \cdot k_d \cdot \cos(\theta), \tag{9.3}$$

gdje je I_i intenzitet točkastog izvora, a k_d empirijski koeficijent refleksije ovisan o valnoj duljini upadne svijetlosti ($0 \leq k_d \leq 1$). θ je kut između normale površine i vektora od promatrane točke prema izvoru.



Slika 9.3: Određivanje difuzne komponente svjetlosti.

Pogledajmo sliku 9.3 i na njoj označene vektore. Vektor $\vec{l}$ predstavlja jedinični vektor iz točke koju promatramo na površini i usmjeren je prema izvoru. Vektor $\vec{n}$ je jedinični vektor koji predstavlja normalu u promatranoj točki. Uz te oznake, kosinus kuta θ možemo izraziti skalarnim produktom tih vektora, pa relacija (9.3) prelazi u:

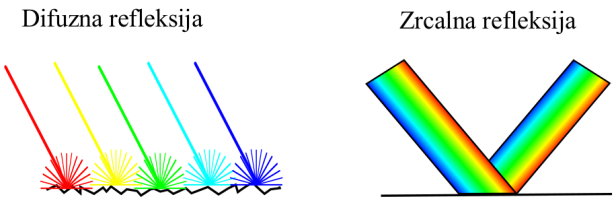
$$I_d = I_i \cdot k_d \cdot (\vec{l} \cdot \vec{n}). \tag{9.4}$$

Ako I_d ispadne negativan zbog $\vec{l} \cdot \vec{n} < 0$, tada se za I_d uzima 0, drugim riječima promatrana točka za taj slučaj nije vidljiva iz izvora i zato nema doprinosa ove komponente. Stoga se uzima $\max(\vec{l} \cdot \vec{n}, 0)$.

Ako u sceni ima više točkastih izvora, tada se za ukupni intenzitet u promatranoj točki može pisati:

$$I_d = k_d \cdot \sum_m I_{i,m} \cdot \max(\vec{l}_m \cdot \vec{n}, 0). \tag{9.5}$$

Valja primijetiti da difuzna komponenta ne ovisi o položaju promatrača, već samo o položaju izvora prema promatranoj površini. Njezin utjecaj dominira na hrapavim površinama. Svjetlost koja dolazi do hrapave površine, raspršuje se podjednako u svim smjerovima kao što je prikazano na slici 9.4 (lijevo). Znači, mikrostruktura, odnosno hrapavost površine utjecat će na ovu komponentu.



Slika 9.4: Na hrapavoj površini dominira difuzna komponenta (lijevo), a na glatkoj površini zrcalna komponenta (desno).

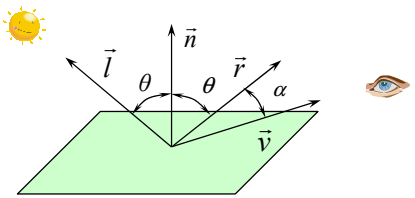
9.2.4 Zrcalna komponenta

Zrcalna komponenta, prema slici 9.5, funkcija je kuta α između reflektirane zrake $\vec{r}$ i zrake prema promatraču $\vec{v}$.

$$I_s = I_i \cdot k_s \cdot \cos^n(\alpha) \tag{9.6}$$

pri čemu je I_i intenzitet izvora, k_s koeficijent ovisan o materijalu ($0 \leq k_s \leq 1$). Koeficijent n u eksponentu (ne treba ga miješati s vektorom normale $\vec{n}$) je indeks (skalar) koji opisuje grubоću površine i njezina reflektirajuća svojstva.

Prema slici 9.5 vektor $\vec{r}$ predstavlja vektor reflektirane zrake i nalazi se u ravnini koju tvore vektori $\vec{l}$ i $\vec{n}$, pod istim je kutem prema $\vec{n}$ kao što ga zatvaraju vektori $\vec{l}$ i vektor $\vec{n}$. Vektor $\vec{v}$ predstavlja vektor usmjeren iz točke površine prema promatraču (oku). Vektori $\vec{r}$ i $\vec{v}$ također su jedinični vektori. U tom slučaju kut α može se opisati skalarnim produktom $\vec{r} \cdot \vec{v}$, pa izraz (9.6) prelazi u:

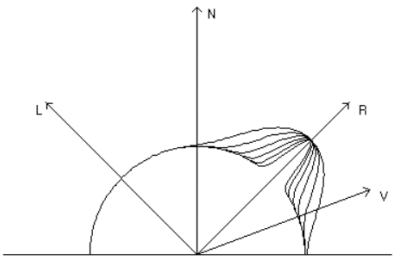


Slika 9.5: Određivanje zrcalne komponente svjetlosti.

$$I_s = I_i \cdot k_s \cdot (\vec{r} \cdot \vec{v})^n \tag{9.7}$$

Na slici 9.5 svi vektori su prikazani u istoj ravnini. U općem slučaju, vektor prema promatraču $\vec{v}$ neće ležati u istoj ravnini s ostalim vektorima. Primjer numeričkog izračuna vektora $\vec{r}$ imali smo u poglavlju 2. u primjeru s vektorima.

Zrcalna komponenta omogućava nam postizanje efekta blještavila. Naime, ovisno o faktoru n kut između promatrača i reflektirane zrake imat će različiti utjecaj na konačan intenzitet promatrane točke. Ako n teži u beskonačnost, površina se ponaša kao zrcalo. Naime, tada će ova komponenta postojati samo u smjeru reflektirane zrake – a to je upravo karakteristika idealnog zrcala. Ako je pak površina nešto grublja, tada ćemo imati rasipanje svjetlosti i u malim kutevima oko reflektirane zrake. To opisujemo konačnim n-om. Slika 9.6 pokazuje utjecaj indeksa n na intenzitet ove komponente uzevši u obzir položaj vektora $\vec{v}$.



Slika 9.6: Utjecaj faktora n na zrcalnu komponentu svjetlosti.

Slika je generirana za $n = 10, 20, 40, 80, 160$ i 320. Uz $n = 10$ dobivena je vanjska krivulja što pokazuje da je za male vrijednosti indeksa n ova komponenta prisutna u širokom spektru kuteva oko reflektirane zrake. Najveći n (320) generirao je najužu krivulju. Iz ovoga proizlazi da se spektar kuteva u kojima ova komponenta ima utjecaja smanjuje i teži prema kutu $\alpha = 0$, u kojem bi slučaju komponenta postojala samo u smjeru reflektirane zrake.

9.2.5 Ukupan utjecaj

Ukupan utjecaj iskazuje se kao linearna kombinacija sve tri komponente:

$$I = I_g + I_d + I_s = I_a \cdot k_a + I_i \cdot \left(k_d \cdot (\vec{l} \cdot \vec{n}) + k_s \cdot (\vec{r} \cdot \vec{v})^n \right) \tag{9.8}$$

Radi jednostavnosti, model može pretpostaviti da se svjetlosni izvor, kao i promatrač, nalaze u beskonačnosti. Ova pretpostavka rezultira konstantnim vrijednostima vektora $\vec{l}$ i $\vec{v}$ kroz čitavu scenu, čime se izbjegava potreba za računanjem istih u svakoj promatranoj točki, pa se cijeli postupak ubrzava. No ovo povlači za posljedicu da će proizvoljno velikoj površini u svakoj točki biti pridjeljen isti vektor prema izvoru i promatraču. U model se još uvodi ovisnost

intenziteta o udaljenosti izvora do površine. Fizikalno intenzitet svjetlosti opada s kvadratom udaljenosti, no eksperimentalno se to pokazalo u grafičkim primjenama kao prejak utjecaj, te se obično uzima linearna ovisnost opadanja utjecaja s udaljenošću. Tada se relacija (9.8) modificira u:

$$I = I_g + \frac{I_d + I_s}{d + k} = I_a \cdot k_a + \frac{I_i \cdot \left(k_d \cdot (\vec{l} \cdot \vec{n}) + k_s \cdot (\vec{r} \cdot \vec{v})^n\right)}{d + k} \tag{9.9}$$

pri čemu je d udaljenost promatrane točke od točke izvora, a konstanta k u nazivniku je veća od nule i sprječava dijeljenje s nulom u slučaju da je $d = 0$.

Ako se za opis svjetlosti koriste R, G i B komponente, tada izraz (9.9) treba primijeniti na sve tri komponente, pri čemu uz uobičajenu fizikalnu interpretaciju k_s nije ovisan o komponenti svjetlosti (odnosno ima istu vrijednost za sve tri komponente). Možemo pisati:

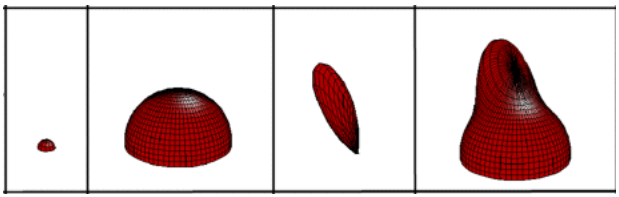
$$I_r = I_{a,r} \cdot k_{a,r} + \frac{I_{i,r} \cdot \left(k_{d,r} \cdot (\vec{l} \cdot \vec{n}) + k_s \cdot (\vec{r} \cdot \vec{v})^n\right)}{d + k} \tag{9.10}$$

$$I_g = I_{a,g} \cdot k_{a,g} + \frac{I_{i,g} \cdot \left(k_{d,g} \cdot (\vec{l} \cdot \vec{n}) + k_s \cdot (\vec{r} \cdot \vec{v})^n\right)}{d + k} \tag{9.11}$$

$$I_b = I_{a,b} \cdot k_{a,b} + \frac{I_{i,b} \cdot \left(k_{d,b} \cdot (\vec{l} \cdot \vec{n}) + k_s \cdot (\vec{r} \cdot \vec{v})^n\right)}{d + k} \tag{9.12}$$

Prostorna razdioba pojedinih komponenti, te njihov ukupni doprinos prikazan je na slici 9.7. Vidljivo je da je doprinos ambijentne komponente relativno malen, difuzna komponeta obično dominira a nju upotpunjuje prostorno pozicionirana oko reflektirane zrake zrcalna komonenta. Karakteristike same površine utjecat će onda na doprinos pojedinih komponenti, a nama ostaje priličan broj različitih koeficijenata za podešavanje i postizanje željenog izgleda površine.

Utjecaj pojedinih komponenata ali razmatran u 3D prostoru prikazan je na slici 9.7. Najlijeviji dio slike prikazuje prostornu razdiobu ambijentne komponente; ona se raspršuje u svim smjerovima podjednako i relativno je malog intenziteta s obzirom na ostale komponente. Središnji-lijevi dio slike prikazuje prostornu razdiobu difuzne komponente. Njezin iznos je značajno veći od ambijentnog intenziteta i također je jednoliko raspršena u prostoru. Središnji-desni dio slike prikazuje prostornu razdiobu zrcalne komponente. Njezina razdioba određena je vektorom smjera reflektirane komponente svjetlosti. Udio ove svjetlosti koju vidi promatrač izravno je određena kutem između vektora smjera reflektirane zrake i vektora prema promatraču. Konačno, desni dio slike prikazuje ukupnu prostornu razdiobu koja odgovara zbroju svih prethodnih triju komponenata. Poseban slučaj koji bismo dobili presjekom prikazane razdiobe ravninom prikazan je na slici 9.6.



Slika 9.7: Prostorna razdioba ambijentne, difuzne, zrcalne i ukupne sume svih komponenti (s lijeva na desno).

9.2.6 Primjer

Pogledajmo kako se ovaj model ponaša u najjednostavnijim slučajevima. U scenu ćemo postaviti kuglu u ishodište 3D sustava. Za prijelaz u 2D sustav koristit ćemo paralelnu projekciju, uz promatrača koji se nalazi u $+\infty$ na z-osi i gleda prema ishodištu. Koristit ćemo jedan svjetlosni izvor, koji će se također nalaziti u beskonačnosti, usmjeru vektora [1 0 1]). Budući da gledamo odozgo, crtat ćemo samo gornji plašt kugle ($z \geq 0$).

Uz paralelnu projekciju preslikavanje iz 3D u 2D sustav vrlo je jednostavno: $x' = x$, $y' = y$, $z' = 0$. Crtanje kugle nije riješeno najefikasnije, no poslužit će za demonstraciju. Kako se cijela kugla projicirana u $z = 0$ ravninu svede na krug upisan kvadratu $-R \leq x \leq R$ i $-R \leq y \leq R$, funkcija provjerava za svaku točku te površine pripada li krugu; ako ne, ide se na slijedeću točku, a ako pripada, računa se intenzitet. Funkcija koja ovo računa prikazana je na sljedećem ispisu.

```
1  void __fastcall TForm1::Button3Click(TObject *Sender)
2  {
3    int x,y;
4    double z, I, Id, Is;
5    const int R = 100;
6    double nx, ny, nz;
7    double lx, ly, lz, rx, ry, rz, l_n_2, vx, vy, vz, naz;
8
9    lx = 1; ly = 0; lz = 1;
10   naz = sqrt(lx*lx + ly*ly + lz*lz);
11   lx = lx / naz; ly = ly / naz; lz = lz / naz;
12
13   vx = 0; vy = 0; vz = 1;
14
15   for( x=-R; x<=R; x++ ) {
16    for( y=-R; y<=R; y++ ) {
17     z = R*R - (x*x + y*y);
18     if( z < 0 ) continue;
19     z = sqrt(z);
20     nz = R;
21     nx = x / nz; ny = y / nz; nz = z / nz;
22     Id = lx*nx + ly*ny + lz*nz;
23     l_n_2 = 2*Id;
```

```
24        if( Id > 0. ) Id = 200*Id; else Id = 0.;
25        rx=l_n_2*nx-lx; ry=l_n_2*ny-ly; rz=l_n_2*nz-lz;
26        naz = sqrt(rx*rx + ry*ry + rz*rz);
27        rx = rx / naz; ry = ry / naz; rz = rz / naz;
28        Is = rx*vx + ry*vy + rz*vz;
29        if( Is > 0. ) {
30          Is = 45*pow(Is,80);
31        } else Is=0.;
32        I = 10. + Id + Is;
33        Image1->Canvas->Pixels[x+150][-y+150]=RGB(I,I/2,I);
34      }
35    }
36  }
```

Svaki izračunati slikovni element crta se na zaslonu, a budući da se dio kruga dobiva uz negativne vrijednosti x i y koordinata, ishodište je pomaknuto u točku $(150, 150)$. Dodatno se mijenja i orijentacija y-osi prema gore (na zaslonima se pozitivni y proteže prema dolje).

Funkcija pretpostavlja slijedeće vrijednosti:

- $I_i = 256$,
- $k_d = 0.78125 \quad \Rightarrow \quad I_d = I_i \cdot k_d = 200$,
- $k_s = 0.17578125 \quad \Rightarrow \quad I_s = I_i \cdot k_s = 45$,
- $I_g = I_a \cdot k_a = 10$, te
- $n = 80$.

Četiri slike uz različite vrijednosti faktora n prikazane su na slici 9.8.

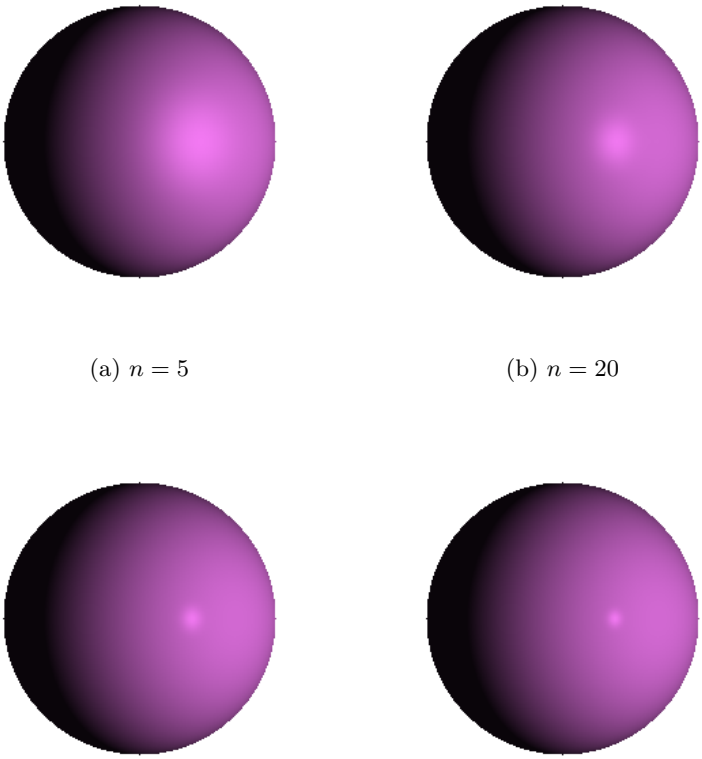
9.3 Konstantno sjenčanje poligona

Pri konstantnom sjenčanju poligona potrebno je za jednu točku poligona uporabom nekog od modela osvjetljavanja odrediti pripadni intenzitet. Ta točka može biti jedan od vrhova ili može biti (bolje!) središte poligona. Na temelju normale poligona i vektora prema izvoru svjetlosti određuje se intenzitet promatranog slikovnog elementa. Potom se taj intenzitet primijeni i na sve preostale točke površine poligona.

Prednost ovog modela je velika brzina jer zahtijeva da se intenzitet uporabom fizikalnog modela računa samo u jednoj točki po poligonu.

9.4 Gouraudovo sjenčanje poligona

Gourardovo sjenčanje poligona računski je zahtijevnije od konstantnog sjenčanja jer zahtjeva da se intenziteti osvjetljavanja izračunaju za sve vrhove poligona (što



(a) $n = 5$ (b) $n = 20$

(c) $n = 80$ (d) $n = 360$

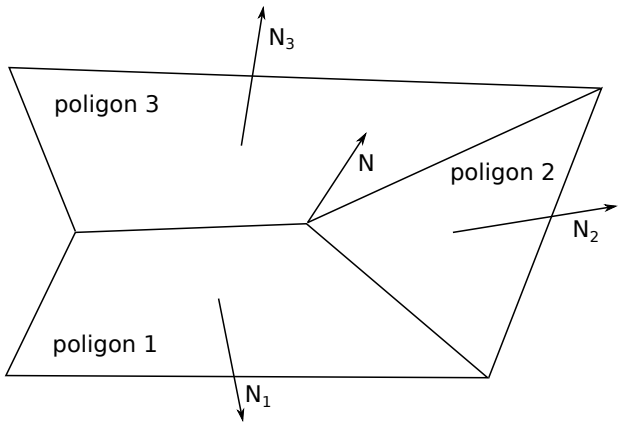
Slika 9.8: Utjecaj parametra n na osvjetljavanje kugle

i dalje nije previše jer su poligoni tipično trokuti pa imamo po tri izračuna po poligonu). Pri tome se može koristiti bilo koji od modela osvjetljanja a mi ćemo u nastavku opisati uporabu Phongovog modela osvjetljavanja. Nakon što su određeni intenzititeti vrhova poligona, potrebno je provesti interpolaciju intenziteta po ostatku poligona što se može učiniti računski učinkovitije no da se u svakoj od tih točaka računa puni model osvjetljavanja.

Phongov model već smo opisali u sekciji 9.2. Prema izrazu (9.4) difuzna komponenta u svakoj točki ovisi o skalarnom produktu vektora $\vec{L}$ i vektora $\vec{N}$.

Tijela (zapravo njihov plašt) u 3D prostoru obično modeliramo (aproksimiramo) nizom poligona. Sjenčanje tijela tada se svodi na sjenčanje poligona. Za svaki vrh potrebno je odrediti normalu u tom vrhu. Ako za tijelo imamo analitički model (npr. znamo da sjenčamo plašt kugle), tada iz samog modela možemo iz-

računati pripadnu normalu. Češće, međutim, tijelo imamo zadano isključivo kao niz poligona koji čine aproksimaciju njegovog plašta i nemamo analitički model. U tom slučaju, u svakom vrhu potrebno je pronaći srednju normalu (na temelju normala svih poligona koji dijele taj vrh), i pomoću nje izračunati intenzitet u tom vrhu. Srednja normala računa se kao aritmetička sredina normala svih poligona koji dijele taj vrh. Poželjno bi bilo da su sve normale jedinični vektori pa u tom slučaju nakon izračuna aritmetičke sredine dobiveni vektor još treba normirati. Slika 9.9 prikazuje primjer.



Slika 9.9: Izračun srednje normale u vrhu tijela

Uz sliku vrijedi relacija $\vec{N} = \frac{1}{3}\left(\vec{N}_1 + \vec{N}_2 + \vec{N}_3\right)$, dok općenito vrijedi:

$$\vec{N} = \frac{1}{n}\sum_{i=1}^{n}\vec{N}_i \tag{9.13}$$

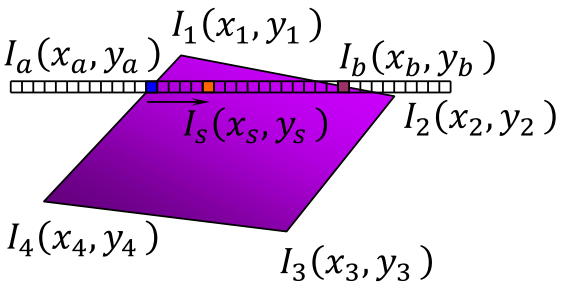
Intenzitet u vrhu računa se prema izrazu (9.8). U izrazu prikazanom u nastavku komponenta I_s je izostavljena zbog jednostavnosti ali to općenito ne mora biti tako. Uz to pojednostavljenje slijedi:

$$I = I_a \cdot k_a + I_i \cdot k_d \cdot (\vec{L} \cdot \vec{N}) \tag{9.14}$$

Nakon što ovo izračunamo za sve vrhove, krećemo na sjenčanje poligona. Sada svaki poligon, općenito govoreći, ima u svojim vrhovima različite intenzitete. Ove intenzitete treba najprije interpolirati uzduž svih bridova (npr. postupkom DDA). Nakon ovoga poznati su intenziteti u svakoj točki svakog brida poligona. Sada još treba te intenzitete interpolirati od lijevih bridova poligona do desnih bridova poligona (opet postupkom DDA). Ideja je pokazana na slici 9.10.

Za točku (x_a, y_s) intenzitet I_a se dobije interpolacijom između I_1 i I_4:

$$I_a = \frac{1}{y_4 - y_1}\left(I_1(y_4 - y_s) + I_4(y_s - y_1)\right).$$



Slika 9.10: Interpolacija intenziteta kod Gouraudovog sjenčanja

Za točku (x_b, y_s) intenzitet I_b se dobije interpolacijom između I_1 i I_2:

$$I_b = \frac{1}{y_2 - y_1} \left(I_1(y_2 - y_s) + I_2(y_s - y_1) \right).$$

Intenzitet točke (x_s, y_s) označit ćemo s I_s (ne treba ga miješati sa zrcalnim intenzitetom Phongovog modela) dobije se interpolacijom intenziteta I_a i I_b:

$$I_s = \frac{1}{x_b - x_a} \left(I_a(x_b - x_s) + I_b(x_s - x_a) \right).$$

Ovakav postupak računanja naziva se bilinearna interpolacija (naime, najprije se linearno interpoliraju intenziteti uzduž y-osi, a potom se radi interpolacija tih interpoliranih vrijednosti uzduž x-osi).

Budući da se intenzitet I_s računa za svaki piksel scan-linije koji se nalazi unutar poligona, postupak se može malo modificirati da bi se dobilo na brzini. Uvede se prirast ΔI_s:

$$\Delta I_s = \frac{\Delta x}{x_b - x_a}(I_b - I_a)$$

pa se intenzitet može računati prema:

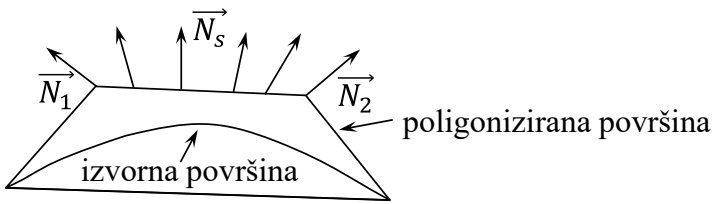
$$I_{s,n} = I_{s,n-1} + \Delta I_s$$

Pri tome Δx označava korak po x-osi. Ako popunjavamo svaki piksel, što je uobičajeno, Δx iznosi 1.

9.5 Phongovo sjenčanje

Ovo je još jedan postupak koji se zasniva na Phongovom reflekcijskom modelu. Postupak je za računalo zahtjevniji, ali daje bolje rezultate od Gouraudovog sjenčanja. Postupak zadržava bilinearnu interpolaciju, ali više se ne interpoliraju intenziteti, već same normale, da bi se na kraju za svaki piksel na temelju dobivene

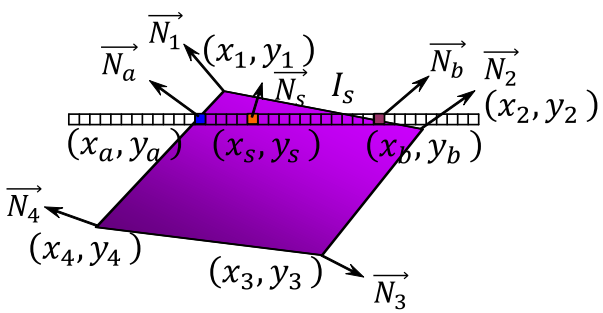
vrijednosti za normalu izvelo računanje intenziteta na temelju izraza (9.14). Sada je vidljivo zašto je postupak puno zahtjevniji. Međutim, dobra strana postupka jest privid da interpolirana normala slijedi zakrivljenosti površine. Slika 9.11 pokazuje primjer.



Slika 9.11: Interpolacija normala kod Phongovog sjenčanja

Zakrivljenu površinu pokušali smo prikazati poligonima. U tu svrhu generirano je nekoliko poligona tako da "slijede" zakrivljenost površine. Naravno, mi smo uzeli svega tri poligona (dok bi za koliko toliko realnu i glatku zakrivljenost trebalo uzeti puno više). Normale u zajedničkim vrhovima su N_1 i N_2. Ako normalu N_s dobijemo (bi)linearnom interpolacijom, dobiva se dojam da normala slijedi zakrivljenost izvorne površine, pa očekujemo i bolje rezultate od sjenčanja.

U postupak krećemo kao i kod Gouraudovog sjenčanja: potrebno je izračunati normale u svim vrhovima prema izrazu (9.13). Zatim se izvodi bilinearna interpolacija normala, tako da u točki (x_s, y_s) dobijemo normalu N_s. Posljednji korak je izračun intenziteta u toj točki prema izrazu (9.14) gdje za N uzimamo N_s. Slika 9.12 pokazuje postupak.



Slika 9.12: Interpolacija normala kod Phongovog sjenčanja, detaljniji primjer

Za točku (x_a, y_s) normala se dobije interpolacijom između N_1 i N_4:

$$N_a = \frac{1}{y_4 - y_1} \left(N_1(y_4 - y_s) + N_4(y_s - y_1) \right)$$

Za točku (x_b, y_s) normala se dobije interpolacijom između N_1 i N_2:

$$N_b = \frac{1}{y_2 - y_1} \left(N_1(y_2 - y_s) + N_2(y_s - y_1) \right)$$

Normala točke (x_s, y_s) dobije se interpolacijom normala N_a i N_b:

$$N_s = \frac{1}{x_b - x_a} \left(N_a(x_b - x_s) + N_b(x_s - x_a) \right)$$

Budući da se normala N_s računa za svaki piksel scan-linije koji se nalazi unutar poligona, postupak se može malo modificirati da bi se dobilo na brzini. Uvede se prirast ΔN_s:

$$\Delta N_s = \frac{\Delta x}{x_b - x_a}(N_b - N_a)$$

pa se konačna normala u promatranoj točki može računati prema:

$$N_{s,n} = N_{s,n-1} + \Delta N_s.$$

Pri tome Δx označava korak po x-osi. Ako popunjavamo svaki piksel, što je uobičajeno, Δx iznosi 1. Treba uočiti da ovdje svi izrazi opisuju vektore, što znači da se svaki izraz pri računanju raspada na tri izraza (jer se računanje provodi za svaku komponentu vektora posebno). To je još jedan razlog daleko veće računske zahtjevnosti na sklopovlje koje izvodi ove kalkulacije.

9.6 Ponavljanje

1. Objasnite Phongov model osvjetljavanja. Koje su njegove komponente i kako se računaju?
2. Kako se provodi konstantno sjenčanje?
3. Kako se provodi Gouraudovo sjenčanje?
4. Kako se provodi Phongovo sjenčanje?

Poglavlje 10

Globalni modeli osvjetljavanja

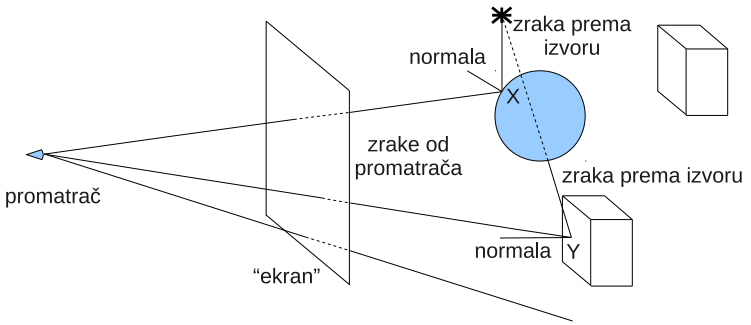
10.1 Uvod

U ovom poglavlju razmotrit ćemo globalne modele osvjetljavanja. Globalni modeli osvjetljavanja za određivanje boje svakog elementa scene u obzir uzimaju primarne svjetlosne izvore prisutne u sceni kao i utjecaj drugih objekata (sekundarnih izvora svjetlosti). Prisjetimo se, tijelo koje je osvjetljeno izvorom svjetlosti dio te svjetlosti reflektira (postajući tako sekundarni izvor svjetlosti) i time dodatno utječe na osvjetljenje drugih objekata u sceni. Phongov model osvjetljavanja ove efekte nije uzimao u obzir. Kod Phongovog modela osvjetljavanja boja pojedinih elemenata scene bila je određena samo sumom utjecaja primarnih točkastih izvora svjetlosti – tada smo govorili o lokalnom modelu osvjetljavanja. U ovom ćemo se poglavlju pozabaviti globalnim modelima. Najprije ćemo krenuti s modelom poznatim kao *bacanje zrake* (engl. *Ray Casting*), čijim ćemo proširenjem doći do prvog globalnog modela poznatog pod nazivom *praćenja zrake* (engl. *Ray Tracing*).

10.2 Algoritam bacanja zrake

Temeljna pretpostavka modela bacanja zrake jest da se svjetlost širi pravocrtno – kao svjetlosne zrake; valna priroda svjetlosti u ovom se modelu zanemaruje. Iz točkastog izvora svjetlosti kreće beskonačno mnogo svjetlosnih zraka u svim smjerovima. Neke zrake pogađaju objekte u sceni, reflektiraju se od njih, moguće ponovno pogađaju druge objekte, reflektiraju se od njih, i u konačnici, jedan mali dio tih zraka pogađa oko promatrača dok preostale zrake odlaze u beskonačnost. Dio zraka koji pogađa promatračevo oko definira sliku koju promatrač vidi.

Krenemo li direktno u implementaciju ove ideje, pojavit će se vrlo neugodan problem: većina zraka koje ćemo pratiti od izvora pa u scenu neće stići do promatrača. Umjesto toga, nakon niti jedne, jedne ili nekoliko refleksija, zrake će



Slika 10.1: Model bacanja zrake

završiti u beskonačnosti (pretpostavka je da je scena otvorena). Ovakav postupak stoga je računski vrlo skup – najveći dio zraka koje ćemo slijediti neće ni na koji način doprinositi slici koju vidi promatrač.

Vratimo se stoga na početak: pretpostavimo da se svjetlost doista širi pravocrtno, i da smo pronašli jednu zraku koja je nakon refleksije od objekta stigla do promatračevog oka. Put te zrake puno efikasnije možemo rekonstruirati obrnemo li put zrake, tako da se pretvaramo da je zraka krenula od oka promatrača – takve nas zrake uvijek zanimaju, jer one doprinose konačnoj slici koju vidi promatrač.

Ideja postupka ilustrirana je na slici 10.1. Prikazana je scena koja se sastoji od jednog izvora svjetlosti, jedne kugle te dva kvadra. Promatrač se na slici nalazi skroz lijevo. Između promatrača i objekata scene postavili smo ravninu projekcije, i u toj ravnini odabrali smo pravokutno područje koje predstavlja "ekran". Ako sliku renderiramo za prikaz na ekranu razlučivosti 1024×768, to pravokutno područje podijelit ćemo u mrežu kvadratića koji će upravo odgovarati svakom od slikovnom elementu zaslona (pikselu). Potom ćemo za svaki kvadratić izračunati zraku koja kreće iz promatračeva oka i prolazi kroz taj kvadratić. Pratit ćemo zraku kroz scenu i pratiti koji od objekata ta zraka prvoga pogađa. Dva su moguća slučaja.

Ako zraka ne probada niti jedan objekt u sceni, slikovnom elementu dodijelit ćemo intenzitet koji odgovara ambijentnoj komponenti (najdonja zraka na slici 10.1). Ako zraka probada jedan ili više objekata, postupak mora pronaći promatraču najbliže sjecište – to je ono što promatrač vidi. Primjeri su najgornja zraka na slici 10.1, gdje zraka probada i kuglu i kvadar (no probodište s kuglom označeno slovom X je bliže, pa se ono promatra) te srednja zraka koja probada samo donji kvadar (probodište je označeno slovom Y). Jednom kada je pronađeno najbliže probodište zrake i objekta, slikovnom elementu treba dodijeliti određenu boju. Prvi korak jest provjera je li sjecište možda u sjeni. Iz sjecišta se prema svakom izvoru prisutnom u sceni stvara nova zraka (engl. *shadow rays*). Za svaku zraku sjene traže se sjecišta s objektima u sceni, kako bi se utvrdilo blokira li koji objekt svjetlost tog izvora. Ako je odgovor potvrdan, odnosno postoji sjecište

zrake sjene i nekog objekta koje se nalazi između početnog sjecišta i izvora, tada izvor ne doprinosti intenzitetu sjecišta (kažemo da je točka u sjeni obzirom na promatrani izvor). Taj je slučaj prikazan za srednju zraku na slici 10.1. Ako točka nije u sjeni (primjer s gornjom zrakom na slici 10.1), računa se doprinos tog izvora i to uporabom lokalnog Phongova modela osvjetljavanja: računa se normala na objekt i temeljem nje intenzitet difuzne i zrcalne komponente. U konačnici, ako je promatrano sjecište u sjeni s obzirom na sve izvore u sceni, njegov intenzitet bit će jednak upravo ambijentnoj komponenti. U konačnici, intenzitet sjecišta bit će jednak zbroju ambijentne komponente te doprinosa difuzne i zrcalne komponente svakog izvora za koji to sjecište nije u sjeni.

Pseudokod algoritma bacanja zrake prikazan je u nastavku.

```
za svaki slikovni element (x,y)
  izračunaj zraku od oka do slikovnog elementa (x,y)
  izračunaj sjecišta zrake sa svim objektima u sceni
  pronađi sjecište (i objekt) koje je najbliže
    dodijeli boju sjecištu (klasični model osvjetljavanja)
    zapiši tu boju slikovnom elementu (x,y)
```

10.2.1 Matematički tretman algoritma

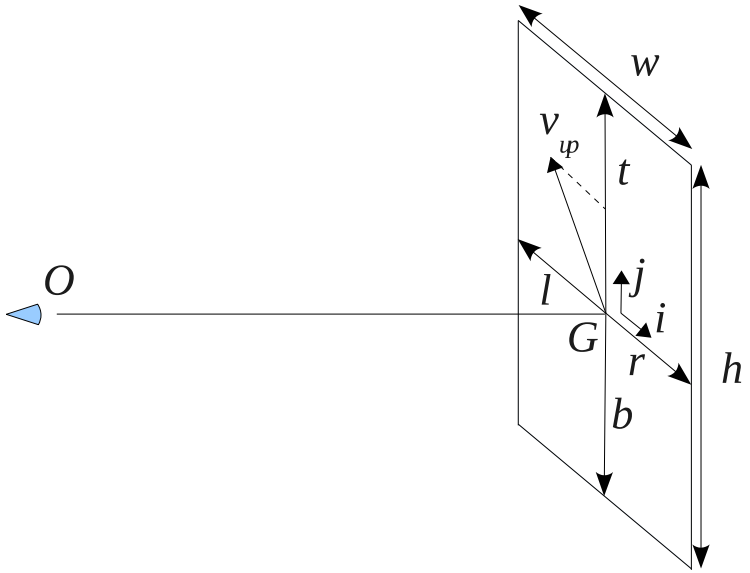
U sekciji 2.7 već smo dali matematički tretman problema koje je potrebno rješavati prilikom provođenja algoritma bacanja zrake, pa je sada pravo vrijeme za pažljivije čitanje te sekcije. Prisjetimo se samo najvažnijih detalja. Zraku ćemo uvijek prikazivati izrazom:

$$\vec{t}(\lambda) = \vec{T_S} + \lambda \cdot \vec{d}$$

pri čemu će $\vec{d}$ biti normirani vektor smjera pravca dobiven kao $\vec{d} = T_E - T_S$. Za zraku koja iz oka promatrača prolazi kroz slikovni element (x, y) točka T_S odgovarat će očištu. Točka T_E bit će točka koja leži u ravnini "ekrana" i koja odgovara slikovnom elementu na poziciji (x, y).

Ravninu u kojoj će ležati naš "ekran", te konkretno pravokutno područje unutar te ravnine moguće je zadati na više načina. Jedan je primjer prikazan na slici 10.2. Ravnina se zadaje preko očišta O, gledišta G te *view-up* vektora $\vec{v}_{up}$ koji ne leži nužno u ravnini. Točka G pri tome leži u ravnini. Projekcija *view-up* vektora $\vec{v}_{up}$ u ravninu i normiranje daje vektor $\vec{j}$ koji pokazuje pozitivan smjer y-osi. Normiranjem vektorskog produkta vektora $\vec{j}$ i $O - G$ dobiva se vektor $\vec{i}$ koji pokazuje pozitivan smjer x-osi.

Vektore $\vec{i}$ i $\vec{j}$ možemo dobiti na drugi način. Vektor $\vec{i}$ dobit ćemo normiranjem vektorskog produkta vektora $\vec{v}_{up}$ i $O - G$. Jednom kada imamo vektor $\vec{i}$, vektor $\vec{j}$ dobit ćemo normiranjem vektorskog produkta vektora $O - G$ i $\vec{i}$.



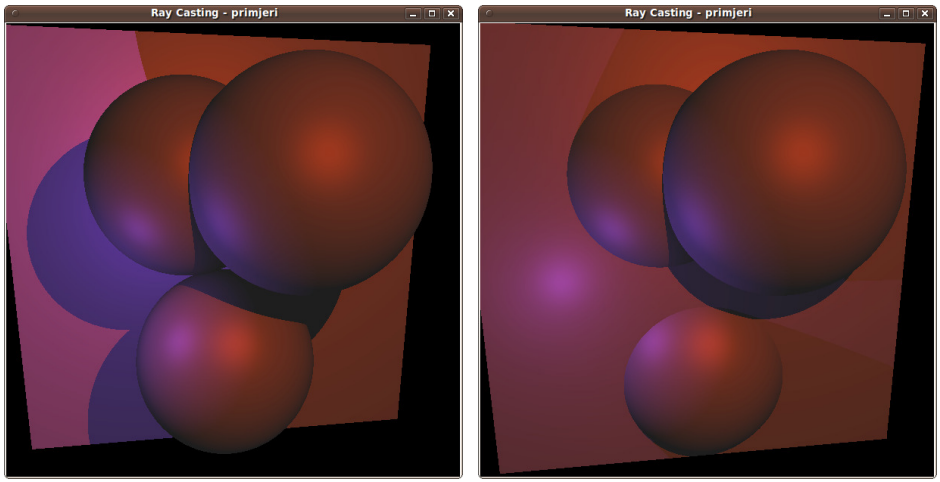
Slika 10.2: Definiranje ravnine ekrana

Jednom kada smo utvrdili vektore $\vec{i}$ i $\vec{j}$, trebamo definirati koliko je (oko gledišta) ekran širok i visok. To je moguće definirati uporabom 4 parametra:

- l - koliko se ekran proteže u lijevo od gledišta (mjereno u duljinama jediničnog vektora $\vec{i}$),
- r - koliko se ekran proteže u desno od gledišta (mjereno u duljinama jediničnog vektora $\vec{i}$),
- t - koliko se ekran proteže iznad gledišta (mjereno u duljinama jediničnog vektora $\vec{j}$) te
- b - koliko se ekran proteže ispod gledišta (mjereno u duljinama jediničnog vektora $\vec{j}$).

Uz ove podatke, još trebamo znati željenu rezoluciju ekrana, i ona je zadana parametrima w i h. Ishodište ekranskog koordinatnog sustava pri tome je smješteno dolje lijevo – točka s koordinatama (0,0) odgovara donjem lijevom uglu promatranog pravokutnog područja. Označimo sada s v točku u prostoru koja leži u ravnini i odgovara slikovnom elementu na koordinatama (x,y). Koordinate te točke su:

$$v = G + \vec{i} \cdot \left(-l + \frac{x}{w} \cdot (l + r) \right) + \vec{j} \cdot \left(-b + \frac{y}{h} \cdot (t + b) \right)$$



(a) Ravnina je ispod svih objekata (b) Ravnina prolazi kroz dio objekata

Slika 10.3: Primjer scene prikazan algoritmom bacanja zrake

Jednom kada smo izračunali točku v krećemo sa zrakom koja polazi iz v i ima vektor smjera:

$$\vec{d} = \frac{v - O}{||v - O||}.$$

Nabrojimo još i neka od svojstava algoritma bacanja zrake.

- Model spada u lokalne modele osvjetljavanja.
- Vidljivost se razriješava provjerom sjecišta zrake i objekata; nema potrebe za uporabom z-spremnika i sličnih podatkovnih struktura.
- Koristi se jednostavan Phongov model osvjetljavanja za određivanje intenziteta točke.
- Podržano je dobivanje grubih sjena.
- Računski je vrlo zahtjevan model.

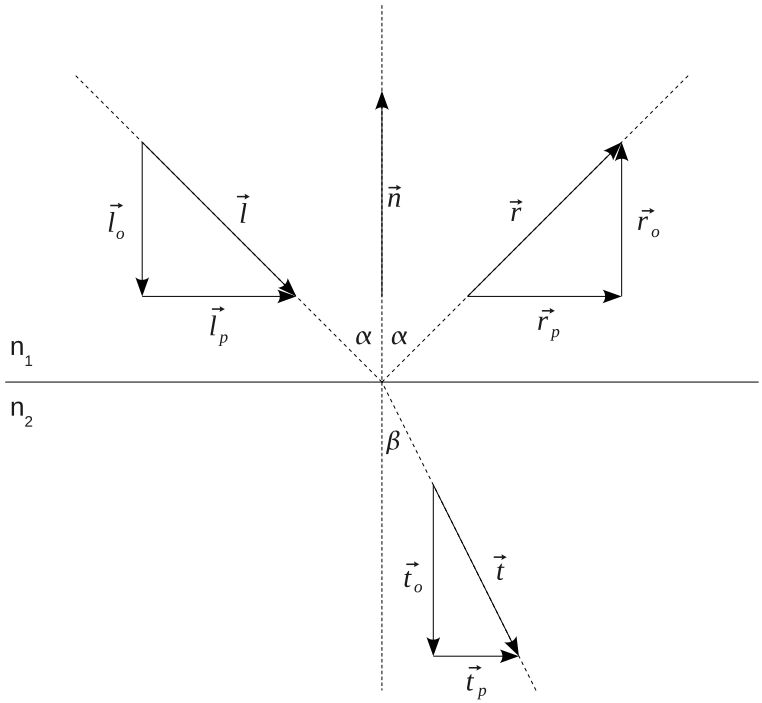
Rezultat rada ovog algoritma koji jasno prikazuje navedena svojstva prikazan je na slici 10.3.

10.3 Algoritam praćenja zrake

Algoritam praćenja zrake dodaje neke od fizikalnih zakonitosti koje smo kod algoritma bacanja zrake zanemarili, i time postaje globalni model osvjetljavanja.

Naime, kada zraka svjetlosti udari u neku površinu, ovisno o svojstvima te površine mogu nastati dvije nove zrake: reflektirana zraka te lomljena zraka. Algoritam praćenja zrake stoga će za promatrano sjecište uzeti u obzir intenzitet koji definira lokalni Phongov model te dodatno još i doprinose od reflektirane zrake i od lomljene zrake. Da bi izračunao doprinos od reflektirane zrake, algoritam započinje postupak praćenja te zrake (isto vrijedi i za doprinos lomljene zrake). Prirodan način za implementaciju ovakvog algoritma jest rekurzija. Naime, kada se krene pratiti reflektiranu zraku, ako se pronađe njezino sjecište s nekim drugim objektom, ta se zraka u tom sjecištu opet dijelom reflektira a dijelom lomi, pa je svaku od tako nastalih zraka opet potrebno pratiti kako bi se utvrdio njihov doprinos intenzitetu. Detaljniji pseudokod algoritma prikazan je u nastavku.

```
void raytrace()
  za svaki piksel ekrana (x,y)
    postaviBoju(x,y,slijedi(izračunaj_zraku_od_oka_do(x,y)))


rgbColor slijedi(zraka r)
  najmanji_t = infinity

  za svaki objekt o
    t = izračunaj_sjecište(r, o)
    najmanji_t = MIN(najmanji_t, t)

  ako je pronađeno sjecište
    return utvrdi_boju(o, r, najmanji_t)
  inače
    return pozadinska_boja


rgbColor utvrdi_boju(objekt o, zraka r, double t)
  point x = r(t)
  rgbColor color = black

  za svaki izvor svjetlosti L
    ako je najbliže_sjecište(shadow_ray(x, L)) >= udaljenost(x,L)
      color += obojaj_prema_phongu(o, x)

  color += k_specular * slijedi(reflektirana_zraka(o,r,x))
  color += k_transmit * slijedi(lomljena_zraka(o,r,x))

  return color
```



Slika 10.4: Izračun reflektirane i lomljene zrake

Izračun reflektirane i lomljene zrake pojasnit ćemo u nastavku (iako je izračun reflektiranog vektora već bio obrađen u poglavlju 2). Poslužimo se slikom 10.4.

Vektori $\vec{n}$ (vektor normale na površinu), $\vec{l}$ (vektor zrake od izvora svjetlosti od površine), $\vec{r}$ (vektor reflektirane zrake) i $\vec{t}$ (vektor lomljene zrake) pri tome predstavljaju normirane vektore, i račun koji slijedi bit će rađen uz tu pretpostavku. Za proizvoljan vektor $\vec{v}$ oznaka $\vec{v}_o$ predstavljat će komponentu vektora $\vec{v}$ koja je okomita na promatranu površinu (tj. komponentu koja je kolinearna vektoru normale na slici 10.4). Oznaka $\vec{v}_p$ predstavljat će komponentu vektora $\vec{v}$ koja je paralelna promatranoj površini na slici 10.4.

Riješimo najprije vektor reflektirane zrake. Vektor $\vec{l}_o$ je kolinearan vektoru $\vec{n}$ ali je suprotnog smjera. Norma mu odgovara kosinusu kuta α (sjetimo se da je $\vec{l}$ normiran), pa možemo pisati:

$$\vec{l}_o = -\vec{n} \cdot \cos(\alpha) = -\vec{n} \cdot ((-\vec{l}) \cdot \vec{n}) = \vec{n} \cdot (\vec{l} \cdot \vec{n}),$$

$$\vec{l}_p = \vec{l} - \vec{l}_o = \vec{l} - \vec{n} \cdot (\vec{l} \cdot \vec{n}).$$

Uočimo sada da je reflektirana zraka pod istim kutem (α) s obzirom na normalu kao i upadna zraka. Tada vrijedi:

$$\vec{r}_o = -\vec{l}_o \qquad \vec{r}_p = \vec{l}_p.$$

Slijedi:

$$\vec{r} = \vec{r_o} + \vec{r_p} = -\vec{l_o} + \vec{l_p} = -\vec{n} \cdot (\vec{l} \cdot \vec{n}) + \vec{l} - \vec{n} \cdot (\vec{l} \cdot \vec{n}) = \vec{l} - 2\vec{n} \cdot (\vec{l} \cdot \vec{n}).$$

Da bismo dobili vektor lomljene zrake, prisjetimo se fizikalnih osnova loma svjetlosti koje opisuje *Snellov zakon*. Prema njemu, ako svjetlost putuje medijem čiji je indeks loma n_1 i dolazi na granicu s drugim medijem čiji je indeks loma n_2, događa se ili totalna refleksija, ili je kut upadne i lomljene zrake određen je izrazom:

$$\frac{\sin(\alpha)}{\sin(\beta)} = \frac{n_2}{n_1}.$$

Poznavanjem upadnog kuta te indeksa lomova dolaznog i odlaznog medija n_1 i n_2 moguće je dobiti sinus kuta lomljene zrake:

$$\sin(\beta) = \sin(\alpha)\frac{n_1}{n_2}.$$

Uočimo: ako je $n_1 > n_2$, omjer $\frac{n_1}{n_2} > 1$, pa se iznos $\sin(\alpha)$ množi s brojem koji je veći od 1, čime bi se moglo dogoditi da čitav umnožak $\sin(\alpha)\frac{n_1}{n_2}$ postane veći od 1. ovo bi pak značilo da je sinus kuta β veći od jedan, što je nemoguće. Na sreću, od trenutka kada izraz $\sin(\alpha)\frac{n_1}{n_2}$ poraste do 1 (ili preko), događa se totalna refleksija: sva svjetlost koja dolazi iz izvora reflektira se, a lomljene zrake nema. Prilikom izrade algoritma praćenja zrake, ovaj uvjet treba uzeti u obzir, i lomljenu zraku slijediti samo ako ona doista postoji.

Uz pretpostavku da se lom doista događa, izvedimo sada izraz i za vektor lomljene zrake. Uočimo da vrijedi:

$$\vec{t} = \vec{t_p} + \vec{t_o}.$$

Kako je $\vec{t}$ normiran, vrijedi $||\vec{t_p}|| = \sin(\beta)$. Prisjetimo se i da vrijedi $||\vec{l_p}|| = \sin(\alpha)$. Kako su $\sin(\alpha)$ i $\sin(\beta)$ povezani Snellovim zakonom, možemo pisati:

$$\sin(\beta) = \frac{n_1}{n_2}\sin(\alpha) \quad \Rightarrow \quad ||\vec{t_p}|| = \frac{n_1}{n_2}||\vec{l_p}||.$$

No, s obzirom da su $\vec{l_p}$ i $\vec{t_p}$ kolinearni i istog smjera, tada vrijedi:

$$\vec{t_p} = \frac{n_1}{n_2}\vec{l_p} = \frac{n_1}{n_2}\left(\vec{l} - \vec{n} \cdot (\vec{l} \cdot \vec{n})\right).$$

Kako je komponenta $\vec{t}_o$ kolinearna vektoru $\vec{n}$ ali je suprotnog smjera, te s obzirom da joj je norma određena izrazom $||\vec{t}_o|| = \cos(\beta)$ vrijedi:

$$\begin{aligned}
\vec{t}_o &= -\vec{n} \cdot \cos(\beta) \\
&= -\vec{n} \cdot \sqrt{1 - \sin^2(\beta)} \\
&= -\vec{n} \cdot \sqrt{1 - \frac{n_1^2}{n_2^2} \sin^2(\alpha)} \\
&= -\vec{n} \cdot \sqrt{1 - \frac{n_1^2}{n_2^2} (1 - \cos^2(\alpha))} \\
&= -\vec{n} \cdot \sqrt{1 - \frac{n_1^2}{n_2^2} \left(1 - (\vec{l} \cdot \vec{n})^2\right)}.
\end{aligned}$$

Konačno, traženi vektor lomljene zrake tada je:

$$\begin{aligned}
\vec{t} &= \vec{t}_p + \vec{t}_o \\
&= \frac{n_1}{n_2} \left(\vec{l} - \vec{n} \cdot (\vec{l} \cdot \vec{n})\right) - \vec{n} \cdot \sqrt{1 - \frac{n_1^2}{n_2^2} \left(1 - (\vec{l} \cdot \vec{n})^2\right)} \\
&= \frac{n_1}{n_2} \vec{l} - \vec{n} \left(\frac{n_1}{n_2} \cdot (\vec{l} \cdot \vec{n}) + \sqrt{1 - \frac{n_1^2}{n_2^2} \left(1 - (\vec{l} \cdot \vec{n})^2\right)}\right)
\end{aligned}$$

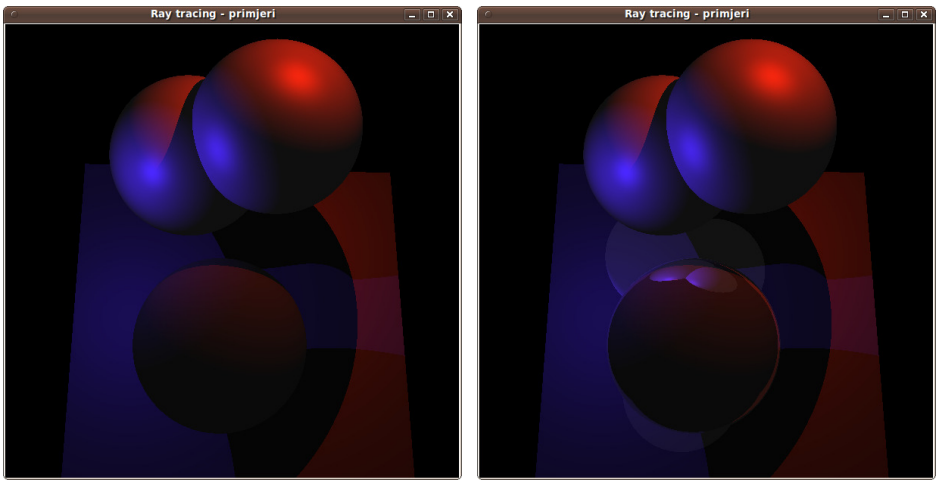
Prilikom implementacija algoritma praćenja zrake, nužno je uvesti još jedno ograničenje: maksimalnu dubinu rekurzije do koje smo spremni slijediti zrake. Ovo je posebice važno uzmemo li u obzir da se svaka zraka koja pogodi objekt razbija u dvije nove (reflektiranu i lomljenu), čime je porast broja zraka s dubinom eksponencijalan.

Rezultat rada ovog algoritma koji jasno prikazuje navedena svojstva prikazan je na slici 10.5.

10.3.1 Jednostavno preslikavanje tekstura

U ovom ćemo se poglavlju još ukratko osvrnuti na najjednostavniji način dodavanja tekstura na sferne objekte. Pretpostavimo za početak da imamo pripremljenu teksturu, kao što je to prikazano na slici 10.6.

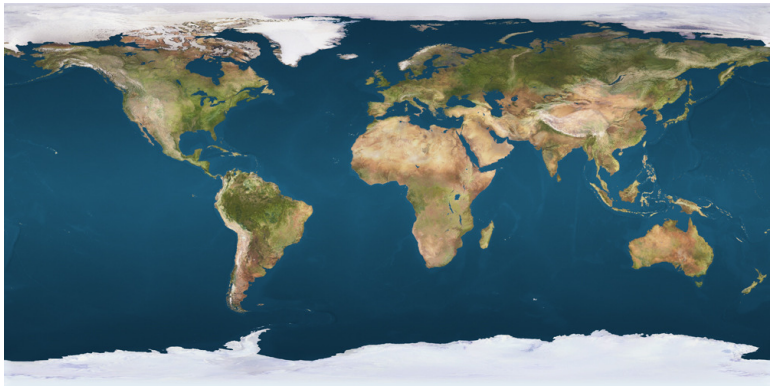
Pretpostavimo sada da smo praćenjem zrake pogodili sferu koja ima definiranu teksturu, i neka je točka probodišta T. Za točku T koja leži na površini sfere potrebno je izračunati sferne koordinate. Označimo s β kut koji zatvaraju vektori razapeti između sjevernog pola sfere i centra sfere, te između točke T i centra sfere. Ako se točka T nalazi na sjevernom polu, taj će kut biti 0; ako se točka nalazi na ekvatoru kut će biti 90 a ako se točka nalazi na južnom polu, kut će biti 180 stupnjeva. Uočimo, dakle, da je raspon kuta β od 0 do 180 stupnjeva.



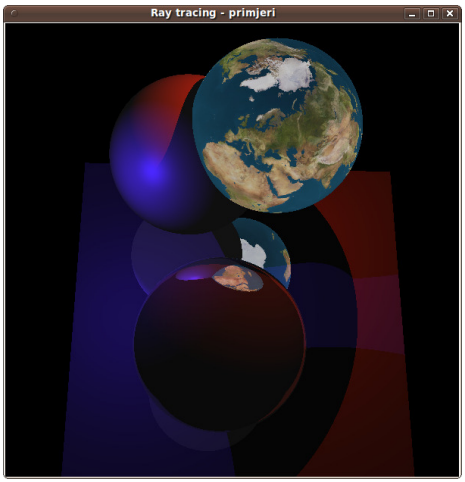
(a) Rekurzivno praćenje je zabranjeno

(b) Rekurzivno praćenje dopušteno je do dubine 6

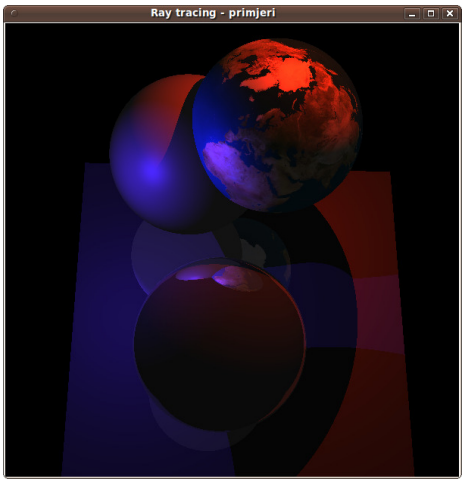
Slika 10.5: Primjer scene prikazan algoritmom praćenja zrake



Slika 10.6: Primjer teksture za sferu



(a) Intenzitet teksture se preslikava u probodište



(b) Intenzitet teksture se u probodištu modulira utjecajem doprinosa primarnih i sekundarnih izvora

Slika 10.7: Primjer praćenja zrake uz preslikavanje teksture na sferni objekt

Označimo sada s α kut koji zatvara vektor razapet između projekcije točke T u ravninu u kojoj leži ekvator i centar sfere te vektor koji iz centra sfere pokazuje u smjeru nultog kuta (u analogiji sa Zemljom, to bi bio vektor koji iz središta Zemlje probada ekvator točno kroz meridijan koji prolazi kroz Greenwich). Ovaj kut treba izmjeriti pazeći na smjer, tako da se sud o kutu ne može donijeti samo temeljem skalarnog produkta tih vektora. Ovaj kut kreće se od 0 do 360 stupnjeva. U analogiji sa Zemljom, kut β predstavlja latitudu a kut α longitudu.

Jednom kada imamo kuteve α i β, potrebno je utvrditi koju to točku predstavlja u teksturi. No to je trivijalno. Neka je tekstura dimenzije $w \times h$. Treba pogledati piksel na koordinatama (x,y) i preuzeti odgovarajuće RBG komponente kao intenzitet probodišta. Koordinate (x,y) određene su izrazima:

$$x = \frac{\alpha}{360} * w \qquad y = \frac{\beta}{180} * h.$$

Kutevi α i β mogu se odrediti poznavanjem vektora normale $\vec{n}$ u točki probodišta prema izrazima:

$$\alpha = \arctan(n_y, n_x) \qquad \beta = \arccos(n_z).$$

Funkcija *arctan* koju ovdje koristimo je funkcija koja prima dva argumenta, i temeljem toga može odrediti korektan kut od 0 do 360 stupnjeva. Primjer slike nastao ovakvom primjenom teksture prikazan je na slici 10.7a.

Konačno, kako bi se dobio utjecaj okolnog osvjetljenja, umjesto da se točki probodišta dodijeli intenzitet preuzet iz teksture možemo posegnuti za malo kompleksnijim načinom. Najprije za točku na klasičan način izračunamo pripadni intenzitet (označimo ga s I_1). Potom iz teksture dohvatimo intenzitet pripadne točke (označimo ga s I_2). Pretpostavimo također da su I_1 i I_2 zapravo trokomponentni (imaju r, g i b komponente čiji je iznos od 0 do 1). Konačni intenzitet izračunat ćemo kao $I_1 \cdot I_2$ i to ćemo uzeti kao intenzitet probodišta. Primjer slike nastao ovakvom modulacijom intenziteta teksture prikazan je na slici 10.7b.

10.4 Ponavljanje

1. Opišite kako se provodi algoritam bacanja zrake.
2. Opišite kako se provodi algoritam praćenja zrake.

Poglavlje 11

Boje

11.1 Uvod

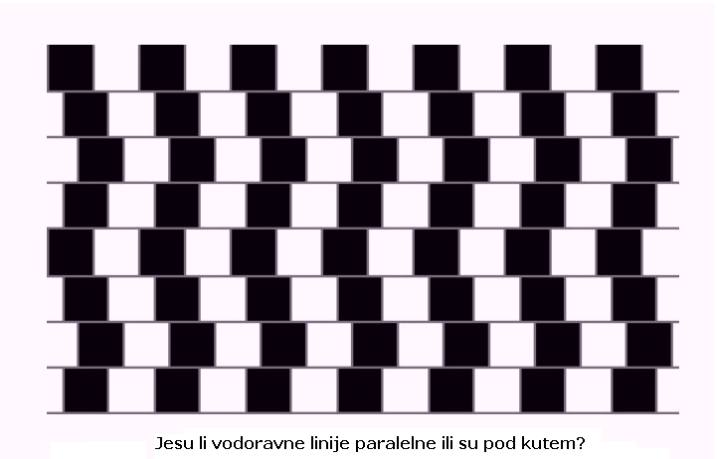
Što su boje? Kako čovjek vidi boje? Kako možemo izmjeriti "boje"? Ovo su samo neka od pitanja na koje ćemo pokušati dati odgovor kroz ovaj tekst. Boje su prvenstveno vezane uz pojam svjetlosti. Čovjek vidi predmete oko sebe zahvaljujući razvijenim senzorima za elektromagnetske valove u području valnih duljina koje nazivamo vidljiva svjetlost. Ovo područje obuhvaća valne duljine od 390 do 760 nm, odnosno frekvencijski spektar između 394.7 i 769.2 THz. Kod čovjeka postoje dvije vrste senzora koji su se specijalizirali za dvije različite namjene, vjerojatno kao prilagodba na čovjekovu okolinu. Naime, čovjek živi, kada o svjetlosti govorimo, u dva različita okruženja: danju i noću.

Noć je karakteristična po tome što svjetlosti ima vrlo malo, odnosno vrlo je slabog intenziteta. U takvom okruženju bitno je raspoznavati osnovne elemente okoline. Bitno je razaznavati različite intenzitete svjetlosti, i na temelju toga raspoznavati objekte koji ga okružuju. U ovakvoj okolini nema dovoljno informacija, a niti potrebe, za raspoznavanjem boje. I upravo za ovakvo okruženje razvila se je prva vrsta osjetila – *štapići*. To su osjetila koja raspoznaju različite intenzitete svjetlosti, i mogu "raditi" već pri vrlo malim količinama svjetlosti.

Za razliku od noći, tijekom dana raspoloživa količina svjetlosti je vrlo velika. Toliko velika, i na toliko različitih valnih duljina, da se na temelju te količine svjetlosti može dobiti puno više informacije od običnog intenziteta – možemo raspoznavati boju. Za ovaj režim rada razvijen je drugi tip osjetila – *čunjići*. Pomoću tih osjetila čovjek je sposoban primati informaciju o boji predmeta iz okoline. U čovjekovom oku postoje tri tipa ovih osjetila: osjetilo za crvenu boju, osjetilo za zelenu boju te osjetilo za plavu boju. Naravno, odmah se postavlja pitanje kako onda vidimo npr. žutu boju, kada za nju nemamo osjetila. Zanim-

ljivo je da neke životinjske vrste imaju više od tri tipa čunjića, pa tako na primjer jedna vrsta račića ima 16 tipova osjetilnih stanica namijenjenih za razlikovanje boja.

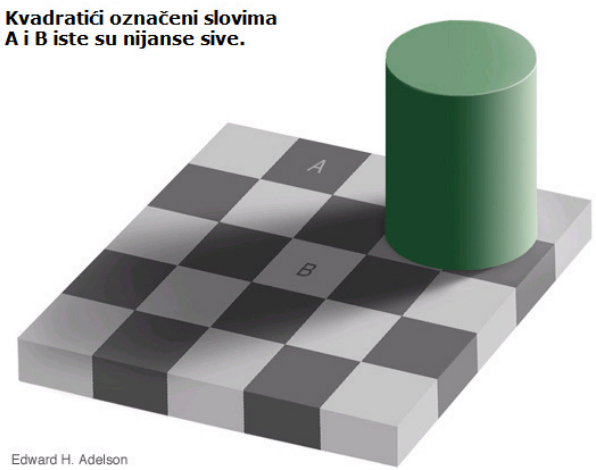
Kada čovjek promatra svijet oko sebe, pored same fiziologije treba uzeti u obzir i psihofizički doživljaj. Fiziološki doživljaj uvjetovan je svjetlošću koja dolazi do oka prolazi kroz leću u oku te stvara sliku na stražnjoj strani oka, mrežnici, dijelu koji je osjetljiv na svjetlost. S druge strane, kako će naš mozak interpretirati sliku koju vidi ovisi jako i o prijašnjem iskustvu i doživljaju. Neke stvari u koje smo uvjereni ispada da i nisu sasvim onakve kakve nam izgledaju. Možemo pogledati primjer na slici 11.1. Vodoravne linije nam izgledaju zakrivljene. No, ako uzmemo papir za koji znamo da nije zakrivljen ili računalom generiran kvadrat ili prozor i poravnamo s pojedinom vodoravnom linijom vrlo brzo ćemo se uvjeriti da sve vodoravne linije jesu ravne a ne zakrivljene. Prostorna percepcija objekata koja nam je uobičajena, u ovom je primjeru zloupotrijebljena prikazom niza crnih kvadrata koji su posloženi tako da nam prividno iskrive prostor. Postoji niz primjera sličan prethodnom koji nam pokazuju kako nas naš sustav vida zapravo vara.



Slika 11.1: Paralelne linije?

Vezano uz doživljaj boje također znamo da boju ne doživljavamo kao na primjer zvuk, u apsolutnom smislu, tako da možemo na jedinstven način identificirati pojedini ton. Doživljaj boje i svjetline ovisi ne samo o izvoru svjetlosti i površini na koju svjetlost pada već i o prilagođenosti našeg oka te o okolnom sadržaju kojeg promatramo na slici na kojoj se nalazi. Kada sa sunčane ulice uđemo u mračni prostor bit će potrebno neko vrijeme da se oko prilagodi novo nastalom okruženju i da ponovo progledamo u zamračenom prostoru. Na Slici 11.2 možemo pogledati kvadrate označene slovima A i B. Siva nijansa oba kvadrata je ista. U ovo se možemo uvjeriti tako da izrežemo ili približimo kvadrate

A i B jedan drugom. To znači da na istoj slici ako je okruženje kvadrata svjetlije tu sivu nijansu doživljavamo tamnijom a ako je okruženje tamnije kako kod kvadrata B, taj kvadrat nam uvjerljivo izgleda svjetlijim. Eto, sada smo svjesni da ono što vidimo i ono što mislimo da vidimo nije isto čak ni na običnoj statičnoj slici. S tim je vezana upotreba boja općenito, pa tako i u računalnoj grafici.



Slika 11.2: Kvadratići iste nijanse sive.

11.2 Sheme za prikaz boja – prostori boja

Kako bi sustavno mogli baratati bojama važno je kvantificirati pojedine boje odnosno bojama pridijeliti numeričke vrijednosti. Posebno je kod nekih proizvoda važno da budu ujednačeni i da na tržište uvijek izlaze u jedinstvenoj boji jer je naše oko posebno osjetljivo već i na male promjene u nijansi kada promatramo dva uzorka nijanse jedan do drugoga. Dva proizvoda, odnosno nijanse boje u koje se proizvodi oboje iz različitih serija, mogle bi odudarati ako nisu identične. Na primjer, tapete ili dijelovi karoserije automobila, mogli bi odudarati kada se stave jedan uz drugi ako boja nije u potpunosti ista.

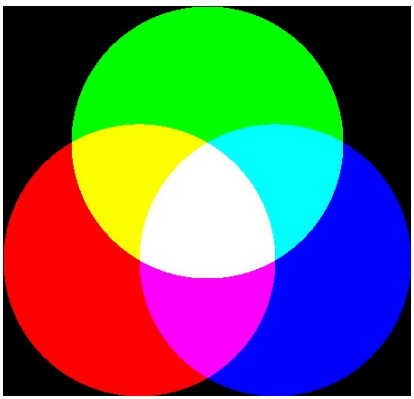
11.2.1 Prostori boja

Tijekom proučavanja svjetlosti i problematike vezane uz boje razvijeno je nekoliko modela kojima se pokušava dobiti kontrola nad bojama. Prvi i osnovni model proizašao je iz otkrića kako čovjek vidi boje. Tako je nastao *RGB - prostor boja*. Slova u nazivu ovog prostora boja su početna slova od engleskih naziva boja: *Red* (crvena), *Green* (zelena) te *Blue* (plava). Teorijskim razmatranjima razvijen je *XYZ - prostor boja*. Iz tehničkih razloga razvijen je i *CMY - prostor*

boja (Cyan, Magenta, Yellow), a potom i *CMYK - prostor boja* (Cyan, Magenta, Yellow, blacK). Radi lakoće odabiranja boja nastao je *HSV - prostor boja* (Hue, Saturation, Value). Razvijeni su još i modeli razreda *Y* (engl. *class Y models*) kao rezultat razvitka TV i video tehnike, i drugi.

11.2.2 RGB - prostor boja

RGB - prostor boja proizašao je iz pokušaja da se imitira način na koji čovjek vidi boje. Naime, čovjek ima osjetila za tri osnovne (primarne) boje: crvenu, zelenu i plavu. Uzevši tu činjenicu u obzir, pretpostavilo se da se sve boje mogu prikazati kao linearna kombinacija ovih triju osnovnih boja. Na ovaj način, da bismo pamtili boju jedne točke, trebamo pamtiti tri broja: koliko imamo crvene, koliko imamo zelene, te koliko imamo plave. Ako sve tri komponente iznose nula, dobit ćemo crnu. Ako sve tri komponente iznose maksimum, dobit ćemo bijelu. Ako su pojedine komponente prisutne s različitim udjelima, dobit ćemo različite boje. Nijanse sivih boja dobivat ćemo ako sve tri osnovne boje držimo jednake po iznosu, i taj iznos variramo. Fizikalno ovaj proces možemo zamisliti na sljedeći način: imamo na raspolaganju tri svjetlosna izvora: izvore crvene, zelene i plave. Sva tri izvora usmjerena su u istu točku, i ne postoji niti jedan drugi izvor svjetlosti. Ako su sva tri izvora ugašena, točka je crna jer nema nikakve svjetlosti koja bi je obasjala. Počnemo li zatim paliti pojedine izvore, točka će poprimati različite boje jer će doći do miješanja boja. Zbog ovog svojstva da se pojedine komponente svjetlosti zbrajaju, ovo miješanje zovemo *aditivno miješanje*. Proces je prikazan na slici 11.3.



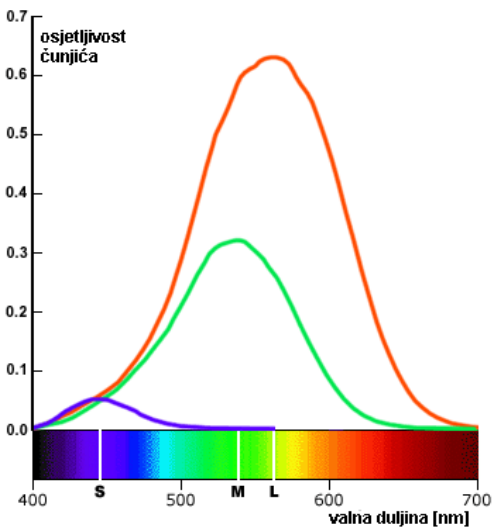
Slika 11.3: Aditivno miješanje boja

Opisani model vrlo je jednostavan. Međutim, ima jednu manu. Sve boje ne mogu se opisati pomoću ovih triju primarnih boja, barem ne ako imamo u vidu

fizikalnu sliku miješanja boja. Naime, u pokusu podudaranja boja popokazalo se da s monokromatskim komponentama boje RGB nije moguće ostvariti sve boje koje ljudsko oko opaža.

11.2.3 Pokus podudaranja boja

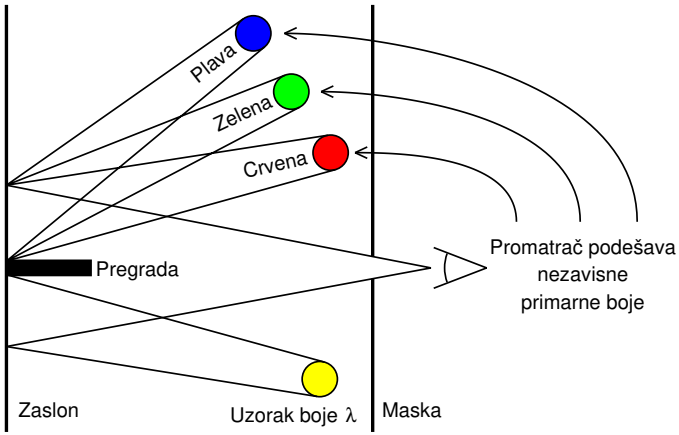
U ljudskom oku postoje tri vrste čunjića temeljem čega je razvijena tropodražajna teorija boja. Po ovoj teoriji sve boje je moguće dobiti kombinacijom tri primarne komponente. Za svaku vrstu čunjića (SML - za male, srednje i velike valne duljine) rađen je graf osjetljivosti na pojedinu valnu duljinu te je ustanovljeno da maksimumi osjetljivosti za pojedinu vrstu čunjića odgovaraju valnim duljinama u kojima mi vidimo crvenu, zelenu i plavu boju. Brojnost pojedine vrste čunjića također se razlikuje pa je temeljem brojnosti dobiven težinski prikaz osjetljivosti čunjića na pojedinu valnu duljinu, prikazan na 11.4. Znači, kada mi promatramo monokromatsku svjetlost određene valne duljine u našem oku je potaknuto više vrsta čunjića tom svjetlošću ali s različitim odazivom na podražaj. S druge strane, htjeli bi model u kojem imamo tri primarne komponente kojima želimo ostvariti jedinstveni zapis za sve ostale boje koje vidimo.



Slika 11.4: Težinski prikaz osjetljivosti tri vrste čunjića SML u ljudskom oku

U pokusu podudaranja boje, većem broju promatrača postavljen je ispitni uzorak boje zadane valne duljine s lijeve strane zaslona. Promatrač je trebao podesiti vrijednosti tri komponente monokromatske valne duljine koja odgovara crvenoj, zelenoj i plavoj (RGB) s desne strane tako da se postavljena ispitna boja podudari s bojom koju je podesio promatrač. Komponente crvene, zelene i plave boje za podešavanje odgovarale su osjetilnim maksimumima dobivenim

na čunjićima ljudskog oka. Znači, ispitna boja s lijeve strane pregrade trebala je odgovarati podešenoj boji s desne strane pregrade, Slika 11.5. No, za neke ispitne uzorke boje to se pokazalo neostvarivim. Primijećeno je da ako se u tom slučaju na stani ispitnog uzorka boje doda još crvena komponenta, traženo podudaranje je bilo moguće ostvariti. Crvena se zbog toga što je dodana sa strane ispitnog uzorka tretirala kao oduzimanje crvene komponente, odnosno dodavanje negativne vrijednosti crvene boje. Rezultat ovog pokusa za pojedine valne duljine prikazan je na Slici 11.6. Na slici je vidljivo da je komponenta crvene boje u području oko 500 nm negativna. Iz ovog pokusa možemo zaključiti da tropodražajnim načinom s tri primarne komponente RGB ne možemo ostvariti sve boje koje vidimo.



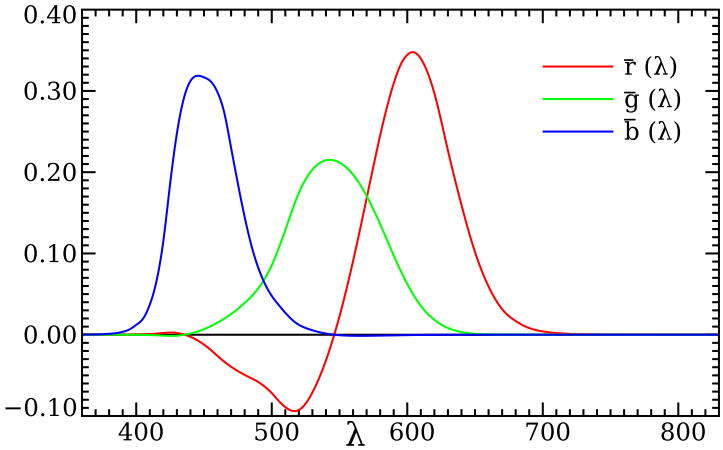
Slika 11.5: Pokus podudaranja ispitnog uzorka boje

Ovaj nedostatak rezultirao je sastankom CIÉ (Commission Internationale de l'Éclairage) gdje se je pokušao definirati model koji bi također imao samo tri primarne boje X, Y i Z, i koji bi opisivao sve boje uz pozitivne koeficijente. Rezultat je CIÉ XYZ sustav boja.

11.2.4 Sustav boja CIÉ XYZ i dijagram kromatičnosti

Primarne boje u sustavu boja CIÉ XYZ su hipotetske boje koje ne postoje i nisu vidljive, ali se njihovom aditivnom kombinacijom mogu dobiti sve vidjive boje. Ovaj prostor boja možemo prikazati kao tri prostorne osi XYZ koje određuju ravninu i prostor boja u toj ravnini (Slika 11.7). U prostoru XYZ razapeta je ravnina određena uvjetom $X + Y + Z = 1$ u kojoj možemo prikazati dijagram kromatičnosti.

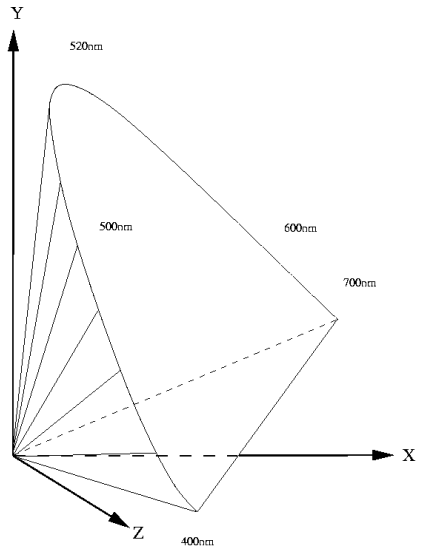
CIÉ je također definirala dijagram kromatičnosti u sustavu xy koji opisuje sve boje koje čovjek može vidjeti. Dijagram je prikazan na slici 11.8. Svjetlina



Slika 11.6: Komponente RGB potrebne za prikaz svih boja

(engl. *brightness*) se u ovom kontekstu promatra odvojeno od boja. Svjetlina nam određuje da je na primjer bijela boja jače svjetline od sive a u dijagramu kromatičnosti promatramo samo maksimalnu svjetlinu. Za dijagram kromatičnosti možemo primijetiti da je potkovastog oblika. Na krivulji ruba potkove nalaze se spektralne boje ili spektrani lokus, a dužina koja spaja lijevi i desni rub potkove zadrži aspektralne odnosno tako zvane purpurne boje. Ako u dijagramu kromatičnosti odaberemo dvije točke, boje koje se nalaze na dužini koja spaja te dvije točke možemo dobiti kao linearnu kombinaciju te dvije boje. U sredini dijagrama je bijela boja koja odgovara Sunčevoj svjetlosti a prema rubu potkove se nalaze spektralne boje koje su potpuno zasićene pa na ovaj način od sredine prema rubu imamo i promjenu zasićenosti pojedine spektralne boje. Komponenta zasićenosti (engl. *saturation*) pojedine boje mijenja se udjelom bijele komponente, dok se potpuno zasićena boja nalazi na rubu potkove.

Na slici 11.8 je na dijagramu kromatičnosti prikazana i krivulja temperature boje (engl. *color temperature*). Ova krivulja predstavlja temperaturu isijavanja crnog tijela izraženog u Kelvinima. Idealno crno tijelo predstavlja tijelo od ugljika (carbon) koje samo isijava dok ništa ne reflektira. Zagrijavanjem ovo tijelo isijava svjetlost boje od crvene preko užareno narančasto-žute do bijele pa sve do plave i ljubičaste. Pojedina točka krivulje ima vrijednost temperature boje izraženu u Kelvinima. Ova vrijednost temperature boje često se koristi za izražavanje temperature boje bijele svjetlosti koju možemo ostvariti na monitoru ili na primjer rasvjetnim tijelom. Pa ako je to temperatura znatno veća od 6500 K boja postaje plavkasta ako je znatno manja boja prelazi u žuto-narančastu.



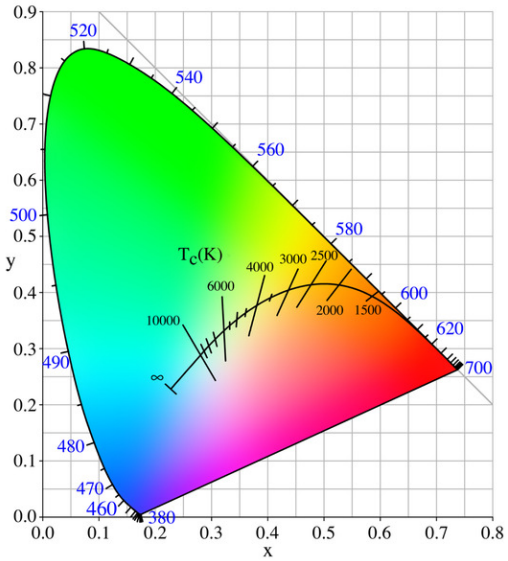
Slika 11.7: Komponente XYZ potrebne za prikaz svih boja

Nedostatak dijagrama kromatičnosti je u tome što su pojedina područja izrazito nelinearna u ljudskoj percepciji. To znači da se male promjene u području plave boje koje primjećujemo nalaze blizu na dijagramu kromatičnosti, dok veća područja u području zelene boje uopće nisu različita našem oku (Slika 11.9).

Kako linearna kombinacija dviju boja određuje spojnicu tih komponenta u dijagramu kromatičnosti, jasno je i da ako imamo tri komponente, kao na primjer RGB komponente kod monitora, moći ćemo aditivnom kombinacijom dobiti samo točke unutar tako razapetog trokuta. Taj trokut, odnosno u općem slučaju poligon, naziva se *gamut* uređaja. Uz specifikaciju uređaja za prikaz kao što su monitori, televizori, pisači ili foto oprema često dolazi i prikaz gamuta uređaja na dijagramu kromatičnosti (slika 11.10). Vrhovi poligona su određeni primarnim komponentama boje kojima dotični uređaj raspolaže.

Boje koje se nalaze izvan trokuta (poligona) gamuta, ne mogu se prikazati kombinacijom tih triju boja. Ako pokušamo umjesto tri izvora uporabiti četiri – ništa se ne mijenja; lik sada neće biti trokut, već četverokut, i površina će biti nešto veća te ćemo bolje pokriti dijagram kromatičnosti. No opet će biti boja koje ne možemo prikazati.

Sve što smo do sada promatrali odnosilo se na svjetlost i miješanje valnih duljina koje dolaze do našeg oka. Kada promatramo obojani zid svjetlost dolazi iz izvora do obojane površine, dijelom se apsorbira a dijelom reflektira od podloge. Za bijelu podlogu znamo da se ne grije na Suncu jer reflektira primljeno zračenje, dok crno obojena tijela apsorbiraju energiju i griju se. Ako na primjer Sunčeva svjetlost dolazi do lista neke biljke, taj list apsirbira sve valne duljine osim zelene



Slika 11.8: Dijagram kromatičnosti

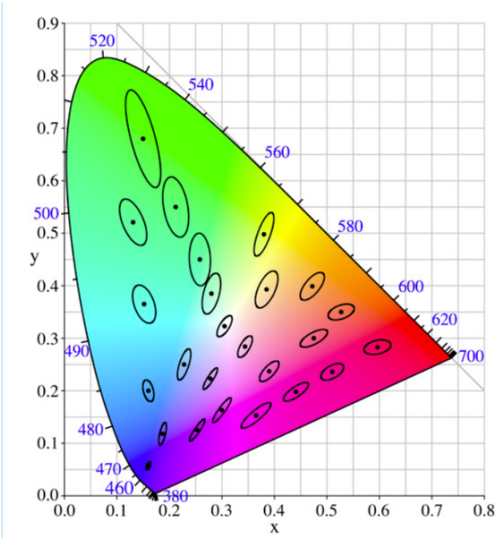
koju biljke reflektiraju kao višak. Na ovaj način dolazimo do *subtraktivnog* modela boja koji se primjenjuje kod miješanja pigmenata boje kao što je na primjer kod tiska.

11.2.5 CMY/CMYK-model

Model CMY slijedi upravo obratnu filozofiju od RGB-modela. Pretpostavka je da bez ikakvih utjecaja na točku boja točke mora biti bijela, kakav je obično papir. Boje ćemo dobiti tako da od bijele oduzimamo pojedine komponente u različitim iznosima, a kada sve komponetne oduzmemo preostat će nam crna. Osnovne boje koje se ovdje koriste su plavozelena *Cyan* (cijan), ljubičasta *Magenta* i žuta *Yellow*. To su komplementarne boje RGB-modelu i možemo primijetiti da se nalaze na presjecima skupova na Slici 11.3. Štoviše, ukoliko normiramo pojedine komponente RGB-modela (dakle, svake boje ima u iznosu od 0 do maksimalno 1), tada vrijedi relacija:

$$C = 1 - R$$
$$M = 1 - G$$
$$Y = 1 - B$$

Slika 11.11 prikazuje miješanje ovih osnovnih boja. Opet možemo primijetiti da se RGB komponente nalaze na presjecima skupova CMY. Žuta je na primjer komplementarna plavoj, nalazi se na dijagramu točno nasuprot i žuta reflektira



Slika 11.9: Neuniformna područja u dijagramu kromatičnosti.

ono što se nalazi u skupu žute, a to je zelena i crvena dok apsorbira palvu. Smjesu zelene i crvene svjetlosti mi vidimo kao žutu. Pa na ovaj način možemo doći do bilo koje kombinacije. To znači i da ako izvor svjetlosti daje na primjer ljubičastu svjetlost (znači sadrži crvenu i plavu komponentu) te njime osvjetlimo žutu podlogu, takva podloga će apsorbirati plavu komponentu i mi ćemo zapravo vidjeti da je podloga crvene boje jer nam je od dolazne ljubičaste svjetlosti jedino ta komponenta reflektirana od žute podloge.

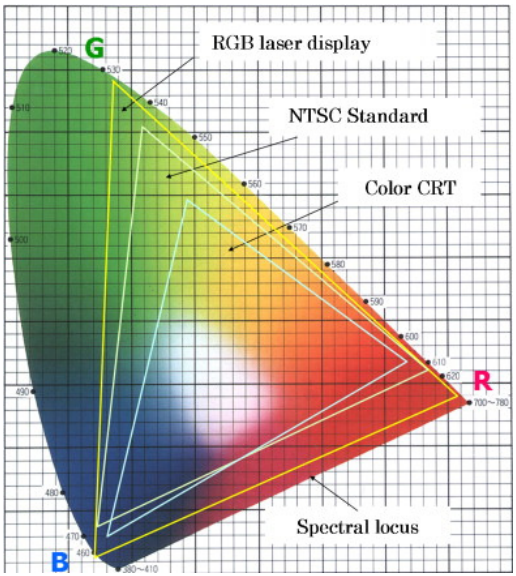
Uvjet da je točka sama po sebi bijela ukoliko na nju ne djeluje ništa nalazimo upravo kod bijelog papira – zbog toga se ovaj model koristi upravo kod printera. Zapravo, kod printera se koristi CMYK model, koji je ekonomiziran CMY model. Naime, teorijske pretpostavke i praksa obično se baš i ne slažu najbolje, pa tako maksimalnim miješanjem boja CMY dobivamo crnu koja baš i nije onako crna kako bismo to željeli. S druge strane, većina dokumenata koja se danas još uvijek ispisuje obilato koristi upravo crnu boju, što za ovaj model znači maksimalnu potrošnju svih boja – a uopće ne ispisujemo u boji. Zbog tih razloga, u model je uvedena još i crna boja, te je model dobio u oznaci slovo K od riječi blacK. Tako se uvode sljedeće relacije:

$$K = \min(C, M, Y)$$

$$C = C - K$$

$$M = M - K$$

$$Y = Y - K$$



Slika 11.10: GAMUT uređaja

11.2.6 HSL-prostor boja

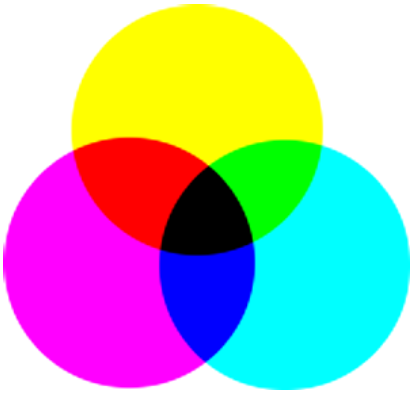
HSL model (Hue, Saturation, Lightnes) nastao je iz potrebe da se olakša odabir boje. Naime, prethodno opisani modeli vrlo su nezgrapni kada je riječ o odabiru boje. Zamislite da ste našli kombinaciju koja vam daje neku nijansu narančaste boje, i sada želite dobiti samo malo zasićeniju boju, ili pak malo svjetliju. Što učiniti da bi se to postiglo? Problem je u tome što treba mijenjati sve tri komponente RGB-modela, a to komplicira jednostavnu uporu. Ideja je da promjenom vrijednosti H obilazimo sve boje. Kada pronađemo odgovarajuću boju, sa S njezinu zasićenost te s L odredimo željenu svjetlinu. Pri tome komponente H, S i L čine stožac, valjak ili čun. Primjer je prikazan na slici 11.12. Vrlo slični prostori boja su HSV (Hue, Saturation, Value), HSB (Hue, Saturation, Brightness), HSI (Hue, Saturation, Intensity) i neki drugi.

H je predstavljen kutom u odnosu na pozitivnu x-os, i njime se biraju boje. Vrijedi sljedeće:

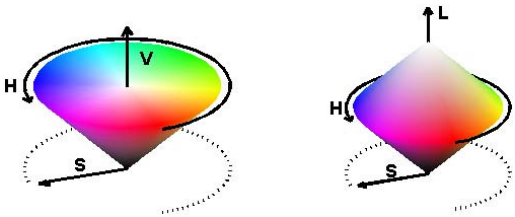
Ako zasićenost stavimo na nulu ($S = 0$), tada promjenom svjetline (L) od 0 do 1 dobivamo sve nijanse sive boje, uključujući crnu ($L = 0$), i bijelu ($L = 1$).

11.3 Odabir intenziteta kod sjenčanja objekata

Zamislimo da se nalazimo pred sljedećim problemom: imamo na raspolaganju 256 lokacija na kojima možemo pamtiti intenzitet boje i želimo sjenčati objekt. Intenzitet je u rasponu od 0 do 1 (0 je minimalni, 1 maksimalni). Potrebno je



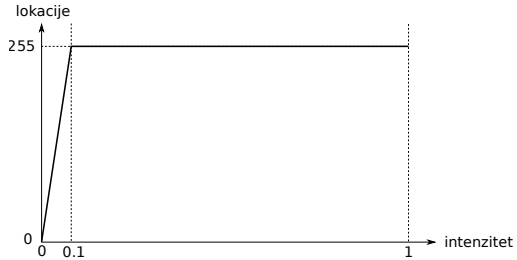
Slika 11.11: CMY-model boja



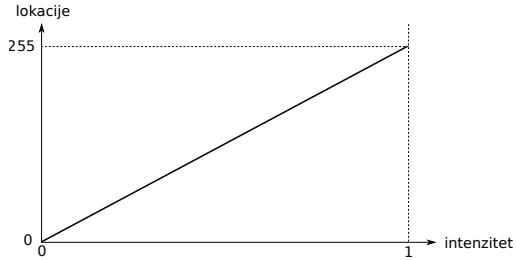
Slika 11.12: HSV i HSL prostori boja

H vrijednost	Odgovarajuća boja
0°	Plava
60°	Magenta
120°	Crvena
180°	Žuta
240°	Zelena
300°	Cijan

Tablica 11.1: Odabir boja kod HLS-modela



Slika 11.13: Linearna raspodjela intenziteta



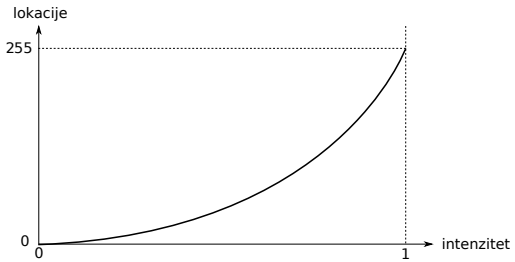
Slika 11.14: Linearna raspodjela intenziteta (2)

odrediti koliki intenzitet ćemo pohraniti u pojedinu lokaciju, a da pri tome dobijemo takvu raspodjelu koja će biti prihvatljiva ljudskom oku. Npr. intenzitete možemo raspodijeliti prema slici 11.13.

Lokacija 0 sadržavat će intenzitet 0, lokacija 255 intenzitet 0.1, dok će intenziteti između 0 i 0.1 biti linearno pridjeljeni lokacijama 1 do 254. Međutim, ovakva raspodjela nije dobra jer ćemo cijelu sliku nacrtanu pomoću ovakvih intenziteta jedva vidjeti – naime, intenziteti su raspodijeljeni kao da je na snazi opća opasnost i metode zamračivanja. To nije dobro.

Možda je bolja raspodjela prikazana na slici 11.14. Tu je intenzitet također linearno raspodijeljen, i to tako da se u pojedinim lokacijama nalaze i najmanji, a u pojedinim i najveći intenziteti kao na slici 11.14. To je, dakako, bolje. Međutim... Ljudsko oko intenzitete ne raspoznaje baš na ovaj način. Naime, ljudsko oko nije osjetljivo na apsolutni iznos intenziteta već na omjer susjednih vrijednosti intenziteta. Odnosno težnja je da omjer susjednih vrijednosti intenziteta bude konstantan, pa pretpostavimo neki početni minimalni intenzitet I_0 i konstantan omjer susjednih intenziteta r. Na ovaj način dobit ćemo nelinearnu karakteristiku raspodjele intenziteta.

$$I_0 = I_0$$
$$I_1 = r \cdot I_0$$
$$I_2 = r^2 \cdot I_0$$



Slika 11.15: Nelinearna raspodjela intenziteta

...

$$I_n = r^n \cdot I_0 \tag{11.1}$$

Ujedno znamo da je I_n upravo jednak 1, pa možemo izračunati faktor r:

$$I_n = r^n \cdot I_0 = 1 \Rightarrow r = \left(\frac{1}{I_0}\right)^{\frac{1}{n}} \tag{11.2}$$

U općem slučaju se, dakle, može napisati formula:

$$I_j = r^j \cdot I_0 = \left(\left(\frac{1}{I_0}\right)^{\frac{1}{n}}\right)^j \cdot I_0 = (I_0)^{1-\frac{j}{n}} \tag{11.3}$$

Kako u našem slučaju imamo 256 razina, odnosno razine od 0 do 255, vrijedi:

$$r = \left(\frac{1}{I_0}\right)^{\frac{1}{255}}, \qquad I_j = (I_0)^{1-\frac{j}{255}}, \qquad j \in \{0, 1, ..., 255\}$$

Ovakva raspodjela intenziteta prikazana je na slici 11.15.

No, razmotrimo još jednom o čemu se ovdje radi. S jedne strane za zapis u računalu koristimo niz brojeva koje nazivamo intenziteti. Ti brojevi vezani su uz slikovne elemente koji onda na zaslonu određuju osvjetljenje pojedinog slikovnog elementa. Razlikujemo raspon sivih nijansi i RGB komponente. Ovu razliku ćemo na tren zanemariti i uzet ćemo samo sive razine. Svjetlost koju daje pojedini slikovni element možemo izmjeriti.

Dva su osnovna načina kako možemo promatrati i izraziti osvijetljene elemente. Jedan način je korištenjem radiometrijskih jedinica drugi način je korištenjem fotometrijskih jedinica. Ako intenzitet razmatramo u radiometrijskim jedinicama dobit ćemo jedinice $[Wsr^{-1}]$ (engl. *watts per steradian*). Steradijan predstavlja prostorni kut. Ovaj intenzitet se još razmatra po jedinici projicirane površine (engl. radijance). U radiometrijskom pristupu promatra se ukupno elektromagnetsko zračenje.

No, kao što smo vidjeli, samo dio elektromagnetskog spektra je vidljiv ljudskom oku. Tako da je ovdje potrebno uzeti u obzir osjetljivost ljudskog oka na pojedine valne duljine. Intenzitet promatran u sklopu vidljivog dijela spektra izražava se u fotometrijskim jedinicama. Intenzitet u fotometrijskim jedinicama mjeri se u kandelama ($cd = lm/sr$) odnosno lumenima po steradijanu. Za slikovne elemente u boji koriste se krivulje koje je definirala CIÉ za standardnog promatrača, kako bi dobili pripadni intenzitet. I ovaj intenzitet se promatra po jedinici projicirane površine (engl. *luminance*) i izražava se u jedninicama ($[cd \cdot m^{-2}]$). Ako ovu vrijednost označimo s Y, za monitor moramo znati koju maksimalnu vrijednost možemo ostvariti pa ako je to $Y_n = 100cd \cdot m^{-2}$ tada ćemo naš raspon, odnosno $Y = 1$, prilagoditi toj referentnoj vrijenosti.

Sada još treba uzeti u obzir da na ove fotometrijski izražene vrijednosti intenziteta po jedinici površine (luminance) ljudsko oko ima nelinearnu percepcijsku ovisnost, odnosno ovisi s trećim korjenom:

$$L^* = 116(Y/Y_n)^{1/3} - 16 \tag{11.4}$$

gdje je Y_n referentno bijelo svjetlo.

Odaziv ljudskog oka na svjetlinu pokazuje da ako se dva susjedna intenziteta ne razlikuju za više od jedan posto, nećemo ih razlikovati. Drugačije rečeno, osjetljivost ljudskog oka je otprilike logaritamska. Upravo to smo uzeli u obzir pri određivanju intenziteta u formulama 11.1 - 11.3.

11.4 Gama korekcija

Kako smo do sada vidjeli, svjetlina koju daje slikovni element nije linearna funkcija vrijednosti primjenjenog signala da bi taj slikovni element osvijetlili. Za različite izlazne naprave ova funkcija se razlikuje. Pogledajmo sada situaciju sa klasičnim CRT monitorom. Osvjetljenje neke točke na zaslonu rezultat je pogađanja te točke (odnosno fosfora u toj točci) zrakom elektrona. Intenzitet tako dobivene svjetlosti to je veći, što više elektrona N pogađa tu točku u jedinici vremena. Matematički se ovisnost intenziteta može opisati relacijom:

$$I = k \cdot N^{\gamma} \tag{11.5}$$

gdje je I intenzitet koji emitira fosfor, a k i γ konstante. Broj elektrona proporcionalan je naponu koji uzrokuje pojavu, te vrijedi:

$$I = k \cdot (k_1 \cdot V)^{\gamma} = k \cdot k_1^{\gamma} \cdot V^{\gamma} = K \cdot V^{\gamma} \tag{11.6}$$

gdje je I intenzitet koji emitira fosfor a k, k_1, K i γ konstante. Konstanta γ se naziva gama. Ako je poznat intenzitet koji želimo dobiti, napon koji je potreban za njegov prikaz dobit ćemo izjednačavanjem relacija (11.1) i (11.6):

$$K \cdot V_j^{\gamma} = r^j \cdot I_0 \Rightarrow I_j = \left(r^j \cdot \frac{I_0}{K} \right)^{\frac{1}{\gamma}} \tag{11.7}$$

Treba uočiti da je γ karakteristika samog monitora, pa će zbog toga na različitim monitorima odgovarajući naponi biti različiti – ili pak isti, ali tada će prikazane slike biti različite, a to ipak ne želimo. Kod CRT je gama otprilike 2.5, dok je kod LCD ta vrijenost manja. Kako je karakteristika ljudskog oka upravo obrnuta ovaj učinak će se dijelom ispraviti no nije dobro da nemamo kontrolu nad time. Odnosno, problem je ako je nešto korigirano a ne znmo s kojom vrijednošću, pa nakon toga radimo prepravke i ispravke samo ćemo pogoršati situaciju.

Neki standardi za zapis slika i videa (PAL, NTSC) vode računa o tome da li je slika gama korigirana i s kojom vrijednošću, pa se temeljem te informacije i informacije o tome na kojem uređaju prikazujemo sliku radi ukupna korekcija, odnosno definira prijenosna funkcija kojim se preslikavaju ulazne vrijenosti intenzitata.

Pored toga ambijent u kojem promatramo zaslon ima također utjecaja. Ako je prostorija potpuno zamračena ili jako rasvijetljena drugačije ćemo doživjeti istu sliku na zaslonu. Obično će biti potrebna korekcija kontrasta da bi uočili pojedine detalje na slici. Kontrast na uređaju određuje omjer između bijele i crne boje ostvarene uređajem pa taj omjer obično možemo podešavati na uređaju.

11.5 Pamćenje boja na računalu

Rad s bojama na računalima izveden je tako da se može prilagoditi potrebama i memorijskim zahtjevima, odnosno raspoloživim memorijskim resursima samog računala. Osnovni način prezentacije boje na računalu je putem RGB-sustava. Dakle, za svaku boju pamte se tri komponente: crvena, zelena i plava. Međutim, tu postoje dva moda rada:

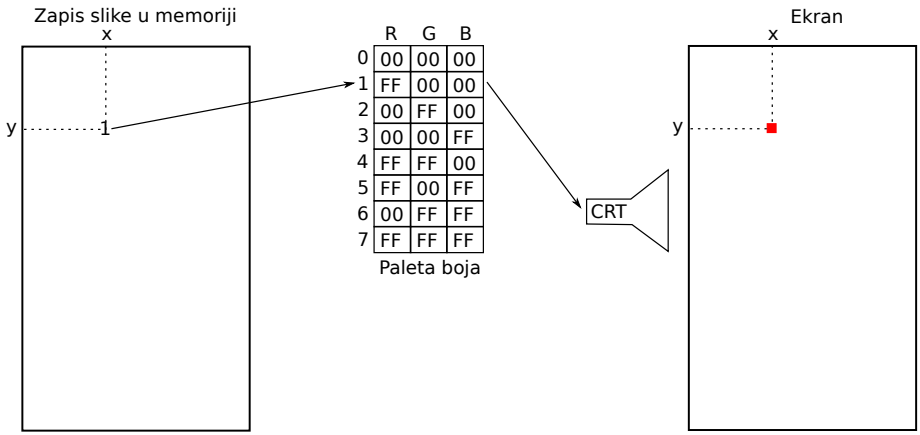
- uporaba paleta boja, te
- direktna reprezentacija boje.

Prva metoda razvijena je zbog uštede memorije, i time naravno nameće određena ograničenja. Glavni faktor koji određuje koliko ćemo različitih boja moći prikazati naziva se dubina. Ovisnost dubine boja i broja mogućih različitih boja dana je relacijom (11.8):

$$n = 2^d \tag{11.8}$$

gdje je n broj raspoloživih boja a d dubina.

Dubina se mjeri u broju bitova. Ideja je da se formira tablica koja ima onoliko elemenata koliko mi to odredimo preko dubine. Svaki taj element sadržavat će tri komponente: količinu crvene, količinu zelene te količinu plave boje. Boja pojedinog piksela (npr. boja j) određivat će se brojem od 0 do $n-1$, i odgovarat će onoj boji čija se definicija nalazi u odgovarajućem retku tablice (dakle, j-ti redak). Slika 11.16 prikazuje kako to radi.



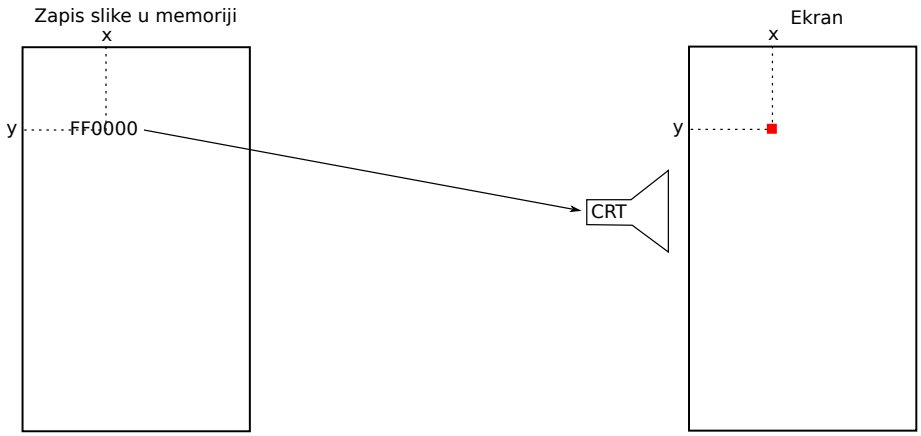
Slika 11.16: Pamćenje boja u paleti boja

Paleta boja na slici 11.16 definira boje prikazane u tablici 11.2.

Indeks u paleti boja	Boja
0	crna
1	crvena
2	zelena
3	plava
4	žuta
5	magenta
6	cijan
7	bijela

Tablica 11.2: Primjer palete boja

U navedenom primjeru, na poziciji *(x,y)* nalazi se boja 1. Prije iscrtavanja na zaslonu, gleda se u paletu boja i pronalazi pod brojem 1 definicija crvene boje. Na zaslonu se crta crvena točka. U ovom primjeru tablica je imala 8 elemenata jer je zadana dubina iznosila 3 bita. Ukoliko odaberemo dubinu od 8 bitova, imat ćemo na raspolaganju 256 mogućih boja. Ova metoda dobra je kada nemamo puno memorije na raspolaganju, a niti ne želimo prikazivati *true-color* slike.



Slika 11.17: Direktna pohrana boja

Povećavamo li dubinu do veličine od 24 bita, paleta boja nam se više ne isplati jer paletom boja možemo definirati jednaki iznos boja kao i da direktno definiramo boju. Naime, uz 24 bita po jednom pikselu, bitove možemo grupirati u grupe po 8: 8 bitova za crvenu, 8 bitova za zelenu i 8 bitova za plavu. U tom slučaju više ne koristimo palete, već se za svaki piksel direktno definira boja. Mana ove metode je što zahtjeva veliku količinu memorije za pohranu iole većih slika. Metoda je prikazana na slici 11.17. Sada na lokaciji *(x,y)* piše kompletan zapis boje: `FF 00 00` što odgovara crvenoj boji.

11.6 Ponavljanje

1. Pojasnite razliku između aditivnih i suptraktivnih modela boja.
2. Opišite RGB model boja.
3. Opišite CIÉ XYZ model boja.
4. Opišite CMY/CMYK model boja.
5. Opišite HSL model boja.
6. Što je Gamma korekcija?
7. Kako se pamte boje na računalu?

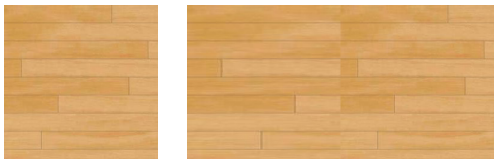
Poglavlje 12

Postupci preslikavanja tekstura

12.1 Uvod

Promatrajući predmete koji nas okružuju vrlo brzo ćemo zaključiti da su značajno raskošniji od jednostavnih glatkih kugli i objekata koje smo do sada razmatrali. Teksture su jedan od elemenata koji značajno vizualno obogaćuju prikazane trodimenzionalne scene. Njihovo preslikavanje na poligone u današnje vrijeme sklopovski je podržano te je iscrtavanje objekata s teksturama vrlo brzo. Što su teksture? 2D teksture možemo zamisliti kao tapete koje lijepimo na objekt. Na primjer, ako promatramo pod na kojem je parket, vrlo komplicirano bi bilo definirati svaku daščicu parketa posebnim poligonom i svaki uzorak pojedine daščice dodatnim vrhovima, poligonima i bojama. Umjesto toga, načinimo sliku parketa koja predstavlja osnovni uzorak koji onda koristimo za popunjavanje većih površina (vidi sliku 12.1a). Pristup izrade tekstura temeljenih na uzorku ima svoje prednosti i nedostatke. Prednost je u tome što nalazimo osnovni uzorak koji onda višestruko koristimo i na taj način štedimo memoriju teksture. Ograničenje je u tome što uzorak mora biti takav da se lijepo nastavlja kod popunjavanja većih površina – to je takozvani ponavljajući uzorak (engl. *tileable*); kada to nije dobro pogođeno, dobivaju se artefakti poput onog prikazanog na slici 12.1b gdje se jasno vidi spoj. Nedostatak je u tome što pravilnost u uzorku daje neželjene artefakte u rezultatu, odnosno ponavljanje uzorka je vidljivo pri promatranju s udaljenije točke. Savršeno ponavljanje osnovnog uzorka ne djeluje prirodno jer u stvarnosti postoje anomalije i nepravilnosti u uzorku, u stvarnosti plohe nisu u potpunosti čiste pojedina mjesta mogu biti zaprljana, zbog čega nemamo savršeno ponavljanje osnovnog uzorka što odaje umjetno napravljena okruženja.

U području računalnog vida razmatraju se različite definicije vezane uz pojam teksture, no u računalnoj grafici bilo koja 2D slika se smatra teksturom. Nije nužno da imamo osnovni uzorak teksture koji će se ponavljati, već svaki djelić površine može imati svoju jedinstvenu boju. Na ovaj način dolazimo do toga



(a) Osnovni uzorak.

(b) Popločavanje osnovnim uzorkom.

Slika 12.1: Primjer teksture. Uzorak teksture (lijevo), tekstura koja nije dobra kao uzorak jer se lijeva/desna i gornja/donja strana ne nastavljaju pri popunjavanju većih površina. Ovaj uzorak nije ponavljajući.

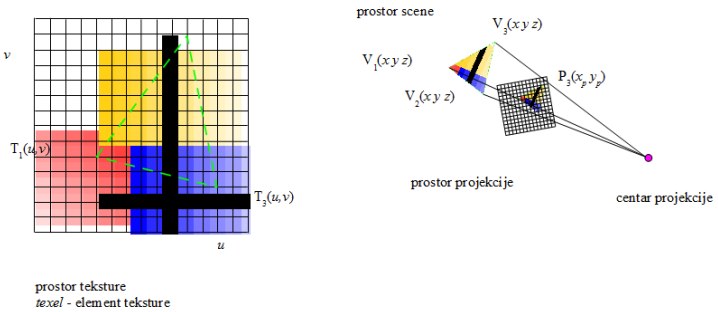
Slika 12.2: Primjer objekta sastavljenog od krpica površine, osjenčanog objekta i objekta s pripadnom teksturom.

da je boja vezana uz površinu objekta zapravo tekstura tog elementa površine (slika 12.2). U prikazanom primjeru svaka točka objekta imat će jedinstvenu boju pridruženu pojedinom slikovnom elementu. Računski gledano, u izrazu za izračun osvjetljenja, komponenta boje (tekstura površine) se dodaje kao svojstvo elementa površine i određuje na koji način će taj element površine reflektirati svjetlo. U Phongovom modelu osvjetljenja (empirijski model) to su jednostavno RGB komponente boje iz 2D slike, a mogu biti i različite za ambijentnu, difuznu i zrcalnu komponentu kojima se množe komponente dobivene izrazom za osvjetljenje.

U postupku preslikavanja teksture prvo ćemo razmotriti preslikavanje teksture na jedan poligon, a potom ćemo razmotriti postupke preslikavanja na niz poligona koji čine objekt.

12.2 Preslikavanje teksture na poligon

U razmatranju preslikavanja teksture krenut ćemo od jednog poligona, odnosno trokuta. Prvo trebamo razlučiti tri osnovna prostora: *prostor scene* koji je 3D

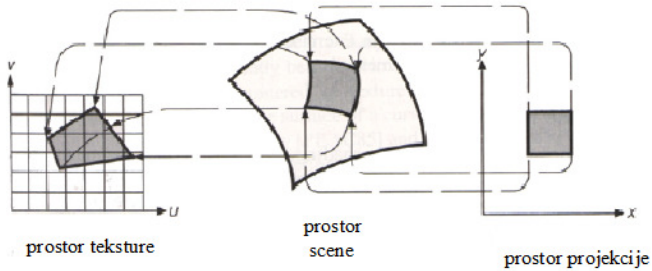


Slika 12.3: Preslikavanje teksture na trokut.

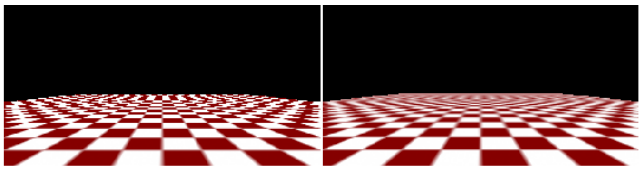
prostor te *prostor projekcije* i *prostor teksture* koji su 2D prostori (slika 12.3). Uzet ćemo primjer trokuta koji se nalazi u sceni, a njegova projekcija je onda naravno u prostoru projekcije. U prostoru teksture nalazi se naša 2D slika koju želimo preslikati na trokut. U prikazanom primjeru to je slika koju čini 14×14 slikovnih elemenata. U prostoru teksture osnovni elementi se zovu *elementi teksture* odnosno *texeli* (engl *texture element*). Ovaj prostor se često naziva i uv-prostor (ili ponekad st-prostor), gdje u određuje x a v određuje y os. Pripadne koordinatne osi kojima je ovaj prostor razapet nazivaju se u i v osi. Na slici 12.3 prikazan je primjer trokuta u 3D prostoru scene koji ima koordinate vrhova $V_1(x, y, z)$, $V_2(x, y, z)$ i $V_3(x, y, z)$ te prostor projekcije u kojem je zbog preglednosti istaknuta slika samo jednog vrha označena s $P_3(x_p, y_p)$. Točka $P_3(x_p, y_p)$ određena je koordinatama u 2D prostoru projekcije. Za svaki vrh trokuta iz prostora scene potrebno je definirati njene koordinate u prostoru teksture. Za sada zamislimo da smo to definirali unaprijed ručno; kako se to može raditi automatski, i to za sve poligone tijela, razmotrit ćemo kasnije. U prikazanom primjeru, koordinate vrhova teksture su $T_1(u, v)$, $T_2(u, v)$, $T_3(u, v)$. To su 2D koordinate u prostoru teksture i svaka je pridružena svom vrhu u prostoru scene.

Važno je istaknuti da se tekstura ne preslikava na 3D poligone već se poligon najprije preslikava u prostor projekcije pa se na tako dobiveni poligon preslikava tekstura. U postupku projekcije za svaku točku trokuta $V_1(x, y, z)$, $V_2(x, y, z)$ i $V_3(x, y, z)$ određuju se pripadne projicirane koordinate $P_1(x_p, y_p)$, $P_2(x_p, y_p)$ i $P_3(x_p, y_p)$ u prostoru projekcije. U postupku rasterizacije trokuta $P_1(x_p, y_p)$, $P_2(x_p, y_p)$ i $P_3(x_p, y_p)$ određujemo koji slikovni elementi čine taj trokut. Tada, za pojedini slikovni element dobiven rasterizacijom, određujemo koje područje teksture prekriva u prostoru teksture taj jedan slikovni element. Cilj je temeljem područja teksture koje određuje jedan slikovni element odrediti pripadnu boju. To je prikazano na slici 12.4.

Na slici 12.4 prikazan je istaknuti slikovni element u prostoru projekcije (desno), pripadno područje u prostoru scene koje se preslikalo na taj slikovni



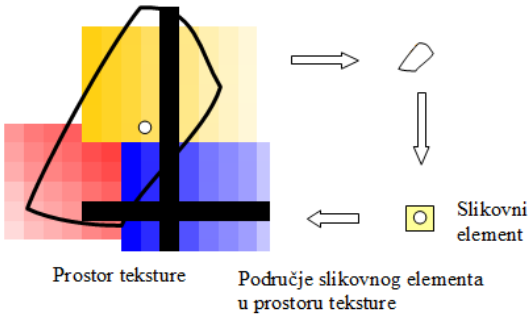
Slika 12.4: Određivanje vrijednosti slikovnog elementa u prostoru projekcije.



Slika 12.5: Problem neželjenog učinka aliasa pri preslikavanju teksture.

element (sredina) te pripadno područje koje pokriva taj slikovni element u prostoru teksture (lijevo). Područje u prostoru teksture može prekrivati veći broj elemenata teksture ili može biti vrlo malo područje i može upasti negdje između slikovnih elemenata u području teksture. Time dolazimo do dva ekstremna slučaja. Jedan je kada puno elemenata teksture određuje jedan element u projekciji. U ovom slučaju obično se računa prosječna vrijednost elemenata obuhvaćenih elemenata teksture za jedan slikovni element površine. Drugi ekstrem je kada se slikovni element iz projekcije preslikava u vrlo malo područje između elemenata teksture (ako se elementi teksture tretiraju kao točke). Tada možemo primjerice temeljem bilinearne interpolacije odrediti vrijednost boje slikovnog elementa u prostoru projekcije.

Za isti poligon koji se nalazi u prostoru scene ovisno o tome koliko je taj poligon blizu očišta možemo dobiti oba ekstremna slučaja. Čak i za jedan poligon koji je istovremeno blizu i daleko od promatrača možemo dobiti istovremeno oba ekstremna slučaja (slika 12.5). Ovisno o načinu na koji se određuje vrijednost elementa teksture dobit ćemo više ili manje izražen neželjeni učinak nazubljenosti u slici s teksturom. Na slici 12.5 (lijevo) možemo vidjeti izraženu nazubljenost, odnosno alias artefakte, dok desno na slici iako slika djeluje mutnija, prihvatljivija je našem sustavu vida. Problem se očituje ne samo u statičkoj slici već se ispoljava i temporalno, odnosno prilikom animacije pogleda oko prikazanih poligona kada titranje daje vrlo neugodan vizualni učinak.



Slika 12.6: Problem neželjenog učinka aliasa pri preslikavanju teksture.

Određivanje vrijednosti slikovnog elementa, u idealnom slučaju, temeljilo bi se na određivanju integrala vrijednosti boje površine koja je obuhvaćena površinom u prostoru teksture koja je pridružena promatranom slikovnom elementu, kao što je prikazano na slici 12.6. No ovaj postupak bio bi problematičan za izvesti uz zadovoljavajuću brzinu na računalu. Zato se rade određene aproksimacije kao bi se postigla zadovoljavajuća kvaliteta uz prihvatljivu brzinu izračuna.

Najjednostavnije rješenja određivanja pripadne boje površine je interpolacija najbližeg susjeda (engl. *nearest neighbor*). To znači da za promatrani slikovni element uzimamo samo središte slikovnog elementa iz prostora teksture te pridružujemo tom slikovnom elementu vrijednost koju smo "pogodili" u prostoru teksture. U prikazanom primjeru na slici 12.6 to je označeno bijelim kružićem. Ako bi to na primjer bio žuti slikovni element rezultat će biti žuti, a ako bi to bilo malo pomaknuto udesno, susjedni element je crni, pa bi rezultat bio crni. Time se javlja velika osjetljivost na promatrani uzorak u postupku, što rezultira neželjenim učinkom aliasa. U slučaju računanja integrala nad žutim i crnim slikovnim elementom, rezultat bi bio sivo-žuti neovisno o pomaku. Uz interpolaciju najbližeg susjeda desit će se titranje prilikom animacije, odnosno pomicanja objekata u vremenu.

Kako bi poboljšali ovu osnovnu ideju, možemo povećati broj uzoraka i umjesto jednog središnjeg uzorka možemo uzeti, na primjer, četiri uzorka u vrhovima slikovnog elementa ili devet, šesnaest, ... slikovnih elemenata za svaki promatrani slikovni element u prostoru projekcije. U tom slučaju vrijednost konačnog promatranog elementa računali bi kao prosječnu vrijednost promatranih slikovnih elemenata, što zapravo odgovara postupku integracije. Ovaj postupak se naziva i povećano uzorkovanje (engl. *super sampling*). U konačnici cilj bi bio uzeti toliko uzoraka da područje u prostoru teksture bude pokriveno dovoljnim brojem uzoraka i da ne "promašimo" uzorcima detalje koji su u prostoru teksture.

Broj uzoraka koji bi trebalo uzeti određen je *Nyqist-Shannonovim* teoremom o uzorkovanju proširenim na 2D slučaj i ovisi o maksimalnoj frekvenciji prisutnoj u signalu. Ovdje je signal u 2D prostoru, odnosno slika. Kod 2D slika visoke frekvencije su prisutne zbog vrlo čestih naglih prijelaza boja u slici.

No kako praktično odrediti koliko povećati broj uzoraka a da ne utrošimo previše vremena i da dobijemo poboljšani rezultat? Ovaj kompromis određivanja broja uzoraka i postizanja brzine izvođenja ostvaruje se takozvanim *Mip-Map* postupkom preslikavanja tekstura.

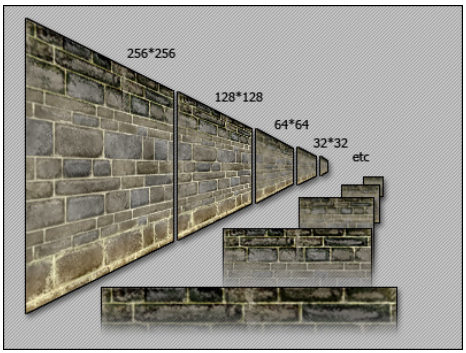
12.3 Postupak Mip-Map preslikavanja tekstura

Kako smo vidjeli, cilj u postupku preslikavanja tekstura je odrediti integral nad vrijednostima slikovnih elemenata u prostoru teksture te na taj način umanjiti neželjene alias učinke. Postupak bi mogli ubrzati ako pripremimo unaprijed integrirane vrijednosti slikovnih elemenata po pojedinim područjima. U osnovi, to se radi u *Mip-Mapama*. *Mip* je kratica latinskog izraza *multum in parvo*, što znači puno u malom. Grafičko sklopovlje podržava funkcionalnost izrade Mip-Mapa; stoga je stvaranje Mip-Mapa vrlo brzo.

Znači, uzima se osnovna slika koja je, na primjer, 256×256 slikovnih elementa. Od te slike napravi se niz slika koje su po svakoj stranici upola smanjene: 128×128, 64×64, 32×32, 16×16, 8×8, 4×4, 2×2, 1×1, tako da po četiri susjedna slikovna elementa prethodne razine određuju (prosječna vrijednost) jedan slikovni element niže razine. Umanjenja koja smo ostvarili izraženo faktorima smanjenja su $4:1$, $16:1$, $64:1$, ... Znači, osnovnu sliku rezolucije 256×256 smo umanjili četiri puta da bi dobili sliku 128×128, pa je njezin faktor smanjenja označen kao $4:1$.

Na ovaj način unaprijed imamo pripremljene prosječne vrijednosti slikovnih elemenata na pojedinoj skali. Najmanji element 1×1 sadržavat će prosječnu vrijednost elementa cijele slike; stoga je ovo i općenito vrlo brz način da to odredimo korištenjem grafičkog sklopovlja. Primjer napravljenih Mip-Mapa prikazan je na slici 12.7a. Pojedina razina naziva se i *LOD* (engl. *level of detail*). Na slici 12.7b možemo vidjeti što se dešava prilikom preslikavanja teksture na zid prikazan u perspektivi. Zid se sastoji od pet dijelova tj. poligona na koje se obično preslikava ista tekstura (pridjeljuju se iste koordinate u prostoru teksture). Na poligon koji je najbliži nama preslikat ćemo Mip-Mapu razine 0. Poligon koji je peti u nastavku prikazanog zida u prostoru projekcije obuhvaća svega nekoliko slikovnih elemenata i bilo bi rastrošno veliku teksturu preslikavati na svega nekoliko slikovnih elemenata jer se ionako detalji ne vide. Zato je cilj odrediti Mip-Mapu niže rezolucije koja će biti preslikana na taj poligon.

(a) Primjer napravljenih MipMapa različitih razina.



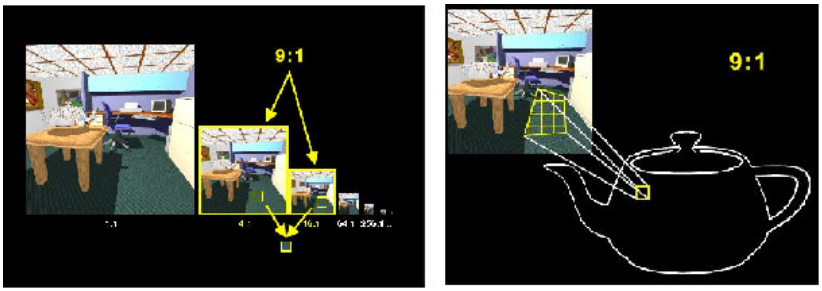
(b) Preslikavanje teksture na zid.

Slika 12.7: Različite Mip-Mape se pridružuju pojedinom poligonu, ovisno o tome koliko je velika projekcija tog poligona u prostoru projekcije.

Slika 12.8: Efikasno pohranjivanje Mip-Map tekstura.

Izračunate slike Mip-Mape efikasno možemo pohraniti u memoriji korištenjem organizacije RGB komponenti kao što je prikazano na slici 12.8. Jednu četvrtinu osnovnog područja koristimo rekurzivno za pohranu RGB komponenti na nižoj razini.

Nakon određivanja Mip-Mapa za pojedini slikovni element potrebno je procijeniti područje koje u postupku preslikavanja pokriva promatrani slikovni element u prostoru teksture. To znači procijeniti površinu poligona koji razapinje slika slikovnog elementa u prostoru teksture. Neka je to, na primjer, devet slikovnih elemenata kao što je prikazano na slici 12.9. Tada određujemo najbližu Mip-Mapu (po faktoru smanjenja) koja je unaprijed određena i iz nje dohvaćamo određen broj slikovnih elemenata. U prikazanom primjeru po omjeru najbliža Mip-Mapa je 16 : 1. Za interpolaciju najbližim susjedom uzimat ćemo jedan slikovni element, a za bilinearnu interpolaciju uzimat ćemo četiri slikovna elementa (iz mape 16 : 1) temeljem kojih određujemo konačan slikovni element. Možemo uzeti i dvije susjedne Mip-Mape, u našem primjeru to su 4 : 1 i 16 : 1 jer je pro-



Slika 12.9: Postupak preslikavanja teksture Mip-Mapama.

matrani faktor smanjenja $9:1$. Iz dviju susjednih Mip-Mapa uzimamo po četiri najbliže vrijednosti slikovnih elemenata a zatim temeljem trilinearne interpolacije određujemo konačan slikovni element. Ovdje smo u razmatranje uzeli samo jednostavne načine interpolacije; interpolaciju najbližeg susjeda i (bi/tri)linearnu interpolaciju, no u složenijim postupcima interpolacije veće područje slikovnih elemenata odnosno napravljenih Mip-Mapa možemo uzeti u proračun. U tom slučaju okoliš od na primjer 3×3 slikovna elementa može biti uzet u razmatranja kako bi dobili viši stupanj interpolacije u proračunu. Osnovne ideje interpolacije prikazane u poglavlju Bézierove krivulje primjenjive su na postupak interpolacije koji se ovdje koristi, no trilinearna interpolacija daje vrlo prihvatljiv rezultat obzirom na trošak dohvaćanja i izračuna.

U postupku određivanja elemenata teksture treba imati na umu da pri izračunu promatranog slikovnog elementa paralelno s izračunavanjem boje treba odrediti i udaljenost (z) tog slikovnog elementa od promatrača. Kako vrijednosti temeljem kojih se računa promatrani slikovni element mogu imati različite z-koordinate, utjecaj se odražava i na konačni promatran slikovni element u prostoru projekcije. I ovdje se mogu desiti artefakti ovisno o načinu na koji će se to obavljati. U ovom slučaju artefakti će biti prisutni u izračunatoj udaljenosti od promatrača i manje su izraženi nego u određivanju boje, no svejedno se očituju. Na grafičkim karticama prisutne su različite opcije koje omogućuju određivanje broja z-vrijednosti temeljem kojih će se izračun raditi što daje kompromis između bolje kvalitete i brzine izračuna. Primjer je *MSAA* (engl. *Multi Sampling AA*) gdje se uzimaju u obzir četiri z-vrijednosti i jedan element teksture, a zapisuje se četiri elementa ili na primjer *SSAA* (engl. *Super Sampled AA*) gdje se uzimaju u obzir četiri z-vrijednosti i četiri elementa teksture, a zapisuje se četiri elementa. Bitna je razlika u memorijskoj propusnosti za kojom se zahtjev u drugom slučaju značajno povećava što utječe na brzinu izvođenja aplikacija.

Za sada smo vidjeli osnovne probleme koji se javljaju u postupku preslikavanja teksture. U nastavku ćemo razmotriti načine na koje možemo teksturu pridružiti i orijentirati prema pojedinom poligonu ili objektu te koje su opcije na raspolaganju u OpenGL-u.

12.4 Preslikavanje teksture na poligon u OpenGL-u

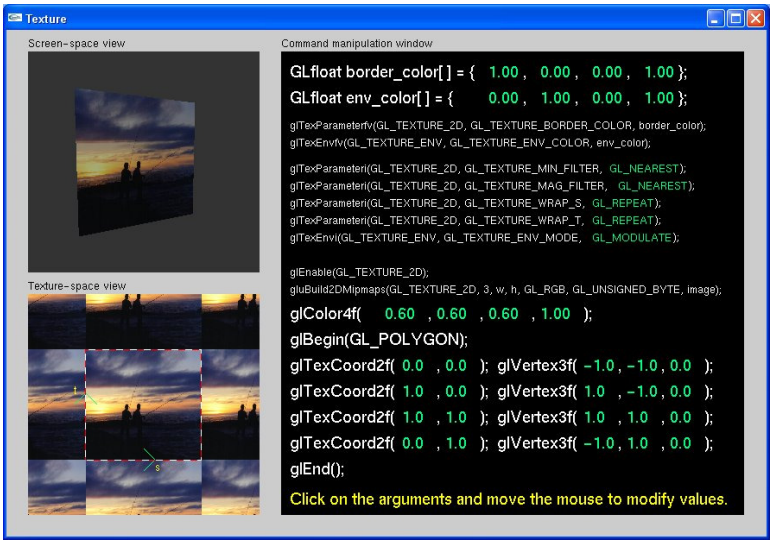
U OpenGL-u imamo različite mogućnosti vezano uz način kako teksturu pridružiti poligonu. To se, pored ostalog, odnosi na područje izvan okvira osnovne teksture, orijentaciju teksture te načine interpolacije koje smo razmatrali u prethodnom poglavlju.

Prvo ćemo pogledati kako možemo podesiti područje izvan zadanog osnovnog okvira teksture. Osnovni okvir teksture zadaje se tipično u granicama $(0, 0)$ do $(1, 1)$. No, prilikom dohvaćanja elemenata teksture može se desiti da teksturu trebamo dohvatiti izvan ovog područja, a u tom slučaju moramo imati definirano što ćemo dohvatiti.

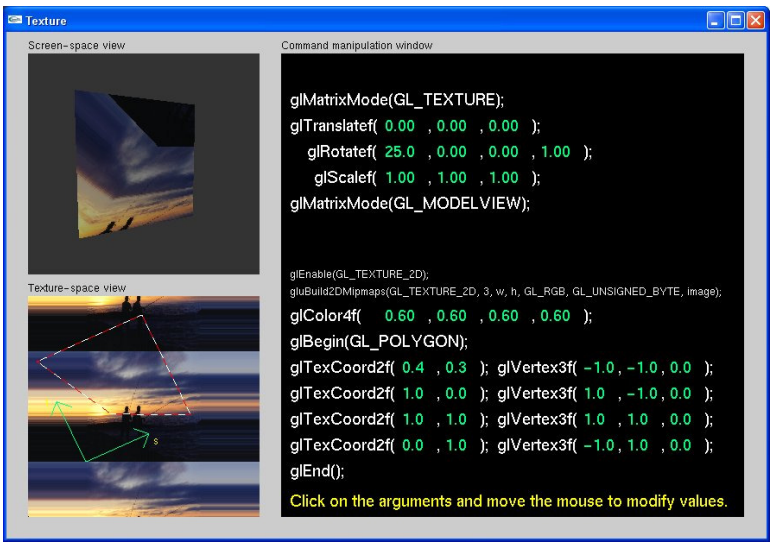
Pogledajmo primjer na slici 12.10. Prvo možemo primijetiti da se između naredbe glBegin i glEnd za svaki vrh poligona nalaze i pripadne koordinate teksture u području teksture. U prikazanom primjeru koordinatni sustav scene u kojoj je zadan poligon s četiri vrha prikazan je u gornjem lijevom uglu. U donjem lijevom uglu je prostor teksture gdje je aktivno područje određeno GlTexCoord2f u intervalu $(0, 0)$ do $(1, 1)$ označeno crveno bijelom crtkanom linijom. Izvan označenog područja u prostoru teksture koji je označen st koordinatnim osima, slika se replicira i ponavlja po s- i po t-koordinatnoj osi. Naredbom
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WARP_S, GL_REPEAT)
i pripadnim parametrima (u ovom slučaju GL_REPEAT) određeno je hoće li se tekstura replicirati (slika 12.10) ili će se jednostavno uzeti zadnja rubna vrijednost (u ovom slučaju postavljen je GL_CLAMP umjesto GL_REPEAT) u području izvan teksture (slika 12.11).

Također možemo primijetiti da možemo odabrati način na koji će se obavljati određivanje vrijednosti slikovnih elemenata kao što je opisano u prethodnom poglavlju i to posebno za slučaj umanjivanja odnosno uvećavanja (MIN_FILTER, MAG_FILTER). Trenutna opcija je interpolacija korištenjem najbližeg susjeda za oba slučaja (GL_NEAREST).

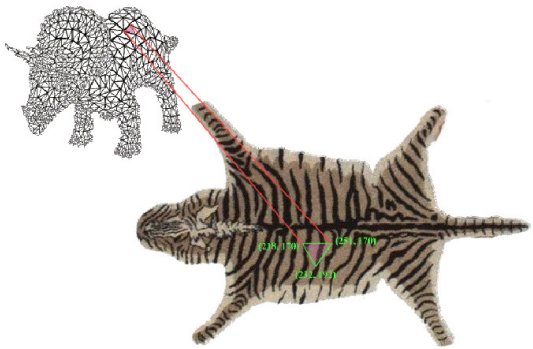
Na slici 12.11 prikazana je dodatna mogućnost upravljanja odabirom područja teksture tako da su koordinate (s, t) prve točke teksture promijenjene i cijelo područje teksture je rotirano za 25° oko z-osi. Iako je tekstura 2D, OpenGL nam dopušta obavljanje transformacija u 3D. Tako da možemo obavljati i rotacije oko bilo koje prostorne osi, no kako je tekstura u 2D prostoru, z-koordinata, iako je možemo koristiti, u konačnici se zanemaruje (postavlja na vrijednost nula).



Slika 12.10: Preslikavanje teksture u OpenGL-u.



Slika 12.11: Podešavanje područja teksture u OpenGLu.



Slika 12.12: Preslikavanje teksture na objekt postupkom foto tekstura.

Kako bi mogli rukovati područjem teksture moramo postaviti aktivnu matricu GL_TEKSTURE naredbom glMatrixMode, te odabrati translaciju, rotaciju, skaliranje ovisno o tome kako želimo podesiti područje teksture u prostoru teksture. Sve zajedno, imamo vrlo bogat skup mogućnosti oko podešavanja područja teksture i prilagođavanja našem poligonu.

U konačnici potrebno je učitati iz neke vanjske datoteke sliku koja će biti naša tekstura i vezati ju (engl. *bind*) kako bi je mogli dalje koristiti, raditi Mip-Mape i preslikati na poligon. Obzirom da postoje brojni standardi zapisa 2D slika, pogodno je odabrati neku od biblioteka koja će nam olakšati učitavanje i rad sa slikama.

Do sada smo vidjeli kako pridružiti teksturu jednom poligonu. Sada ćemo pogledati kako se radi pridruživanje teksture cijelom objektu.

12.5 Preslikavanje teksture na objekte

12.5.1 Foto teksture

Postupak koji smo razmotrili za pridruživanje teksture jednom poligonu možemo primijeniti i na cijeli objekt. No određivanje koordinata teksture u prostoru teksture i pripadne orijentacije prilično je zahtijevan postupak za ručno namještanje. Primjerice, kako je prikazano na slici 12.12, za svaki bi vrh svakog poligona tijela u tom slučaju ručno trebalo odrediti koordinate teksture u koje se taj vrh preslikava. Ovaj oblik preslikanih tekstura naziva se *fototeksturama*. Iako ovaj postupak daje vrlo lijepe rezultate, pridruživanje teksture po pojedinim dijelovima u prostoru teksture, tako da imamo lijepo nastavljanje teksture bez rupa i prilagođavanje oblika uzorka teksture obliku objekta, može biti prilično naporno.

Obrnuti pristup pridruživanja tekstura je prilikom izrade trodimenzionalnih modela raznim alatima za modeliranje gdje se pri izradi samog modela definiraju i pripadne boje odnosno teksture. Nakon toga model se programskim alatima

(a) Tijelo s teksturom. (b) Razmotani model. (c) Razmotani model.

Slika 12.13: Stvaranje *uv*-mape na osnovi modela.

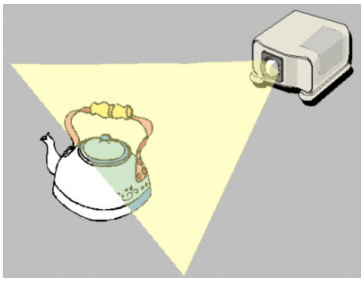
"razmota" (engl. *unwrapping*) odnosno preslika na 2D plohu u *uv*-mapu. Na slici 12.13a prikazan je primjer 3D modela objekta, na slici 12.13b zapečena *uv*-mapa teksture (engl. *baked*) i na slici 12.13c pripadna mapa normala. U tome postupku objekt se pojednostavljuje što se tiče broja poligona a detalji se onda dobiju iz mape normala u postupku ostvarivanja prikaza. U *uv*-mapu teksture često se dodaje i komponenta osvjetljenja, točnije ambijentnog zasjenjenja (engl. *ambient occlusion AO*) a uz mapu normala se dodaje i visinska mapa.

12.5.2 Projekcijske teksture

Možemo primijetiti da je u općem slučaju osnovni problem u određivanju koordinata teksture pridružene svakom od vrhova objekata. Do sada smo razmatrali jedan poligon, no mi želimo preslikati teksturu na cijeli objekt. Pogledajmo nekoliko jednostavnih pristupa ovom problemu. Prvo najjednostavnije rješenje određivanja veze između vrhova objekta i *uv*-koordinata u prostoru teksture prikazano je na slici 12.14. Ideja je pridružiti teksturu objektu kao da projiciramo neku sliku (teksturu) projektorom na proizvoljan objekt. U ovom preslikavanju možemo uzeti da je *uv*-koordinata u sustavu projektora zapravo mapa teksture. Za svaku promatranu točku u prostoru tada imamo pridijeljenu *uv*-koordinatu u prostoru teksture. Možemo primijetiti da će i vrhovi čajnika sa stražnje strane objekta imati isto pridijeljenu *uv*-koordinatu iz prostora teksture. Odnosno, pravci koji se radijalno šire iz centra projektora imat će istu boju. Pored ovakvog "perspektivnog" načina preslikavanja teksture možemo zamisliti i "ortografsko" preslikavanje s ravnine teksture na objekt. U oba slučaja ostvareno je jednostavno povezivanje vrhova proizvoljnog objekta s koordinatama u području teksture.

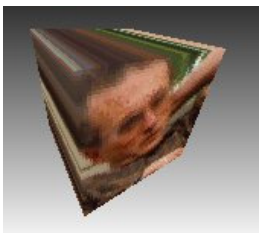
Nedostatak ovog načina preslikavanja teksture je u tome što će nam preslikana tekstura na objekt, kao što je na primjer kocka, na bočnim stranicama imati razvučenu vrijednost duž cijele stranice slikovnog elementa s prednje strane kocke kao na slici 12.15. Na stražnjoj stranici kocke bit će zrcalna slika teksture.

(a) Projiciranje teksture projektorom.

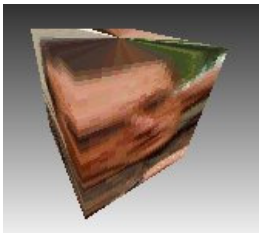


(b) Slika teksture na objektu.

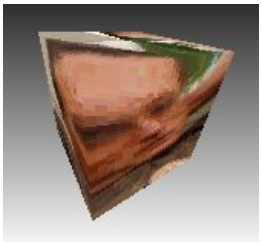
Slika 12.14: Projekcijska planarna tekstura.



(a) Ortografsko projiciranje teksture na kocku – planarna tekstura.

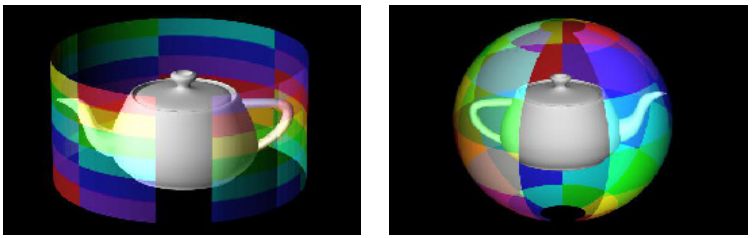


(b) Preslikavanje s cilindra.



(c) Preslikavanje s kugle.

Slika 12.15: Projekcijska planarna tekstura.



(a) Preslikavanje s cilindra. (b) Preslikavanje s kugle.

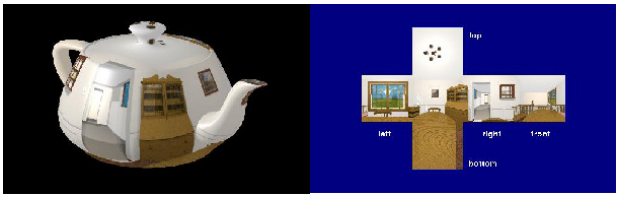
Slika 12.16: Preslikavanje teksture.

Možemo zamisliti još nekoliko jednostavnih načina preslikavanja teksture pomoću jednostavnih geometrijskih tijela. U osnovi, cilj je prvo preslikati teksturu iz prostora teksture na jednostavno geometrijsko tijelo, a zatim s tog tijela preslikati teksturu na proizvoljan objekt (slika 12.16). Jednostavni geometrijski objekti za ovu primjenu su cilindar, kugla ili kocka. Kod cilindrične teksture tekstura se prvo preslika na cilindar (slika 12.16a), a zatim s cilindra radijalno na objekt. Na gornjoj stranici kocke tada (Slika 12.15b) možemo primijetiti radijalno širenje elemenata teksture. Sferni koordinatni sastav koristi se kod preslikavanja s kugle (slika 12.16b). U zadnja dva slučaja na stražnjoj stranici kocke (slika 12.15) moći ćemo primijetiti "šav" na mjestu gdje se omotana tekstura spaja, odnosno gdje se spaja lijeva i desna strana teksture koja se preslikava na objekt.

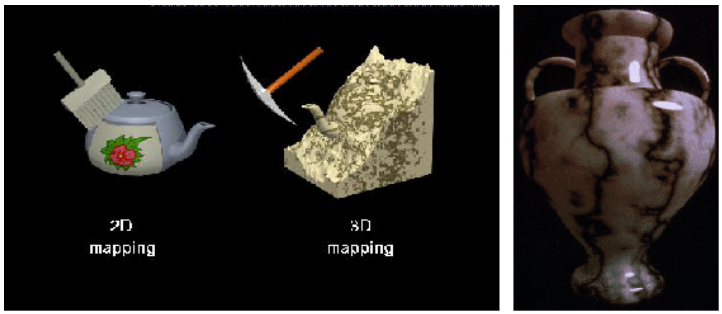
Teksturu možemo prvo preslikati i na kocku. Ovo preslikavanje se koristi kada želimo ostvariti sliku zrcaljenja okoliša na promatrani objekt (slika 12.17). Objekt se prvo pozicionira u kocku koja se nalazi negdje u sceni. Kameru postavimo u središte kocke i pogled usmjerimo redom u šest središta stranica kocke. Za svaki od ovih pogleda napravimo poseban prikaz scene i tih šest prikaza upotrijebimo kao teksture pridružene pojedinim stranicama kocke. Sada imamo mapu teksture za svaku od stranica kocke s kojih onda teksturu preslikamo na proizvoljni objekt. Možemo primijetiti na slici 12.17 da se lijevak čajnika ne zrcali na samom čajniku, niti da se ručica poklopce na zrcali na poklopcu, no to su nedostaci ovog postupka. Znači ako objekt nije konveksan, konkavni dijelovi neće imati zrcalnu sliku na samom objektu. Po ovim detaljima možemo prepoznati je li prikaz ostvaren postupkom praćenja zrake ili ovakvim trikom. U OpenGL-u postoji sklopovska podrška za preslikavanje teksture s kocke (parametar GL_TEXTURE_CUBE_MAP). Na sasvim sličan način možemo zamisliti i zrcaljenje okoliša ostvareno preslikavanjem s kugle.

12.5.3 Volumne teksture

Za razliku od 2D-tekstura koje "lijepimo" na poligone objekta, 3D-teksture odnosno volumne teksture razmatramo kao objekte koje izrađujemo od volumnog



Slika 12.17: Zrcaljenja okoliša na objektu preslikavanjem teksture s kocke.



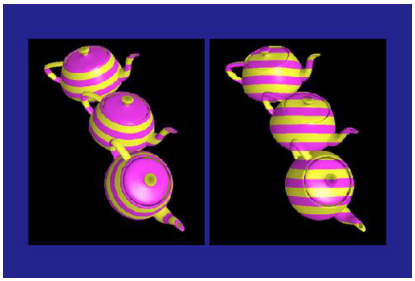
Slika 12.18: Usporedba 2D i 3D preslikavanja tekstura. Volumne ili 3D-teksture daju dojam kao da je objekt isječen od volumnog materijala (desno).

komada materijala. Na primjer, objekte koje imamo izrađene od komada drveta ili kamena ili nekog sličnog materijala (slika 12.18). Godovi drveta tako će biti vidljivi na rezbarenom objektu, a ako takav objekt presiječemo u unutrašnjosti objekta također će biti prisuta tekstura koja će se lijepo nastavljati na presječenim dijelovima. U stvari ako sada pogledamo projekcijske teksture možemo primijetiti da su to isto volumne teksture, jer su definirane za svaku točku 3D-prostora iako je to implicitno učinjeno, poznavanjem 2D-teksture koju projiciramo.

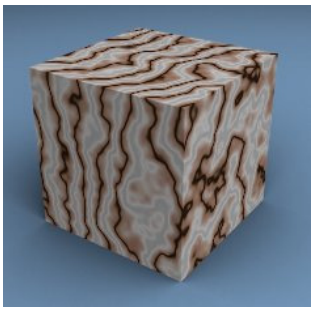
U OpenGL također postoji podrška za 3D-teksture i postavlja se naredbom glTexParameteri odnosno parametrom GL_TEXTURE_3D.

12.6 Generiranje tekstura

Teksture koje su 2D-prostoru mogu se sintetski generirati ili se mogu koristiti slike stvarnih tekstura koje nas okružuju. Važno je u nekom alatu za obradu slike urediti teksture tako da budu ponavljajuće tako da uzastopnim nizanjem, odnosno popločavanjem teksturom (engl. *tiling*) dobijemo kontinuirani uzorak bez prekida na rubovima. Kod fotografiranja uzoraka čak i male sjene ili promjene intenziteta koje su slabo uočljive na jednoj slici mogu znatno utjecati na vrlo neželjene ponavljajuće učinke pri popločavanju većih površina. Volumne teksture teže je dobiti fotografiranjem jer moramo sloj po sloj skidati bez oštećivanja nakon svakog slikanja kako bi dobili volumnu teksturu.



(a) Primjer volumne teksture koja je pridružena objektu, odnosno objekt se pomiče kroz fiksni prostor teksture.



(b) Primjer funkcijski generirane teksture.

Slika 12.19: Primjeri volumnih tekstura.

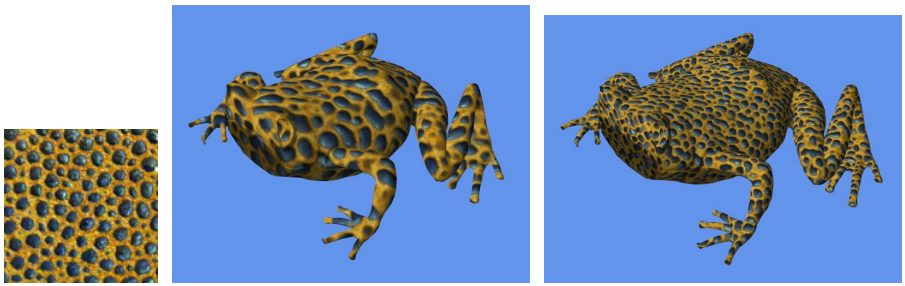
Razni su načini na koje se mogu generirati volumne teksture. Jedna od jednostavnih ideja je jednostavno pridružiti različitu boju za različitu npr. y-koordinatu u prostoru (slika 12.19). Dobivenu teksturu pri animaciji kretanja objekta u sceni možemo vezati uz objekt, ili prostor teksture možemo fiksirati a objekt onda pomicati u takvom prostoru. Ovakvim postupkom generiranja teksture u osnovi ćemo dobiti funkcijski pridruženu vrijednost svakoj točki 3D-prostora $f(x, y, z)$, odnosno volumnu teksturu. Takve teksture često se zovu i hiperteksturama. U općem slučaju, to može biti bilo kakva funkcija. Primjer funkcije za ostvarivanje volumnog uzorka mramora je:

$$\text{Marble} = \sin(n \cdot (x + A \cdot Turb(x, y, z)))$$

a pripadni izgled teksture je na slici 12.19b. Parametrima ove funkcije možemo mijenjati frekvenciju pruga kao i utjecaj turbulencije, odnosno dodavanja slučajne vrijednosti funkcijom $Turb(x, y, z)$ kako ne bi imali niz jednostavnih pravilnih linija. Ovaj način često se koristi za generiranje uzorka mramora ili godova drveta.

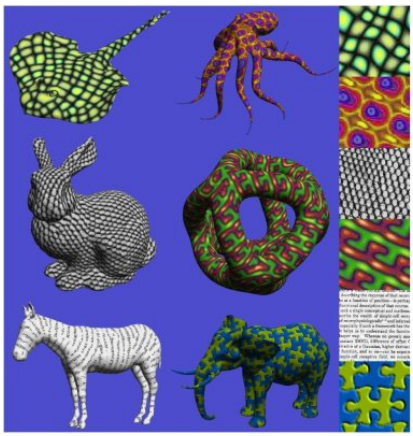
Jedan od zanimljivih načina generiranja volumnih tekstura je temeljem statističkih metoda koje pretražuju prostor zadanog uzorka u 2D-prostoru i generiraju volumni 3D-uzorak (slika 12.20). Tako generirani volumni uzorak može se pridružiti bilo kojem objektu.

Tehnike koje istovremeno generiraju teksturu i pridružuju teksturu proizvoljnom objektu na primjer zebri, vode računa o zakrivljenosti i orijentaciji pojedinih dijelova površine. Tako da na primjer na tijelu zebre budu krupnije pruge orijentirane oko trbuha, a na nogama zebre gdje je veća zakrivljenost pruge će biti tanje i isto će pratiti glavni smjer zakrivljenosti površine objekta. Na spojevima trupa i noga treba osigurati kontinuirane prijelaze koji će odgovarati stvarnoj zebri, što je vrlo izazovan zadatak. Primjeri različitih volumnih tekstura primijenjeni na različite objekte prikazani su na slici 12.21.



(a) 2D uzorak. (b) Primjena teksture. (c) Primjena teksture.

Slika 12.20: Primjer sintetski generirane volumne teksture različitih frekvencija na objektu. Volumna tekstura generirana je na temelju 2D uzorka i pridružena je objektu. Na prerezanom objektu vidljivo je da se tekstura lijepo nastavlja kao što bi bilo na stvarnom prerezanom objektu.



Slika 12.21: Primjeri različitih volumnih tekstura na objektima.

12.7 Ponavljanje

1. Što su i čemu služe teksture?
2. Kako se teksture preslikavaju na poligone?
3. Zašto se preslikavanje tekstura na poligone radi u prostoru projekcije?
4. Kako se umanjuje aliasing nastao pri preslikavanju teksture?
5. Što su mip-mape, kako su izgrađene te kako i zašto se koriste?
6. Koji se parametri koriste u OpenGL-u pri specificiranju načina preslikavanja tekstura?
7. Objasnite razliku između fototekstura, projekcijskih tekstura te volumnih tekstura.
8. Kako se ostvaruje zrcaljenje okoliša na objektu pomoću tekstura?

Poglavlje 13

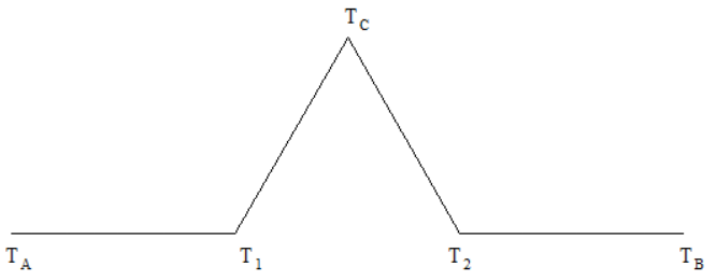
Fraktali

13.1 Uvod

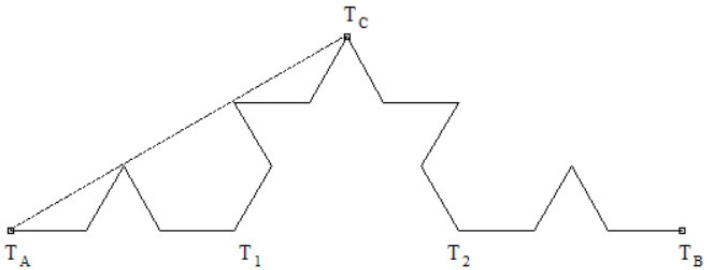
Jedno od ljepših područja računalne grafike pokriva i matematičke umotvorine – fraktale. U nastavku ćemo dati prikaz nekoliko karakterističnih fraktala i pojasniti postupak kako se do njih dolazi. Treba napomenuti da ovdje prikazani fraktali čine vrlo mali dio do sada otkrivenih fraktala, a generalno govoreći, skup fraktala je beskonačan skup tako da ih nikada niti nećemo upoznati sve. Inače, trebamo malo korigirati uvodnu rečenicu: fraktali zapravo i nisu matematičke, već su prirodne tvorevine. Matematičari su samo našli način kako baratati s njima. Pa krenimo od najjednostavnije vrste.

13.2 Samoponavljajući fraktali

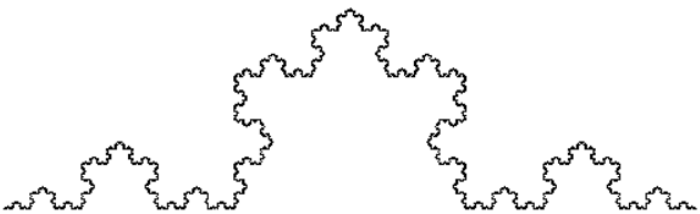
Samoponavljajući fraktali su tvorevine koje na svim skalama umanjenja zadržavaju isti karakteristični oblik. Kao primjer ćemo uzeti *Kochinu krivulju* – fraktal koji je ime dobio prema švedskoj matematičarki Helge von Koch. Kochina krivulja dobije se sljedećim jednostavnim algoritmom.



Slika 13.1: Kochina krivulja – početak konstrukcije



Slika 13.2: Kochina krivulja – drugi korak rekurzije



Slika 13.3: Kochina krivulja – rezultat uz dubinu rekurzije 6

1. Krenimo od osnovnog oblika prikazanog na slici 13.1.

2. Svaki od četiri segmenta prepravi se tako da se umjesto linije umetne čitav lik iz točke 1. Dobiti će se slika 13.2. Točke T_A i T_C spojene su spojnicom da se pokaže kako novi vrhovi leže na njoj; inače spojnica ne spada u proces generiranja fraktala.

3. Svaki segment nastao u prethodnom koraku opet se zamijeni oblikom iz koraka 1; proces se ponavlja u beskonačnost.

Praktična implementacija gornjeg algoritma neće naravno ići u beskonačnost, već do neke zadane dubine. Npr. ako algoritam pustimo do dubine 6, dobit ćemo sliku 13.3. Kochina krivulja dobije se kada se rekurziju pusti u beskonačnost. Dakako, na računalima obično prikazujemo aproksimaciju Kochine krivulje. Nitko još nije uspio nacrtati sasvim točnu krivulju. Uočimo i neka zanimljiva svojstva Kochine krivulje:

1. duljina Kochine krivulje je beskonačna,

2. duljina bilo kojeg segmenta Kochine krivulje (ma kako mali segment gledali), opet je beskonačna,

3. Kochina krivulja je neprekidna krivulja,

4. niti u jednoj točki Kochine krivulje derivacija ne postoji.

Funkcija koja iscrtava krivulju prema prethodnom algoritmu može se napisati kao vrlo jednostavna rekurzivna funkcija, prikazana u nastavku.

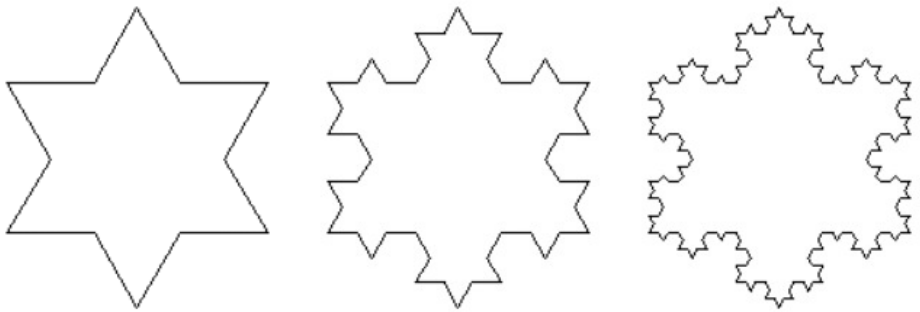
```
1  void DrawFractal1Rek(T2DRealPoint A, T2DRealPoint B,
2          T2DRealPoint C, int depth)
3  {
4   T2DRealPoint Ap,Bp,Cp;
5
6   if( depth > 5 ) {
7    MoveTo(fround(A.x),fround(A.y));
8    LineTo(fround(A.x+(B.x–A.x)/3.),
9           fround(A.y+(B.y–A.y)/3.));
10   LineTo(fround(C.x),fround(C.y));
11   LineTo(fround(A.x+2.*(B.x–A.x)/3.),
12          fround(A.y+2.*(B.y–A.y)/3.));
13   LineTo(fround(B.x),fround(B.y));
14   return;
15  }
16
17  depth++;
18
19  Ap.x = A.x; Ap.y = A.y;
20  Bp.x = A.x+(B.x–A.x)/3.; Bp.y = A.y+(B.y–A.y)/3.;
21  Cp.x = A.x+(C.x–A.x)/3.; Cp.y =A.y+(C.y–A.y)/3.;
22  DrawFractal1Rek( Ap, Bp, Cp, depth );
23
24  Ap.x = Bp.x; Ap.y = Bp.y;
25  Bp.x = C.x; Bp.y = C.y;
26  Cp.x = A.x+2.*(C.x–A.x)/3.; Cp.y =A.y+2.*(C.y–A.y)/3.;
27  DrawFractal1Rek( Ap, Bp, Cp, depth );
28
29  Ap.x = Bp.x; Ap.y = Bp.y;
30  Bp.x = A.x+2.*(B.x–A.x)/3.; Bp.y = A.y+2.*(B.y–A.y)/3.;
31  Cp.x = B.x+2.*(C.x–B.x)/3.; Cp.y =B.y+2.*(C.y–B.y)/3.;
32  DrawFractal1Rek( Ap, Bp, Cp, depth );
33
34  Ap.x = Bp.x; Ap.y = Bp.y;
35  Bp.x = B.x; Bp.y = B.y;
36  Cp.x = B.x+(C.x–B.x)/3.; Cp.y =B.y+(C.y–B.y)/3.;
37  DrawFractal1Rek( Ap, Bp, Cp, depth );
38 }
```

Funkcija prima točke A, B i C koje na prethodnim slikama odgovaraju točkama T_A, T_B i T_C, te uz njih i malo elementarne geometrije računa točke T'_A, T'_B i T'_C za svoj svaki segment te izračunate točke predaje u novi rekurzivni poziv. Maksimalna dubina rekurzije određena je prvim ispitivanjem i postavljena je na prvu veću od 5 (dakle 6). Uz ovu funkciju, još je potrebna i nerekurzivna funkcija koja će pozvati svoju rekurzivnu inačicu, a prikazana je u nastavku.

```
1  void DrawFractal1()
2  {
3     T2DRealPoint A,B,C;
4
5     A.x = 10; A.y = 200 - 10;
6     B.x = 320 - 10; B.y = 200 - 10;
7     C.x = ( A.x + B.x )/2.;
8     C.y = A.y - sqrt(3)/2.*( B.x - A.x )/3.;
9     DrawFractal1Rek(A, B, C, 1);
10 }
```

Funkcija pretpostavlja zaslon razlučivosti 320×200 i ostavlja po 10 praznih pixela lijevo, desno i ispod krivulje.

Jednostavnim proširenjem gornjeg algoritma dobiti ćemo poznatu *Kochinu pahuljicu* (engl. *Koch's Snowflake*). Umjesto od jedne linije krenut ćemo od trokuta čije su stranice segmenti prikazani u prethodnom algoritmu pod brojem 1. Dobiti ćemo lik prikazan na slici 13.4a.



(a) Početak (b) Drugi korak rekurzije (c) Treći korak rekurzije

Slika 13.4: Kochina pahuljica: ilustracija rezultata za različite dubine rekurzije

Sada svaku liniju iterativno zamijenimo tim segmentima. Razvoj slike fraktala prikazan je na slikama 13.4b i 13.4c.

Iz opisanog postupka može se vidjeti da se Kochina pahuljica može dobiti kao jednakostranični trokut čije su stranice Kochine krivulje. Matematički gledano, ovaj fraktal zanimljiv je po još nečemu: duljina njegova opsega je beskonačna, iako je površina koju lik zauzima konačna. Algoritam za crtanje pahuljice također se sastoji od dva dijela: jedne rekurzivne funkcije koja je već navedena kod Kochine krivulje (DrawFractal1Rek), i nerekurzivne funkcije koja poziva istu, koja je prikazana u nastavku.

```
1  void DrawFractalSnowflake()
2  {
3     T2DRealPoint A,B,C;
```

```
4     double visina,d;
5
6     visina = 200 - 20;
7     d = sqrt(3.)*visina/2.;
8
9     A.x = 10; A.y = 200 - 10 - visina/4.;
10    B.x = A.x + d; B.y = A.y;
11    C.x = ( A.x + B.x )/2.;
12    C.y = A.y + sqrt(3)/2.*d/3.;
13    DrawFractal1Rek(A, B, C, 1);
14
15    B.x = A.x + d/2.;
16    B.y = A.y - d*sqrt(3.)/2.;
17    C.x = (A.x+B.x)/2.-d/4.;
18    C.y = (A.y+B.y)/2.-visina/4./2.;
19    DrawFractal1Rek(A, B, C, 1);
20
21    A.x = A.x + d;
22    C.x = (A.x+B.x)/2.+d/4.;
23    C.y = (A.y+B.y)/2.-visina/4./2.;
24    DrawFractal1Rek(A, B, C, 1);
25  }
```

I ova funkcija podrazumijeva ekran 320×200 pixela. Funkcija se sastoji od tri cjeline: poziva za iscrtavanje donje Kochine krivulje, poziva za iscrtavanje lijeve Kochine krivulje te poziva za iscrtavanje desne Kochine krivulje.

Spomenimo još i zanimljiva svojstva Kochine pahuljice:

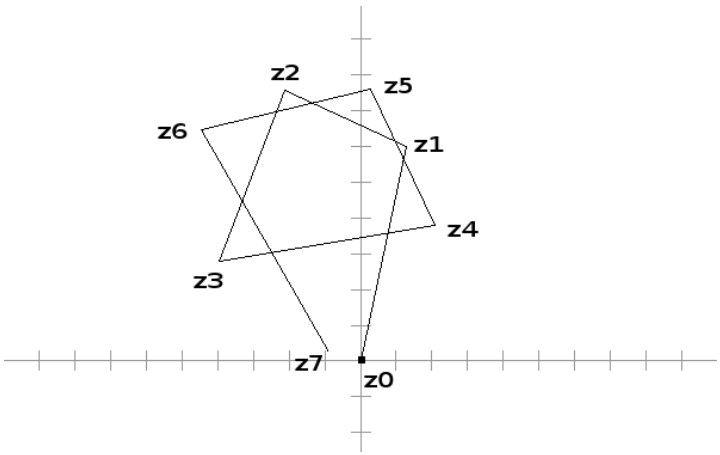
1. opseg Kochine pahuljice je beskonačan,
2. površina Kochine pahuljice je konačna.

13.3 Mandelbrotov fraktal

Kako bismo opisali postupak nastanka Mandelbrotovog fraktala, morat ćemo se najprije upoznati s temeljnijim pojmom – Mandelbrotovim skupom. Mandelbrotov skup je skup točaka u kompleksnoj ravnini, koji zadovoljava određeni uvjet s kojim ćemo se upoznati u nastavku, i čiju granicu nazivamo Mandelbrotovim fraktalom (fraktal je ime dobio prema matematičaru Benoîtu Mandelbrotu). Pa kada za neku točku c iz kompleksne ravnine kažemo da pripada Mandelbrotovom skupu? Da bismo na to odgovorili, definirajmo naprije kompleksnu rekurzivnu funkciju z_{n+1}:

$$z_{n+1} = z_n^2 + c \qquad \text{uz početni uvijet } z_0 = 0 \tag{13.1}$$

Promotrimo niz kompleksnih brojeva koje ova rekurzija generira za jedan konkretan kompleksni broj c. Primjerice, ako uzmemo $c = 0.13 + 0.6i$, dobit ćemo



Slika 13.5: Ispitivanje pripadnosti točke Mandelbrotovom skupu

sljedeći niz kompleksnih brojeva: $z_0 = 0$, $z_1 = 0.13 + 0.6i$, $z_2 = -0.2131 + 0.756i$, $z_3 = -0.3961 + 0.2778i$, $z_4 = 0.2097 + 0.3799i$, $z_5 = 0.0297 + 0.7594i$, $z_6 = -0.4458 + 0.6450i$, $z_7 = -0.0874 + 0.0249$, itd. Brojevi su prikazani na slici 13.5, pri čemu je njihova trajektorija naznačena linijama. Ono što nas zanima jest je li modul ovih kompleksnih brojeva koje generira promatrana rekurzivna funkcija ograničen, ako pustimo da n teži u beskonačnost. Riječ *ograničen* u ovom kontekstu znači da je moguće pronaći konačan broj ϵ takav da je $\forall n.z_n < \epsilon$. U geometrijskom smislu, zanima nas ako oko ishodišta kompleksne ravnine (točke $0 + 0i$) nacrtamo kružnicu radijusa ϵ, hoće li svi kompleksni brojevi koje generira ova rekurzivna jednadžba ostati unutar dijela kompleksne ravnine omeđenog tom kružnicom. Ako je odgovor *da*, tada kompleksni broj c za koji smo radili ispitivanje pripada Mandelbrotovom skupu. Ako je odgovor *ne*, odnosno ako niz divergira, kompleksni broj c ne pripada Mandelbrotovom skupu.

Pitanje na koje još treba odgovoriti jest koliki treba biti ϵ? Prema modernim spoznajama, za sve točke c Mandelbrotovog skupa, elementi pripadnog niza z_n nikada ne izlaze iz kruga radijus 2 povučenog oko ishodišta kompleksne ravnine. Ako se dogodi da u bilo kojem trenutku modul elementa niza z_n postane veći od 2, niz će divergirati, i pripadna točka c ne pripada Mandelbrotovom skupu.

Programski isječak koji ilustrira ispitivanje divergencije prikazan je u nastavku. Prilikom implementacije provjere divergira li niz očitno nije moguće raditi provjeru do beskonačnosti. Stoga metoda divergence_test kao drugi parametar prima dozvoljenu dubinu rekurzije. Ako se generira i ispita limit točaka, i njihov modul je i dalje unutar ϵ kružnice, predana točka c proglasit će se točkom koja pripada Mandelbrotovom skupu (metoda će vratiti -1). Ako se u nekom koraku

utvrdi divergencija, metoda kao povratnu vrijednost vraća korak u kojem se je to utvrdilo. Metoda kao radijus koristi $\epsilon = 2$, odnosno ispituje je li kvadrat modula veći od $2 \cdot 2 = 4$, kako bi se izbjeglo vađenje korijena.

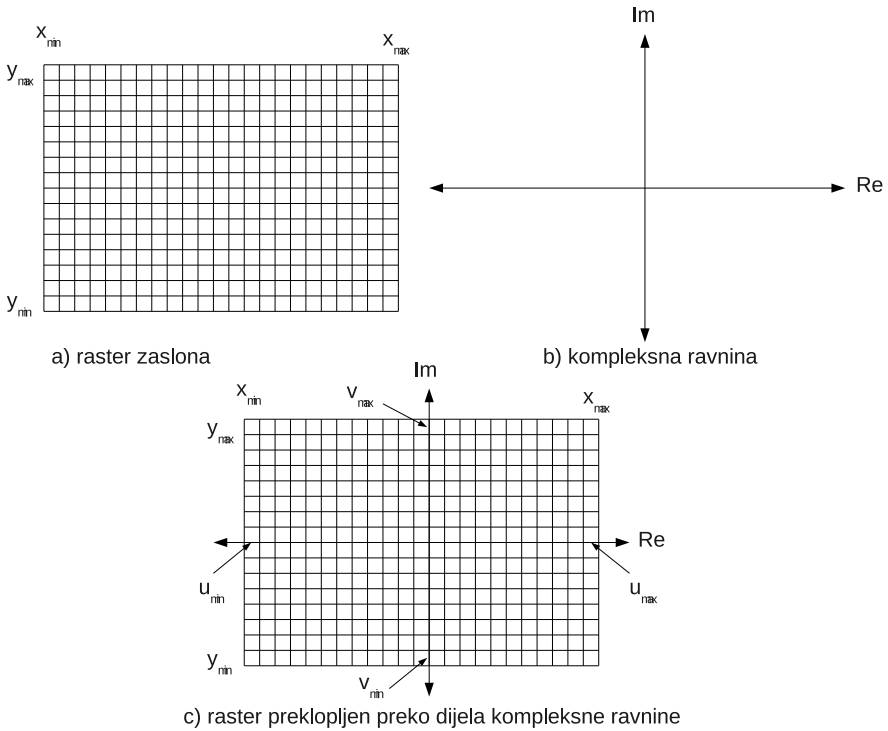
```
1  typedef struct {
2      double re;
3      double im;
4  } complex;
5
6  int divergence_test(complex c, int limit) {
7      complex z;
8      z.re = 0; z.im = 0;
9      for(int i = 1; i <= limit; i++) {
10         double next_re = z.re*z.re - z.im*z.im + c.re;
11         double next_im = 2*z.re*z.im + c.im;
12         z.re = next_re;
13         z.im = next_im;
14         double modul2 = z.re*z.re + z.im*z.im;
15         if(modul2 > 4) return i;
16     }
17     return -1;
18 }
```

No kako nacrtati Mandelbrotov fraktal? Uočimo da Mandelbrotov skup, baš kao i Mandelbrotov fraktal (granica tog skupa) imaju beskonačno točaka. S druge pak strane, ekran zaslona omogućava nam isključivo prikaz rastera - konačnog diskretnog niza slikovnih elemenata. Stoga ćemo za potrebe crtanja napraviti uzorkovanje točaka kompleksne ravnine, i samo ćemo za njih provjeriti pripadaju li one Mandelbrotovom skupu ili ne. To ćemo napraviti tako da ćemo raster zaslona preklopiti preko pravokutnog područja u kompleksnoj ravnini, i potom za svaki slikovni element definiran ekranskim koordinatama (x, y) izračunati u koju se on točku kompleksne ravnine c preslikava (slika 13.6). Pri tome lijevi rub ekrana poravnavamo s vrijednosti u_{min} na realnoj osi, desni rub ekrana poravnavamo s vrijednosti u_{max} na realnoj osi, donji rub ekrana poravnavamo s vrijednosti v_{min} na imaginarnoj osi, te gornji rub ekrana poravnavamo s vrijednosti v_{max} na imaginarnoj osi. Točke kompleksne ravnine čiji je realni dio manji od u_{min} ili veći od u_{max}, ili čiji je imaginarni dio manji od v_{min} ili veći od v_{max} uopće nećemo ispitivati.

Dio kompleksne ravnine koji je preklopljen s rasterom uzorkovat ćemo u skladu s gustoćom rastera. Promotrimo neku točku ekrana (x, y). Neka se ona u preslikava u kompleksnoj ravnini u točku (u, v). Uočimo da vrijedi:

$$\frac{x - x_{min}}{x_{max} - x_{min}} = \frac{u - u_{min}}{u_{max} - u_{min}},$$

$$\frac{y - y_{min}}{y_{max} - y_{min}} = \frac{v - v_{min}}{v_{max} - v_{min}}.$$



Slika 13.6: Crtanje Mandelbrotovog skupa

Temeljem tih izraza za zadanu ekransku točku (x, y) možemo očitati koordinate pripadne točke kompleksne ravnine (u, v):

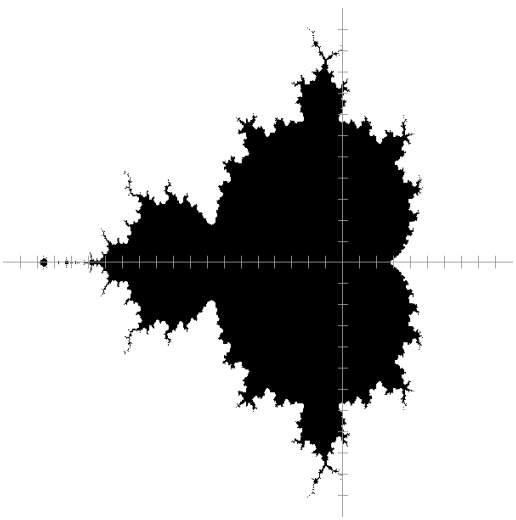
$$u = \frac{x - x_{min}}{x_{max} - x_{min}} \cdot (u_{max} - u_{min}) + u_{min} \tag{13.2}$$

$$v = \frac{y - y_{min}}{y_{max} - y_{min}} \cdot (v_{max} - v_{min}) + v_{min} \tag{13.3}$$

Crtanje Mandelbrotovog fraktala sada se svodi na ugniježđene **for**-petlje: jedna koja ide po retcima i druga koje ide po stupcima. Za svaki slikovni element ispita se pripada li Mandelbrotovom skupu, i ako pripada, točka se nacrta crnom bojom, a ako ne pripada, nacrta se bijelom bojom. Algoritam je prikazan u nastavku. Slika 13.7 prikazuje rezultat za parametre $u_{min} = -2$, $u_{max} = 1$, $v_{min} = -1.2$, $v_{max} = 1.2$ (koordinatni sustav dodan je kako bi se vidjela pozicija točaka u kompleksnoj ravnini).

```
1
2  double xmin = 0;
3  double xmax = 599;
4  double ymin = 0;
```

```
5  double ymax = 599;
6  double umin = -2;
7  double umax = 1;
8  double vmin = -1.2;
9  double vmax = 1.2;
10
11 void renderScene() {
12     glPointSize(1);
13     glBegin(GL_POINTS);
14     for(int y = ymin; y <= ymax; y++) {
15         for(int x = xmin; x <= xmax; x++) {
16             complex c;
17             c.re = (x-xmin)/(double)(xmax-xmin)*(umax-umin)+umin;
18             c.im = (y-ymin)/(double)(ymax-ymin)*(vmax-vmin)+vmin;
19             int n = divergence_test(c, 16);
20             if(n==-1) {
21                 glColor3f(0.0f, 0.0f, 0.0f);
22             } else {
23                 glColor3f(1.0f, 1.0f, 1.0f);
24             }
25             glVertex2i(x, y);
26         }
27     }
28     glEnd();
29 }
```



Slika 13.7: Mandelbrotov skup

13.3.1 Bojanje Mandelbrotovog fraktala

Kada govorimo o Mandelbrotovom fraktalu, obično u glavi imamo viziju slike žarkih boja – međutim, naš prvi pokušaj rezultirao je crno bijelom slikom. Takva slika nastala je stoga što smo gledali pripada li promatrana točka Mandelbrotovom skupu ili ne. Potrebna je jednostavna modifikacija kako bismo uveli malo više boja u sliku: umjesto da gledamo čistu pripadnost točke Mandelbrotovom skupu, idemo iskoristiti informaciju koju nam vraća metoda divergence_test – brzinu divergencije. Prisjetimo se, ta nam je metoda vraćala -1 ako divergencija nije utvrđena, a ako je, vraćala nam je korak u kojem je ona utvrđena. Iskoristimo to tako da temeljem brzine divergencije točke odaberemo prikladnu boju. Modifikacija je prikazana u programskom isječku u nastavku, a rezultat prikazuje slika 13.8.

```
1  void renderScene() {
2      glPointSize(1);
3      glBegin(GL_POINTS);
4      for(int y = ymin; y <= ymax; y++) {
5          for(int x = xmin; x <= xmax; x++) {
6              complex c;
7              c.re = (x-xmin)/(double)(xmax-xmin)*(umax-umin)+umin;
8              c.im = (y-ymin)/(double)(ymax-ymin)*(vmax-vmin)+vmin;
9              int n = divergence_test(c, 16);
10             if(n==-1) {
11                 glColor3f(0.0f, 0.0f, 0.0f);
12             } else {
13                 glColor3f((double)n/limit,
14                     1.0-(double)n/limit/2.0,
15                     0.8-(double)n/limit/3.0);
16             }
17             glVertex2i(x, y);
18         }
19     }
20     glEnd();
21 }
```

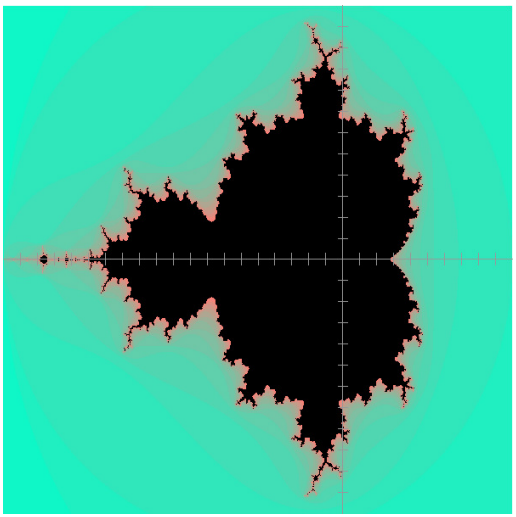
Umjesto da Mandelbrotov fraktal gledamo na područje od poprilici ± 2, zanimljive slike dobit ćemo i ako promatramo neko uže područje (što zapravo odgovara uvećavanju tog dijela slike i prikazivanju na zaslonu). Slika 13.9 sadrži niz pogleda na različite dijelove Mandelbrotovog fraktala, koji su dobiveni promatranjem dijela kompleksne ravnine koji odgovara pravokutnom području ukupne širine i visine w, čiji je centar točka $(C_x, C_y) = (-0.7454265, 0.1130090)$. Uz ovu konvenciju, vrijedi: $u_{min} = C_x - w/2$, $u_{max} = C_x + w/2$, $v_{min} = C_y - w/2$, $v_{max} = C_y + w/2$.

13.4 Julijev fraktal i Julijeva krivulja

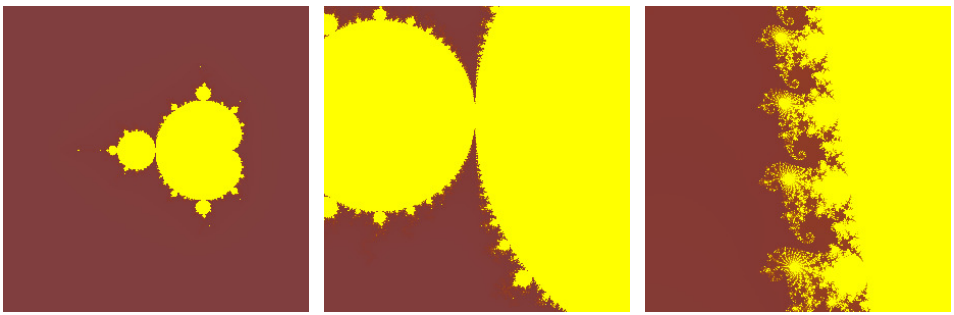
Da bismo korektno definirali Julijev skup, morali bismo se pozabaviti teorijom dinamičkih sustava i pojmovima poput atraktora, orbita i sl. Umjesto toga, ovdje ćemo pokušati definirati Julijev skup jednostavnije (i možda ne baš skroz matematički korektno). Pogledajmo jedan jednostavan primjer kompleksne funkcije kompleksne varijable:

$$f(z) = z^2.$$

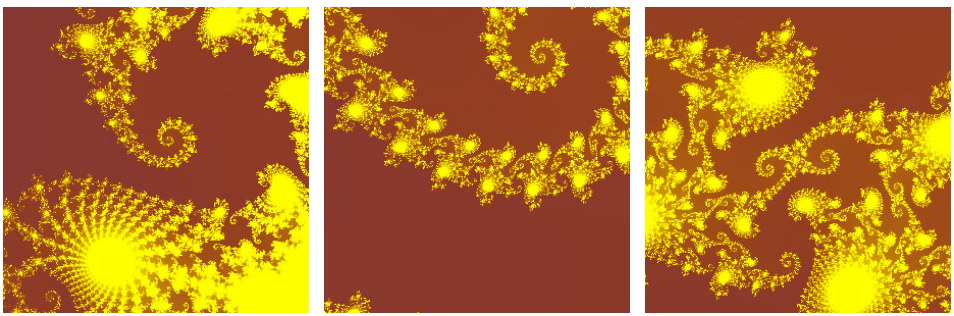
Zanima nas slična stvar kao i kod Mandelbrotovog skupa: ako odaberemo neki početni kompleksni broj z_0, izračunamo vrijednost funkcije $z_1 = f(z_0)$, potom tu vrijednost uzmemo kao argument i ponovno izračunamo vrijednost funkcije $z_2 = f(z_1) = f(f(z_0))$, i tako postupak nastavimo u beskonačnost, što će se dogoditi s nizom brojeva koje dobivamo? U slučaju promatrane funkcije, lako je pokazati da su moguća tri scenarija. Ako je modul od z_0 bio manji od 1, niz će konvergirati u točku $0 + 0i$ (to je jedan atraktor). Ako je modul od z_0 bio veći od 1, niz će divergirati u beskonačnost. Konačno, ako je modul od z_0 bio točno jednak 1, niz će se sastojati od brojeva koji i sami imaju modul jednak 1. Ovo ispitivanje možemo napraviti za sve brojeve kompleksne ravnine. Prema rezultatu tog ispitivanja, sve brojeve kompleksne ravnine $z \in \mathbb{C}$ podijelit ćemo u dva skupa. Sve točke $z \in \mathbb{C}$ za koje prethodni niz divergira zvat ćemo *escape set*. Sve točke koje konvergiraju prema atraktoru (u našem slučaju prema ishodištu) tvore *trapping set*. *Granica ovih dvaju skupova čini Julijev skup*, i za promatranu funkciju prikazana je na slici 13.10. Crnom bojom prikazan je *trapping set*, bijelom *escape set* – granica (točke na jediničnoj kružnici) čini Julijev



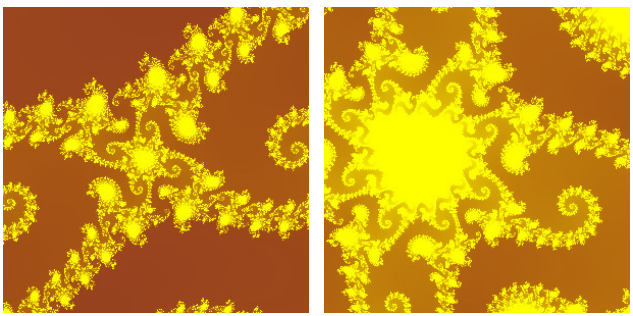
Slika 13.8: Mandelbrotov skup uz bojanje prema brzini divergencije



(a) $w = 4$ (b) $w = 4/5$ (c) $w = 4/(5^3)$

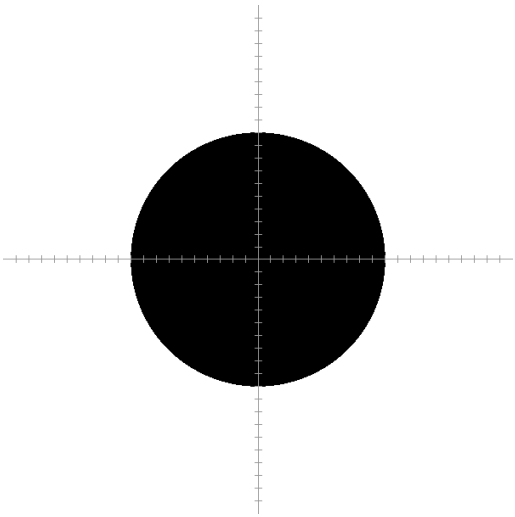


(d) $w = 4/(5^4)$ (e) $w = 4/(5^5)$ (f) $w = 4/(5^6)$



(g) $w = 4/(5^7)$ (h) $w = 4/(5^8)$

Slika 13.9: Mandelbrotov fraktal uz različita uvećanja



Slika 13.10: Julijev skup za funkciju $f(z) = z^2$

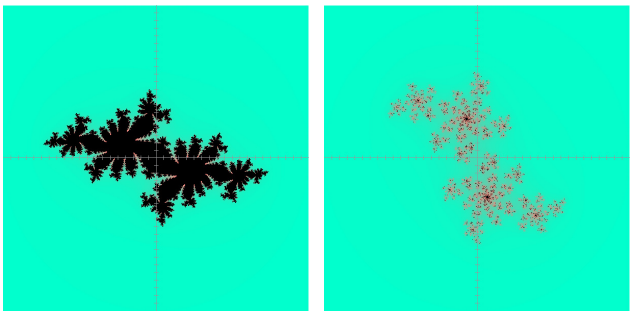
skup, i uočimo da je to u ovom slučaju jednostavna krivulja – kružnica (nije fraktal). Francuski matematičar Gaston Julia zaslužan je za otkriće Julijeva skupa, koji je usko vezan sa skupom koji je otkrio također francuski matematičar Pierre Fatou – Fatouovim skupom, koji čini komplement Julijevog skupa.

Uočimo da promatranu funkciju možemo i poopćiti. Kada se danas govori o Julijevom skupu, uobičajeno se misli barem na funkciju oblika:

$$f(z) = z^2 + c$$

gdje su $z, c \in \mathbb{C}$ (iako je moguće koristiti i druge funkcije). Uočimo da je ovo isti izraz koji smo promatrali i kod Mandelbrotovog fraktala. Za Mandelbrotov skup točka c bila je funkcija promatrane točke ekrana (nju smo mijenjali) dok je z_0 bio 0. Kod Julijevog skupa parametar c se fiksira na početku, a svaka točka ekrana (x, y) odredit će zasebnu početnu točku $z_0 = (u, v)$. Uzmemo li kao $c = -2$, Julijev skup bit će skup točaka koje leže na realnoj osi i protežu se od -2 do 2 (točke čine segment linije). Interesantno, za sve točke koje se ne nalaze na tom segmentu, iterativno preslikavanje će divergirati u beskonačnost. U ovom slučaju Julijev skup opet čini krivulju i to krivulju koja nije fraktal.

Konačno, za različite druge vrijednosti $c \in \mathbb{C}$ dobit ćemo različite Julijeve skupove. Svi Julijevi skupovi mogu se podijeliti u dvije grupe: *povezani* i *nepovezani* (slika 13.11). Povezani Julijev skup nastaje kada kao vrijednost parametra c odaberemo vrijednost koja je u Mandelbrotovom skupu. U tom slučaju, od svake točke koja pripada Julijevu skupu do svake druge točke koja pripada Julijevu skupu moguće je doći direktno prateći točke koje su također u Julijevu skupu (otuda pojam *povezan* skup). Ako kao c odaberemo vrijednost koja nije u



(a) $c.re = -0.67$, $c.im = 0.34$ (b) $c.re = 0.11$, $c.im = 0.65$

Slika 13.11: Povezan i nepovezan Julijev skup

Mandelbrotovom skupu, rezultat će biti nepovezan Julijev skup. Za takav skup kažemo da predstavlja *prašinu* (engl. *dust*) – to je skup sastavljen od beskonačnog broja *izoliranih* točaka koje su kondenzirane u grupe. Taj je skup također beskonačno gust: za svaki konačno mali radijus oko svake točke Julijevog skupa postoji još barem jedna točka koja također pripada Julijevom skupu.

Kod koji uvažava ove razlike prikazan je u nastavku, a nekoliko dodatnih primjera slika zajedno s korištenim parametrom c prikazano je na slici 13.12.

```
1  int divergence_test(complex z0, complex c, int limit,
2              int epsilonSquare) {
3      complex z;
4      z.re = z0.re; z.im = z0.im;
5      double modul2 = z.re*z.re + z.im*z.im;
6      if(modul2 > epsilonSquare) return 0;
7      for(int i = 1; i <= limit; i++) {
8          double next_re = z.re*z.re - z.im*z.im + c.re;
9          double next_im = 2*z.re*z.im + c.im;
10         z.re = next_re;
11         z.im = next_im;
12         double modul2 = z.re*z.re + z.im*z.im;
13         if(modul2 > epsilonSquare) return i;
14     }
15     return -1;
16 }
17
18 int max(int a, int b) {
19     return (b>a) ? b : a;
20 }
21
22 void renderScene() {
23     int limit = 64;
24     complex c;
25     // Ovdje je odabran parametar 'c':
```

```
26      c.re = 0.11; c.im = 0.65;
27      double epsilon = max(c.re*c.re + c.im*c.im, 2.0);
28      double epsilonSquare = epsilon * epsilon;
29      glPointSize(1);
30      glBegin(GL_POINTS);
31      for(int y = ymin; y <= ymax; y++) {
32          for(int x = xmin; x <= xmax; x++) {
33              complex z0;
34              z0.re = (x-xmin)/(double)(xmax-xmin)*(umax-umin)+umin;
35              z0.im = (y-ymin)/(double)(ymax-ymin)*(vmax-vmin)+vmin;
36              int n = divergence_test(z0, c, limit, epsilonSquare);
37              if(n==-1) {
38                  glColor3f(0.0f, 0.0f, 0.0f);
39              } else {
40                  glColor3f((double)n/limit,
41                      1-(double)n/limit/2.0,
42                      0.8-(double)n/limit/3.0);
43              }
44              glVertex2i(x, y);
45          }
46      }
47      glEnd();
48  }
```

13.5 IFS fraktali

IFS u imenu ove vrste fraktala dolazi od engleskog naziva *Iterated Function System*, što opisuje postupak kojim se dolazi do fraktala. Ideja je sljedeća. Neka je dana afina transformacija:

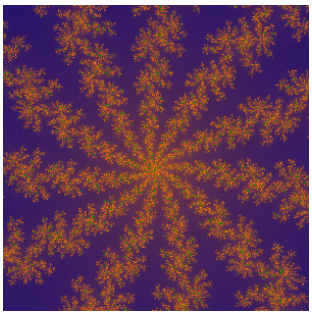
$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{13.4}$$

Ova transformacija točku $T_i = (x_i, y_i)$ preslikava u točku $T_{i+1} = (x_{i+1}, y_{i+1})$, pri čemu matričnu jednadžbu (13.4) možemo raspisati po komponentama kako slijedi.

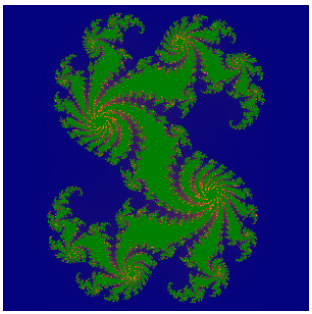
$$x_{i+1} = a \cdot x_i + b \cdot y_i + e \tag{13.5}$$

$$y_{i+1} = c \cdot x_i + d \cdot y_i + f \tag{13.6}$$

Fraktal ćemo dobiti tako da definiramo n različitih transformacija i za svaku transformaciju definiramo vjerojatnost p_i da će ta transformacija biti primijenjena (mora vrijediti $\sum_{i=1}^{n} p_i = 1$). Ovakav sustav transformacija tipično ćemo prikazivati u tabličnom obliku, gdje će retci tablice biti transformacije, a stupci

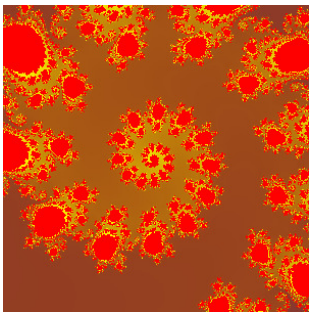


(a) $c.re = 0.179180$, $c.im = 0.588843$

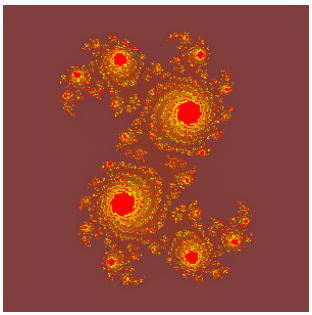


(b) $c.re = 0.32$, $c.im = 0.043$

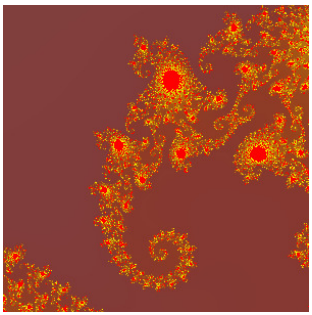
(c) $c.re = -0.807984$, $c.im = -0.145732$



(d) $c.re = -0.500036$, $c.im = -0.520156$



(e) $c.re = 0.371588$, $c.im = -0.155767$



(f) $c.re = -0.74543$, $c.im = 0.11301$

Slika 13.12: Primjeri Julijevog fraktala

tablice koeficijenti odnosno pripadna vjerojatnost. Primjer takvog zapisa prikazan je u tablici 13.1. IFS se u prikazanom primjeru sastoji od 4 funkcije, pri čemu je vjerojatnost primjene prve jednaka 1%, vjerojatnost primjene druge jednaka je 85%, te su vjerojatnosti primjene treće i četvrte jednaka 7% za svaku.

13.5.1 Kako crtamo IFS fraktal

IFS fraktal nastaje kao rezultat iterativne primjene definiranih transformacija na neku početnu točku. Pri tome se u svakom koraku posredstvom slučajnog mehanizma odabere koja će transformacija biti korištena u tom koraku (dakako, proporcionalno vjerojatnostima primjene definiranima uz svaku transformaciju). Postupak se ponovi određen broj puta (do zadane dubine), i potom se nacrta samo konačni piksel. Čitav postupak potom se ponavlja veliki broj puta. Izvorni kod programa koji crta fraktal u skladu s IFS-om definiranim u tablici 13.1 prikazan je u nastavku.

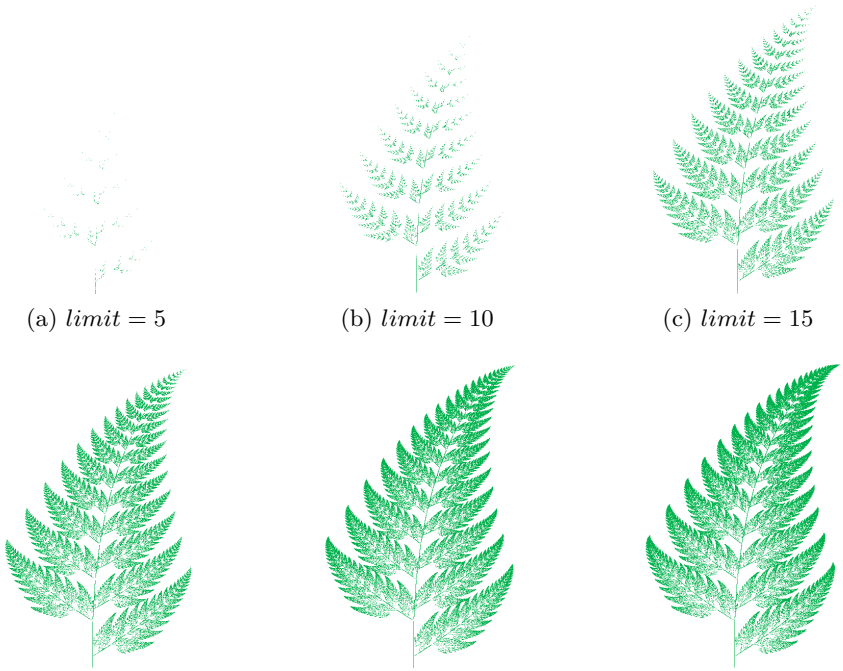
```
1  int zaokruzi(double d) {
```

a	b	c	d	e	f	p_i
0.00	0.00	0.00	0.16	0.00	0.00	0.01
0.85	0.04	-0.04	0.85	0.00	1.60	0.85
0.20	-0.26	0.23	0.22	0.00	1.60	0.07
-0.15	0.28	0.26	0.24	0.00	0.44	0.07

Tablica 13.1: IFS za primjer vrste paprati

```
2       if(d>=0) return (int)(d+0.5);
3       return (int)(d-0.5);
4   }
5
6   void renderScene() {
7       int limit = 25;
8       glPointSize(1);
9       glColor3f(0.0f, 0.7f, 0.3f);
10      glBegin(GL_POINTS);
11      double x0, y0;
12      for(int brojac = 0; brojac < 200000; brojac++) {
13          // pocetna tocka:
14          x0 = 0;
15          y0 = 0;
16          // iterativna primjena:
17          for(int iter = 0; iter < limit; iter++) {
18              double x,y;
19              int p = rand() % 100;
20              if(p<1) {
21                  x = 0;
22                  y = 0.16*y0;
23              } else if(p<8) {
24                  x = 0.2*x0-0.26*y0+0;
25                  y = 0.23*x0+0.22*y0+1.6;
26              } else if(p<15) {
27                  x = -0.15*x0+0.28*y0+0;
28                  y = 0.26*x0+0.24*y0+0.44;
29              } else {
30                  x = 0.85*x0+0.04*y0+0;
31                  y = -0.04*x0+0.85*y0+1.6;
32              }
33              x0 = x; y0 = y;
34          }
35          // crtanje konacne tocke
36          glVertex2i(zaokruzi(x0*80+300), zaokruzi(y0*60));
37      }
38      glEnd();
39  }
```

Iterativna primjena transformacija ovog IFS-a daje x-koordinate iz raspona poprilici ± 2.5, te y-koordinate iz raspona od 0 do poprilici 10. Kako je prikazani



(a) *limit* = 5 (b) *limit* = 10 (c) *limit* = 15

(d) *limit* = 20 (e) *limit* = 25 (f) *limit* = 30

Slika 13.13: Primjeri lista paprati generiranog IFS-om

izvorni kod korišten za crtanje na prozoru dimenzija 600×600, konačna točka prije crtanja dodatno se transformira tako da se x skalira s 80 (čime mu raspon postaje ± 200) i zatim translatira za 300 piksela (prema centru prozora). y-koordinata samo se skalira sa 60, čime joj raspon postaje od 0 do poprilici 600. Primjena ovog algoritma uz različite vrijednosti parametra *limit* prikazana je na slici 13.13. Taj fraktal poznat je pod nazivom *Barnsley*-eva paprat.

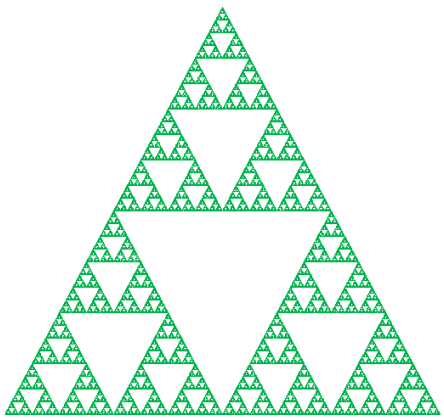
Različitim IFS-ovima moguće je dobiti niz različitih fraktala. Primjerice, tablica 13.2 prikazuje transformacije koje generiraju poznati *trokut Sierpinskog*, koji je potom nacrtan prilagodbom prethodnog programa, i prikazan je na slici 13.14. Spomenimo samo da je kao konačna transformacija prije crtanja piksela korišteno $x \leftarrow x \cdot 200 + 50$, $y \leftarrow y \cdot 300$.

13.5.2 Konstrukcija IFS fraktala

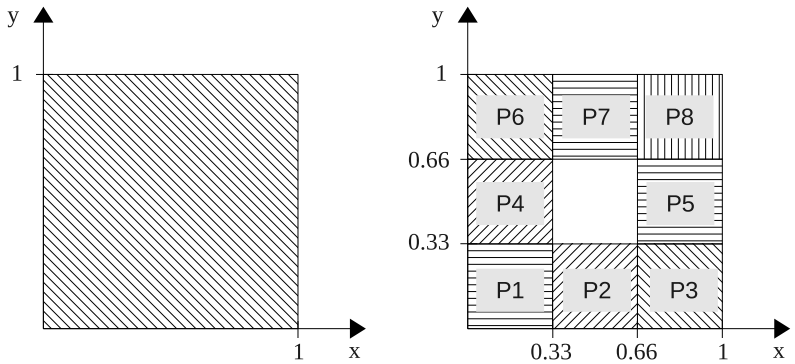
U ovoj sekciji pokušat ćemo približiti način na koji konstruiramo IFS fraktale, ali bez ulaženja u detaljna matematička razmatranja koja su podloga čitavog procesa. Proces je ilustriran na slici 13.15. Prisjetimo se najprije – ovdje govorimo o fraktalima koji su samosličnи; uvećamo li neki dio fraktala, opet ćemo uočiti

a	b	c	d	e	f	p_i
0.5	0.0	0.0	0.5	0.0	0.0	1/3
0.5	0.0	0.0	0.5	1.28	0.0	1/3
0.5	0.0	0.0	0.5	0.64	1.11	1/3

Tablica 13.2: IFS za trokut Sierpinskog



Slika 13.14: Trokut Sierpinskog



Slika 13.15: Konstrukcija IFS fraktala

jednaku građu. Imajući to u vidu, zamislimo da je čitav fraktal obuhvaćen jediničnim pravokutnikom (pravokutnik koji je smješten u ishodište, i ima visinu i širinu jednaku 1).

Građu fraktala obuhvaćenog tim pravokutnikom želimo prikazati pomoću više manjih pravokutnika – od kojih će svaki ponovno biti umanjena kopija tog istog fraktala. U ovom trenutku važno je samo zapamtiti da ti pravokutnici moraju biti manji od polaznog pravokutnika, da bi proces konvergirao. Primjerice, odlučili smo se da će fraktal imati imati 8 umanjenih kopija, koje su na slici prikazane kao pravokutnici $P1$ do $P8$. Središnji dio bit će prazan. Naš je zadatak sada pronaći afine transformacije w_i koje će originalni pravokutnik preslikati u svaki od pravokutnika $P1$ do $P8$. Krenimo redom: stranica svakog od tih umanjenih pravokutnika velika je upravo 1/3 originalne – što znači da trebamo skaliranje po obje osi i to s faktorom 1/3. Samo pomoću te transformacije originalni pravokutnik preslikava se direktno u $P1$, pa je prva transformacija w_1 jednaka:

$$\mathrm{w1}: \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Pravokutnik $P2$ dobit ćemo na sličan način: originalni pravokutnik skalirat ćemo s 1/3, i potom translatirati po osi x za 1/3. Za $P3$ radimo isto, samo što je pomak za 2/3. Pravokutnici $P4$ i $P5$ nakon skaliranja translatirani su po osi y za 1/3, a $P5$ je dodatno translatiran i po osi x za 2/3. Konačno, pravokutnici $P6$, $P7$ i $P8$ nakon skaliranja translatirani su po osi y za 2/3, te je $P7$ dodatno translatiran i po osi x za 1/3 a $P8$ za 2/3. Ovime smo dobili i ostatak transformacija:

$$\mathrm{w2}: \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} 1/3 \\ 0 \end{bmatrix}$$

a	b	c	d	e	f	p_i
0.33	0.0	0.0	0.33	0.0	0.0	1/8
0.33	0.0	0.0	0.33	0.33	0.0	1/8
0.33	0.0	0.0	0.33	0.67	0.0	1/8
0.33	0.0	0.0	0.33	0.0	0.33	1/8
0.33	0.0	0.0	0.33	0.67	0.33	1/8
0.33	0.0	0.0	0.33	0.0	0.67	1/8
0.33	0.0	0.0	0.33	0.33	0.67	1/8
0.33	0.0	0.0	0.33	0.67	0.67	1/8

Tablica 13.3: Tablica za IFS za konstruirani fraktal

$$\text{w3}: \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} 2/3 \\ 0 \end{bmatrix}$$

$$\text{w4}: \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} 0 \\ 1/3 \end{bmatrix}$$

$$\text{w5}: \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} 2/3 \\ 1/3 \end{bmatrix}$$

$$\text{w6}: \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} 0 \\ 2/3 \end{bmatrix}$$

$$\text{w7}: \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} 1/3 \\ 2/3 \end{bmatrix}$$

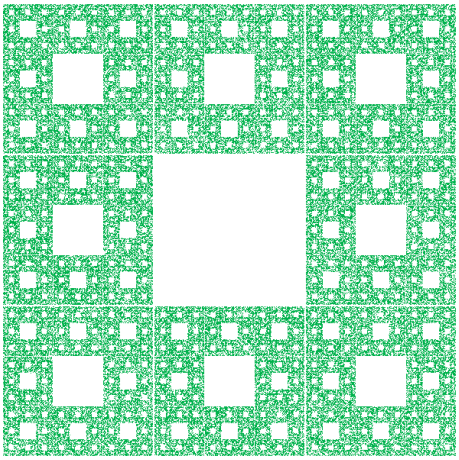
$$\text{w8}: \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} 2/3 \\ 2/3 \end{bmatrix}$$

Tablični prikaz ovih transformacija uobičajen kod IFS-fraktala dan je u tablici 13.3. Svim transformacijama dodjelili smo jednaku vjerojatnost (1/8).

Ako temeljem podataka iz tablice 13.3 preradimo izvorni kod koji je crtao trokut Sierpinskog, rezultat koji ćemo dobiti prikazan na slici 13.16. Uočite na slici položaj svakog od pravokutnika $P1$ do $P8$, i njihov sadržaj.

Pokušajte temeljem ovog razmatranja objasniti kako je nastao trokut Sierpinskog, koji smo prikazali na slici 13.14. Kako su tamo pravokutnici bili složeni, odnosno koje su afine transformacije korištene?

Spomenimo još i da nam osim skaliranja i translacije na raspolaganju stoji i rotacija, čijom se primjenom mogu dobiti interesantni fraktalni oblici. A linijski segmenti (primjerice, peteljka kod paprati) dobiju se tako da se pravokutnik po jednoj dimenziji skaliranjem degenerira u segment linije (primjerice, transformacijom koja x skalira s 0); upravo je takva transformacija prikazana u prvom retku tablice 13.1.



Slika 13.16: Rezultat konstrukcije IFS fraktala – tepih Sierpinskog

13.6 Lindermayerovi sustavi

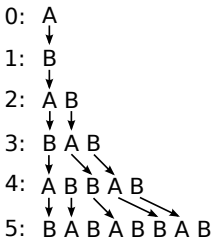
Lindermayerove sustave, ili kraće, *L-sustave* predložio je 1968. godine mađarski biolog Aristid Lindenmayer. L-sustav je formalni jezik koji se temelji na sustavima s prepisivanjem (engl. *rewriting systems*), a razvijen je kako bi se modeliralo ponašanje biljnih stanica. Osnovna ideja je vrlo jednostavna. Zamislimo sustav koji radi nad znakovnim nizovima, pri čemu znakovi dolaze iz ograničenog alfabeta. Za sustav se definira početni uzorak (koji još zovemo i *axiom*), te niz pravila koja govore kako se dijelovi postojećeg uzorka zamjenjuju novim (tipično složenijim) dijelovima. U svakom se koraku svi znakovi postojećeg niza temeljem definiranih pravila zamjenjuju novim nizovima – paralelno. Ovaj paralelizam motiviran je ponašanjem živih stanica koje se u tkivu ne dijele slijedno, već se mnoštvo dioba i promjena događa paralelno. Postoji više vrsta L-sustava, a mi ćemo se u nastavku upoznati s deterministčkim kontekstno-neovisnim L-sustavom poznatim kao *DOL-sustav*.

Formalno, DOL-sustav definiran je kao uređena trojka:

$$\mathbf{G} = (\mathbf{V}, \omega, \mathbf{P})$$

pri čemu je $\mathbf{V}$ skup znakova alfabeta, ω početni niz odnosno aksiom te $\mathbf{P}$ skup produkcijskih pravila.

Neka je alfabet $\mathbf{V} = A, B$, aksiom $\omega = A$ te neka je skup produkcijskih pravila $\mathbf{P} = \{A \to B, B \to AB\}$. Prvo produkcijsko pravilo kaže nam da svaki znak A trebamo zamijeniti znakom B; drugo produkcijsko pravilo kaže nam da svaki znak B trebamo zamijeniti novim nizom AB. Prepisivanje do dubine 5 prikazano je na slici 13.17. Pogledamo li duljine znakovnih nizova nastalih u svakom koraku, uočit ćemo da ovaj sustav generira slijed Fibonaccijevih brojeva.



Slika 13.17: L-sustav za Fibonaccijeve brojeve

Kako bismo mogli dalje, potrebno je još nizove koje sustav generira povezati s računalnom grafikom. Odnosno, s popularno nazvanom grafikom kornjače (engl. *turtle graphics*). Evo ideje. Zamislimo malu kornjaču koju smo stavili u lijevi donji ugao ekrana i okrenuli tako da gleda prema desno. Svaki simbol generiranog niza kornjača će tumačiti kao jednu naredbu koju treba napraviti. Izvođenjem tih naredbi kornjača će se pomicati po ekranu ostavljajući za sobom trag čime će u konačnici nastati slika koja odgovara promatranom nizu. Modelirajmo kornjaču jednostavnom podatkovnom strukturom prikazanom u nastavku. Kornjača pamti svoje x i y-koordinate na ekranu, te smjer u kojem gleda. Smjer opisujemo kutem u stupnjevima, pri čemu 0 odgovara pogledu u desno.

```
1  typedef struct {
2      double x;
3      double y;
4      double angle;
5  } Turtle;
```

Definirajmo i naredbe koje kornjača razumije.

Simbol	**Značenje**
F	Kornjača se pomiče za iznos δ u smjeru u kojem gleda. Ovo će rezultirati novom pozicijom: $x \leftarrow x + \delta \cdot \cos(angle)$ te $y \leftarrow y + \delta \cdot \sin(angle)$. Prilikom pomicanja na novu poziciju kornjača će na ekranu ostaviti trag.
f	Isto kao i F samo što prilikom pomicanja na novu poziciju kornjača neće na ekranu ostaviti trag.
+	Kornjača se okreće za kut ψ u smjeru suprotnom od smjera kazaljke na satu. Pozicija se ne mijenja.
-	Kornjača se okreće za kut ψ u smjeru kazaljke na satu. Pozicija se ne mijenja.

Uz ovako definirane naredbe spremni smo za crtanje Kochine krivulje uporabom L-sustava:

$$\mathbf{G} = (\{F, +, -\}, F, \{F \to F + F - -F + F\}).$$

Slika koju ovaj sustav generira prikazana je na slici 13.18. Pri tome je kao parametar δ korištena fiksna vrijednost 55 piksela, a kut ψ bio je 60. Tri slike prikazane na slici 13.18 pri tome odgovaraju aksiomu F (gornja slika), nizu $F+F--F+F$ koji odgovara prvoj iteraciji primjene produkcijskih pravila na aksiom (srednja slika) te nizu $F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F$ koji odgovara drugoj iteraciji primjene produkcijskih pravila na aksiom (donja slika). Budući da tijekom crtanja pomak δ držimo fiksnim, slika koju dobivamo svakom sljedećom iteracijom sve je veća i veća.

(a) aksiom, $\delta = \delta_0$ (b) dubina 1, $\delta = \delta_0$ (c) dubina 2, $\delta = \delta_0$

Slika 13.18: Kochina krivulja generirana L-sustavom bez skaliranja pomaka

Problemu s rastom slike možemo doskočiti tako da se prisjetimo koja je karakteristika Kochine krivulje: prilikom supstitucije svaki segment mijenja se umanjenom kopijom kod koje su duljine segmenata jednake trećini duljine promatranog segmenta. To će za posljedicu imati eksponencijalan pad duljine segmenta s iteracijama. Označimo s δ_0 početnu duljinu jednog segmenta (dakle, ono što se crta aksiomom). Tada ćemo kao iznos pomaka koji ćemo koristiti ako crtamo niz dobiven L-sustavom uz ograničenje dubine na d koristiti $\delta = \delta_0 \cdot \left(\frac{1}{3}\right)^d$. Slike dobivene uz takvu korekciju prikazane su na slici 13.19, gdje se može uočiti da su uz sve tri dubine (0, 1 i 2) generirane slike jednakih dimenzija.

(a) aksiom, $\delta = \delta_0$ (b) dubina 1, $\delta = \delta_0 \cdot \frac{1}{3}$ (c) dubina 2, $\delta = \delta_0 \cdot \left(\frac{1}{3}\right)^2$

Slika 13.19: Kochina krivulja generirana L-sustavom uz skaliranje pomaka

Isječak koda koji crta Kochinu krivulju na opisani način prikazan je u nastavku. Pri tome su prikazane samo metode ključne za crtanje navedene slike. Dijelovi koda koji inicijaliziraju *GLUT* i pripremaju opće postavke isti su kao i svim do sada prikazanim primjerima za 2D slike. Metoda pronadiPravilo (...) u skupu pravila traži pravilo koje je primjenjivo na zadani simbol. Ako takvo pravilo ne postoji, metoda vraća NULL. Metoda moveTurtle(...) ažurira poziciju kornjače zadanim pomakom u smjeru u kojem kornjača trenutno gleda.

Metoda lSustav (...) predstavlja implementaciju sustava prepisivanja do zadane dubine. Metoda prima *aksiom* te zadanu dubinu, i generira konačni niz znakova

uporabom definiranih pravila. U svakoj iteraciji ova metoda prolazi kroz dotada generirani niz i za svaki znak niza traži pravilo koje kaže čime se znak mijenja. Ako se ne pronađe odgovarajuće pravilo, metoda prepisuje taj znak (ponaša se kao da je pronađeno pravilo oblika $x \to x$). Metoda vraća konačni niz znakova.

Konačno, metoda renderScene() stvara početni niz (postavlja ga na aksiom), poziva metodu lSustav (...) i potom crta prikaz koji odgovara tumačenju generiranog niza znakova preko prethodno definirane tablice značenja simbola. Ovaj programski isječak još koristi i nešto pomoćnog koda vezanog uz strukturu CharSequence koji ovdje nije prikazan i ostavlja se čitatelju da ga implementira za vježbu. Navedena struktura kao i korištene metode omogućavaju nam rad sa znakovnim nizom koji sam povećava svoj kapacitet onom dinamikom kojom mu nadodajemo znakove. Na tu se strukturu može gledati kao na polje koje se samo povećava kako mu nadodajemo elemente – pa je stoga vrlo zgodno za uporabu u L-sustavima.

```
1  #define PI (3.141592653589793)
2
3  char *RULE1 = "F>F+F--F+F";
4  char *rules[] = {RULE1,NULL};
5
6  char *pronadiPravilo(char simbol) {
7      int i = 0;
8      while(true) {
9          if(rules[i]==NULL) return NULL;
10         if(rules[i][0]==simbol) return rules[i]+2;
11         i++;
12     }
13 }
14
15 void moveTurtle(Turtle *turtle, double pomak) {
16     turtle->x += pomak * cos(turtle->angle*PI/180.0);
17     turtle->y += pomak * sin(turtle->angle*PI/180.0);
18 }
19
20 CharSequence *lSustav(CharSequence *axiom, int dubina) {
21     CharSequence *niz = copyOfCharSequence(axiom);
22     for(int i = 0; i < dubina; i++) {
23         CharSequence *novi = allocCharSequence();
24         for(int i = 0; i < niz->n; i++) {
25             char simbol = niz->data[i];
26             char *pravilo = pronadiPravilo(simbol);
27             if(pravilo==NULL) {
28                 charSequenceAdd(novi, simbol);
29             } else {
30                 charSequenceAddString(novi, pravilo);
31             }
32         }
33         freeCharSequence(niz);
34         niz = novi;
35     }
36     return niz;
37 }
38
39 void renderScene() {
40
41     int dubina = 5;
42
43     CharSequence *axiom = allocCharSequenceFromString("F");
44     CharSequence *niz = lSustav(axiom, dubina);
45
46     double pomak = 500 * pow(1.0/3.0,dubina);
47     double angle = 60;
48
49     Turtle turtle;
50     turtle.angle = 0;
51     turtle.x = 50;
```

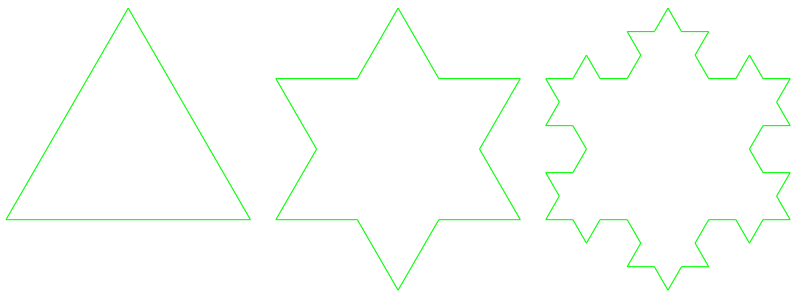
```
52      turtle.y = 150;
53
54      glColor3f(0.0f, 0.1f, 1.0f);
55      glPointSize(1);
56
57      for(int i = 0; i < niz->n; i++) {
58          glBegin(GL_LINE);
59          char simbol = niz->data[i];
60          if(simbol=='F') {
61              // zapamti staru poziciju
62              double x0 = turtle.x;
63              double y0 = turtle.y;
64              moveTurtle(&turtle, pomak);
65              // zapamti novu poziciju
66              double x1 = turtle.x;
67              double y1 = turtle.y;
68              // nacrtaj liniju stara-nova
69              glVertex2i(zaokruzi(x0), zaokruzi(y0));
70              glVertex2i(zaokruzi(x1), zaokruzi(y1));
71          } else if(simbol=='+') {
72              turtle.angle += angle;
73          } else if(simbol=='-') {
74              turtle.angle -= angle;
75          }
76          glEnd();
77      }
78
79      freeCharSequence(niz);
80      freeCharSequence(axiom);
81  }
```

Kochinu pahuljicu možemo nacrtati uporabom sljedećeg L-sustava:

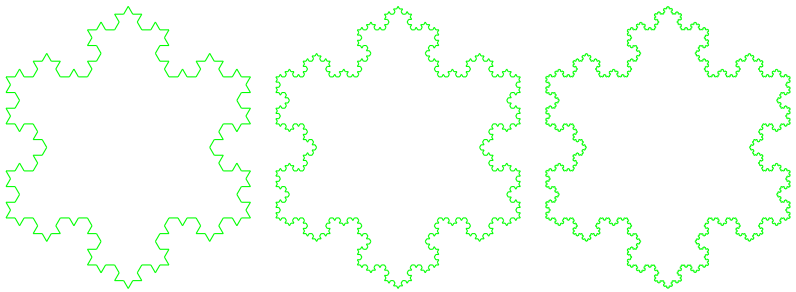
$$\mathbf{G} = (\{F, +, -\}, F--F--F, \{F \to F+F--F+F\}).$$

Uočimo da se je u odnosu na prethodni primjer promjenio samo aksiom, koji crta jednakostranični trokut. Rezultat je prikazan na slici 13.20.

Zajednička karakteristika svih dosad prikazanih L-sustava je bila generiranje neprekinute krivulje. Međutim, često to nam nije dovoljno. Primjerice, stablo je struktura koja se grana, i problem koji se tu javlja jest kojim putem krenuti kod grananja? Očit odgovor je: svim putevima – trebamo nacrtati sve grane. Međutim, kako je kornjača ograničena na jednu poziciju u jednom trenutku, to očito nije moguće direktno postići. Jedno moguće rješenje bi bilo da kornjača dolaskom na grananje negdje zapamti svoju trenutnu poziciju, i potom krene jednom granom. Nakon što granu nacrta, kornjača se teleportira na prethodno zapamćenu poziciju (u redu je, problem je složen pa je opravdana posudba *Star Trek* tehnologije), i krene u drugu granu. Kako će stablaste strukture tipično imati više od jednog grananja, potreban nam je stog kornjačinih stanja. Za rad sa stogom definirat ćemo dva nova simbola "[" i "]". Također, kornjača u svom stanju



(a) aksiom (b) dubina 1 (c) dubina 2



(d) dubina 3 (e) dubina 4 (f) dubina 5

Slika 13.20: Kochina pahuljica generirana L-sustavom

osim trenutne pozicije i smjera gledanja može pamtiti i druge parametre, poput boje kojom crta, debljine linije i slično. Definirajmo stoga dodatni simbol "w" koji će kornjači naložiti da smanji korištenu debljinu linije kojom crta za određeni postotak, te simbol "s" koji će kornjači naložiti da smanji duljinu segmenta koji crta. Evo tabličnog prikaza novih simbola.

Simbol	**Značenje**
[	Kornjača trenutno stanje zapisuje na stog.
]	Kornjača trenutno stanje restaurira sa stoga.
w	Debljinu linije potrebno je smanjiti za propisani faktor.
s	Duljinu segmenta potrebno je skratiti za propisani faktor.

Pogledajmo nekoliko primjera sustava s grananjima. Prvi sustav čija je slika prikazana na slici 13.21a definiran je kao:

$$\mathbf{G} = (\{F, X, +, -, s, w, [,], \}, FX, \{X \rightarrow sw[-FX] + FX\}).$$

pri čemu je "X" simbol koji se pri generiranju niza koristi kao nezavršni simbol a u konačnom se nizu izbacuje (odnosno nema nikakvu semantiku kako je vidljivo u programskom kodu u nastavku).

Metoda renderScene() koja prikazuje sliku koju generira ovaj sustav prikazana je u nastavku. Ujedno je i redefinirana podatkovna struktura koja pamti stanje

kornjače dodavanjem podataka o trenutnoj debljini linije te trenutnoj duljini segmenta koji se iscrtava pod djelovanjem simbola "F". Kut za koji se mijenja smjer kornjače pod djelovanjem simbola "+" i "-" je 50. Simbol "s" trenutnu duljinu segmenta množi s 0.6 dok simbol "w" trenutnu širinu linije množi s 0.7. U kodu se može vidjeti i uporaba podatkovne strukture TurtleStack – stoga na koji se stanje kornjače potiskuje svaki puta kada se naiđe na simbol "[", odnosno s kojeg se stanje vadi svaki puta kada se naiđe na simbol "]". Programski kod koji opslužuje ovu strukturu nije prikazan i ostavlja se čitateljima za vježbu.

```
1  typedef struct {
2      double x;
3      double y;
4      double angle;
5      double penSize;
6      double segmentSize;
7  } Turtle;
8
9  void renderScene() {
10
11     int dubina = 7;
12
13     CharSequence *axiom = allocCharSequenceFromString("FX");
14     CharSequence *niz = lSustav(axiom, dubina);
15
16     double angle = 50;
17
18     double penFactor = 0.7;
19     double segmentFactor = 0.6;
20
21     Turtle turtle;
22     turtle.angle = 90;
23     turtle.x = 300;
24     turtle.y = 10;
25     turtle.penSize = 10;
26     turtle.segmentSize = 220;
27
28     glColor3f(0.0f, 0.1f, 1.0f);
29     glPointSize(1);
30
31     TurtleStack *stack = newTurtleStack();
32     for(int i = 0; i < niz->n; i++) {
33         char simbol = niz->data[i];
34         if(simbol=='F') {
35             glLineWidth((float)turtle.penSize);
36             glBegin(GL_LINE);
37             double x0 = turtle.x;
38             double y0 = turtle.y;
39             moveTurtle(&turtle, turtle.segmentSize);
40             double x1 = turtle.x;
41             double y1 = turtle.y;
```

```
42                glVertex2i(zaokruzi(x0), zaokruzi(y0));
43                glVertex2i(zaokruzi(x1), zaokruzi(y1));
44                glEnd();
45            } else if(simbol=='+') {
46                turtle.angle += angle;
47            } else if(simbol=='-') {
48                turtle.angle -= angle;
49            } else if(simbol=='s') {
50                turtle.segmentSize =
51                    max(1.0, turtle.segmentSize*segmentFactor);
52            } else if(simbol=='w') {
53                turtle.penSize = max(1.0, turtle.penSize*penFactor);
54            } else if(simbol=='[') {
55                pushTurtleStack(stack, &turtle);
56            } else if(simbol==']') {
57                popTurtleStack(stack, &turtle);
58            }
59        }
60        freeTurtleStack(stack);
61
62        freeCharSequence(niz);
63        freeCharSequence(axiom);
64    }
```

L-sustav čija je slika prikazana na slici 13.21b definiran je kao:

$$\mathbf{G} = (\{F, +, -, [,]\}, F, \{F \to FF - [-F + F + F] + [+F - F - F]\}).$$

Kut za koji se mijenja smjer kornjače pod djelovanjem simbola "+" i "-" je 22.5.

L-sustav čija je slika prikazana na slici 13.21c definiran je kao:

$$\mathbf{G} = (\{F, X, Y, +, -, s, w, [,]\}, FX, \{X \to sw[-FY]+FX, Y \to FX+FY-FX\}).$$

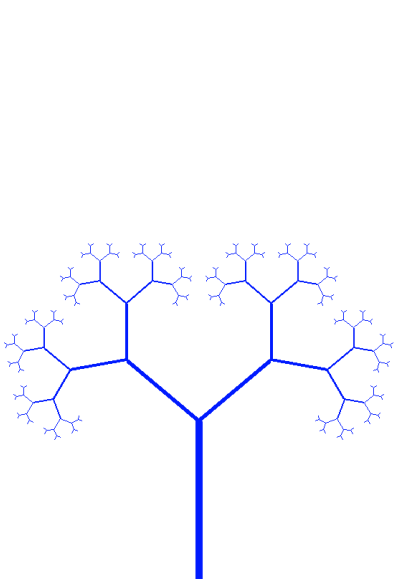
Kut za koji se mijenja smjer kornjače pod djelovanjem simbola "+" i "-" je 30. Simbol "s" trenutnu duljinu segmenta množi s 0.65 dok simbol "w" trenutnu širinu linije množi s 0.7.

L-sustav čija je slika prikazana na slici 13.21d definiran je kao:

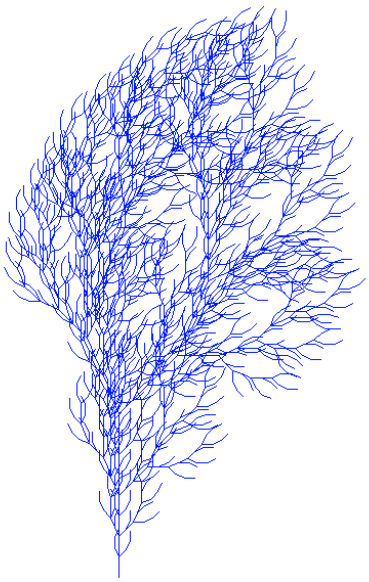
$$\mathbf{G} = (\{F, +, -, [,]\}, -F, \{F \to F[+F]F[-F][F]\}).$$

Kut za koji se mijenja smjer kornjače pod djelovanjem simbola "+" i "-" je 20.

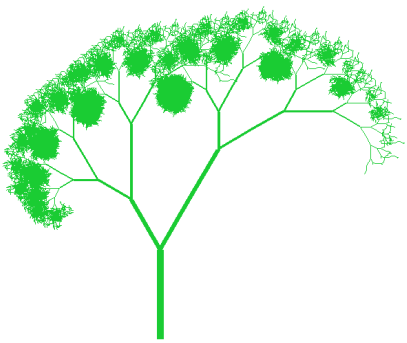
L-sustavi, općenito govoreći, mogu biti puno kompleksniji od ovdje opisanih. Primjerice, jedna modifikacija su stohastički L-sustavi, kod kojih za pojedine simbole može biti definirano više produkcija te svaka produkcija može imati vjerojatnost da će baš ona biti odabrana; u tom slučaju, simbol se zamjenjuje prema produkciji koja se odabere posredstvom slučajnog mehanizma i pridijeljenih vjerojatnosti. Druga modifikacija jest uporaba kontekstno ovisnih produkcija, koje



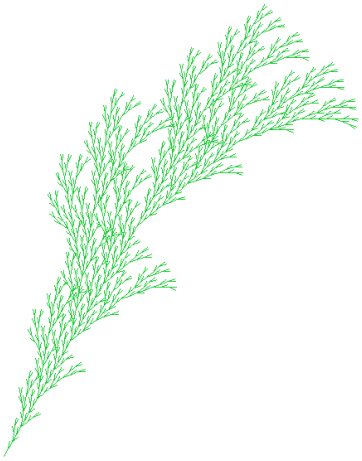
(a) Biljka 1



(b) Biljka 2



(c) Biljka 3



(d) Biljka 4

Slika 13.21: L-sustavom s grananjem

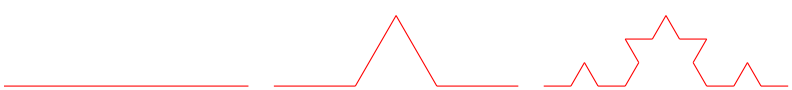
s lijeve strane nemaju samo simbol X već mogu imati i niz α koji se mora nalaziti prije tog simbola i niz β koji se mora nalaziti nakon tog simbola. Tada će produkcija "paliti" i obaviti zamjenu samo ako se u promatranom nizu neposredno lijevo od simbola X nalazi niz α te ako se od njega neposredno desno nalazi niz β. Također, osim što mogu generirati 2D slike, L-sustavi mogu raditi i u 3D prostoru generirajući različite trodimenzionalne objekte. U ovom poglavlju stoga je predstavljena ideja L-sustava, te je dan samo osnovni pregled DOL-sustava.

13.7 Opseg i površina fraktala. Fraktalna dimenzija.

Započnimo najprije s izračunom opsega Kochine krivulje. Neka je početni segment Kochine krivulje segment linije duljine a (slika 13.22). Taj segment mijenjamo s 4 segmenta, svaki duljine $a/3$, pa je nakon prvog koraka duljina tako dobivene krivulje $4 \cdot (a/3) = a \cdot \frac{4}{3}$. Promotrimo sada jedan mali segment te krivulje. U sljedećem koraku taj segment duljine $a/3$ zamijenit ćemo s nova 4 segmenta čija je pojedinačna duljina trećina promatrane: $(a/3)/3$. Kako ih ima 4, segment duljine $a/3$ zamijenili smo konstrukcijom duljine $4 \cdot \frac{a}{9}$. Krivulja je imala 4 segmenta duljine $a/3$, pa je duljina krivulje nakon drugog koraka jednaka $4 \cdot 4 \cdot \frac{a}{9} = a \cdot \left(\frac{4}{3}\right)^2$. Sada možemo uočiti uzorak koji se pojavljuje:

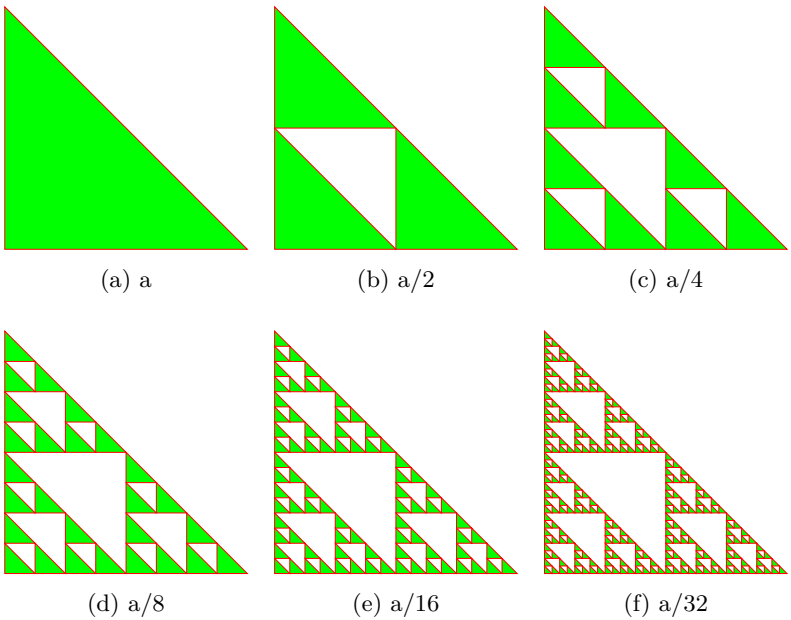
$$a \;\rightarrow\; a \cdot \frac{4}{3} \;\rightarrow\; a \cdot \left(\frac{4}{3}\right)^2 \;\rightarrow\; a \cdot \left(\frac{4}{3}\right)^3 \;\rightarrow\; \ldots \;\rightarrow\; a \cdot \left(\frac{4}{3}\right)^{\infty} = \infty$$

Možemo dakle zaključiti da je duljina Kochine krivulje doista beskonačna. Pogledajmo sada primjer fraktalnog lika – trokut Sierpinskog, pa izračunajmo njegov opseg i površinu (slika 13.23). Krenimo od jednakokračnog pravokutnog trokuta čiji je krak duljine a. Opseg tog trokuta je $(2 + \sqrt{2}) \cdot a$, površina $a^2/2$. Početni trokut sada zamijenimo s tri pravokutna jednakokračna trokuta čija je duljina kraka jednaka polovici početne duljine kraka. Ukupni opseg takvog lika je suma triju opsega manjih trokuta: $(2 + \sqrt{2}) \cdot a \cdot \frac{3}{2}$. Površina je jednaka sumi površina triju manjih trokuta: $\left(\frac{a}{2}\right)^2/2 + \left(\frac{a}{2}\right)^2/2 + \left(\frac{a}{2}\right)^2/2 = \frac{a^2}{2} \cdot \frac{3}{4}$. U sljedećem koraku konstrukcije svaki bi trokut kraka $a/2$ bio zamijenjen s 3 trokuta upola manjeg kraka, dakle s $a/4$. Time bi ukupni opseg tog lika bio $(2 + \sqrt{2}) \cdot a \cdot \left(\frac{3}{2}\right)^2$,



(a) a (b) a/3 (c) a/9

Slika 13.22: Opseg Kochine krivulje



(a) a (b) a/2 (c) a/4

(d) a/8 (e) a/16 (f) a/32

Slika 13.23: Površina i opseg trokuta Sierpinskog

a ukupna površina $\frac{a^2}{2} \cdot \left(\frac{3}{4}\right)^2$. Opet možemo uočiti uzorak koji se pojavljuje:

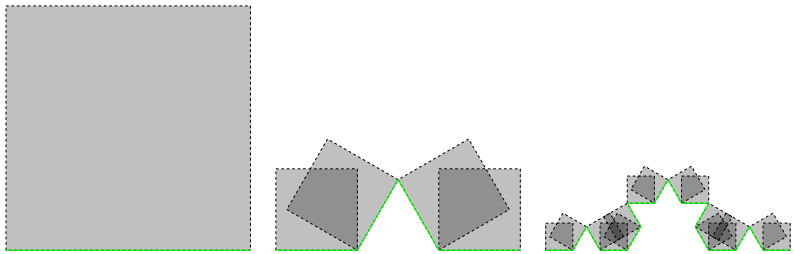
$$(2+\sqrt{2})\cdot a \to (2+\sqrt{2})\cdot a\cdot\tfrac{3}{2} \to (2+\sqrt{2})\cdot a\cdot\left(\tfrac{3}{2}\right)^2 \to (2+\sqrt{2})\cdot a\cdot\left(\tfrac{3}{2}\right)^3 \to \ldots \to (2+\sqrt{2})\cdot a\cdot\left(\tfrac{3}{2}\right)^\infty = \infty$$

$$a^2/2 \;\to\; a^2/2\cdot\frac{3}{4} \;\to\; a^2/2\cdot\left(\frac{3}{4}\right)^2 \;\to\; a^2/2\cdot\left(\frac{3}{4}\right)^3 \;\to\; \ldots \;\to\; a^2/2\cdot\left(\frac{3}{4}\right)^\infty = 0$$

Trokut Sierpinskog ima dakle beskonačan ukupni opseg te površinu jednaku 0.

Pogledajmo malo rezultate prethodnih izračuna. Krenimo s Kochinom krivuljom. Promotrimo skup točaka koji čini tu krivulju. Zanima nas koje je dimenzionalnosti taj objekt? Je li to 1D objekt? Kada bi bio, teško bismo mogli objasniti kako to da je njegova duljina beskonačna – a opet, objekt je smješten na konačnoj površini! Možda je to dvodimenzionalni objekt? U tom slučaju, mogli bismo pokušati procijeniti njegovu površinu; dapače, ne točnu, već barem naći nekakvu gornju ogradu. Evo eksperimenta. Aproksimirajmo površinu ovog objekta sumom površina kvadrata koje postavimo iznad svakog segmenta krivulje (slika 13.24). Ovo će očito biti vrlo gruba procjena, i puno prevelika. No, rezultat će biti zanimljiv.

Konstrukciju Kochine krivulje započinjemo jednim segmentom linije duljine a. U ovom koraku, kako imamo samo jedan segment, površinu aproksimiramo površinom pripadnog kvadrata: a^2. Napravimo sada jedan korak izgradnje krivulje: liniju zamijenimo s 4 segmenta, svaki duljine $a/3$. Nad njima možemo podignuti



(a) a (b) a/2 (c) a/4

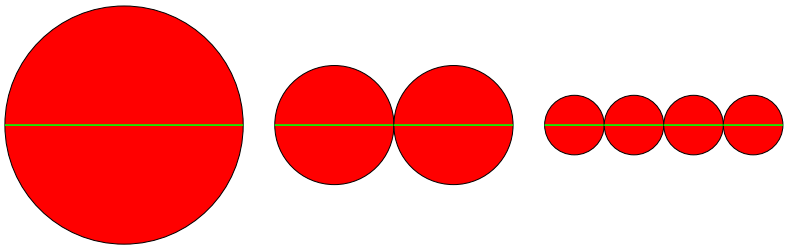
Slika 13.24: Površina Kochine krivulje

4 kvadrata stranice $a/3$, pa je suma njihovih površina jednaka $4 \cdot (a/3)^2 = a^2 \cdot \frac{4}{3^2}$. Napravimo li sada sljedeći korak u konstrukciji, svaki segment duljine $a/3$ zamijenit ćemo s 4 segmenta duljine $(a/3)/3$. Suma površina svih kvadrata nad tim segmentima bit će tada: $a^2 \cdot \frac{4^2}{3^4}$. Sljedeći će korak dati površinu od $a^2 \cdot \frac{4^3}{3^6}$. Iz ovoga je lagano uočiti da je procjena površine u koraku n dana formulom: $a^2 \cdot \frac{4^n}{3^{(2n)}}$ što je zapravo $a^2 \cdot \left(\frac{4}{9}\right)^n$. Kada n pustimo da teži u beskonačnost, površina teži k nuli. Iz ovog eksperimenta možemo zaključiti da objekt nije niti dvodimenzionalan – nema površine. Stoga je zaključak da je dimenzija ove krivulje negdje između 1 i 2.

Fraktalne dimenzije nastale su kao pokušaj da se definira dimenzionalnost ovakvih objekata. Danas postoji više načina kako je moguće definirati fraktalne dimenzije. Jedan od matematički najpoznatijih jest Hausdorffova dimenzija koju nije baš najjednostavnije izravno računati prema definiciji. Stoga ćemo u nastavku pokazati dva druga načina kako je moguće odrediti dimenzionalnost fraktala, a koji se temelje na metodi prebrojavanja kutija (engl. *Box-Counting Method*). Vidjet ćemo u nastavku da se ovaj pojam "kutija" može interpretirati na različite načine, pa može doista predstavljati hiperkocke, hiperkugle i slično.

Ideja je sljedeća: promatrani skup točaka koji čini promatrani objekt potrebno je prekriti najmanjim brojem kugli radijusa r. Očito je da ako smanjujemo radijus r, trebat ćemo više kugli; označimo najmanji broj kugli radijusa r potreban za prekrivanje objekta Θ s $N_\Theta(r)$. Dimenzija dobivena metodom prebrojavanja kutija tada je definirana kao limes od negativnog omjera logaritma potrebnog broja kugli i logaritma radijusa tih kugli, kada radijus teži ka nuli. Pri tome pojam *kugla* treba shvatiti u kontekstu n-dimenzionalnih hiperkugli, što primjerice za $n = 2$ odgovara krugu. Možemo pisati:

$$d_H(\Theta) = -\lim_{r \to 0} \frac{\log\ N_\Theta(r)}{\log\ r} \tag{13.7}$$



(a) $1 \times a$, $r = a/2$ (b) $2 \times a/2$, $r = a/4$ (c) $4 \times a/4$, $r = a/8$

Slika 13.25: Određivanje dimenzije segmenta linije

Ovako dobivena dimenzija odgovara našem uobičajenom poimanju dimenzije. Pogledajmo to na dva primjera.

Primjer: 14

Neka je kao objekt zadan segment linije. Odredite dimenziju dobivenu metodom prebrojavanja kutija tog objekta.

Rješenje:

Neka je duljina segment linije jednaka a (slika 13.25). Čitav segment moguće je obuhvatiti jednim krugom čiji je centar u središtu segmenta, a radijus $a/2$. Međutim, treba vidjeti što se događa ako smanjujemo r. Podijelimo stoga naš segment na dva jednako dugačka dijela: lijevi i desni. Svaki od njih ima duljinu $a/2$. Svaki segment moguće je obuhvatiti jednim krugom radijusa $a/4$ čiji je centar smješten u središtu tog segmenta. Broj takvih krugova potreban da se prekrije čitav objekt bit će 2. Ako nastavimo sa smanjivanjem, početnu liniju možemo rekurzivno podijeliti na 4 jednaka dijela. Tada će nam trebati 4 kruga radijusa $a/8$, kako bismo prekrili čitav objekt. Trend je prikazan u sljedećoj tablici:

Dubina rekurzije	0	1	2	...	n
Broj krugova H_Θ	1	2	4	...	2^n
Radijus kruga r	$a/2$	$a/4$	$a/8$	...	$\frac{a}{2^{n+1}}$

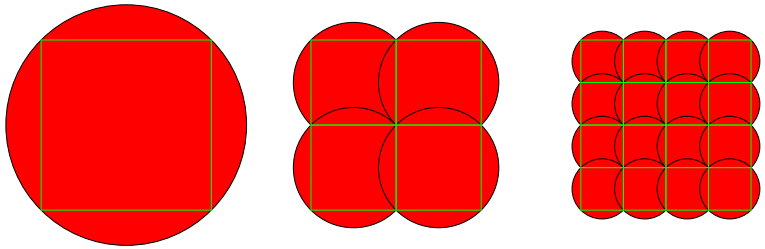
Očito je da puštanjem $n \to \infty$ radijus r teži ka nuli. Pogledajmo stoga kamo teži omjer broja krugova i radijusa:

$$\begin{aligned} d_H(\Theta) &= -\lim_{r\to 0} \frac{\log\ N_\Theta(r)}{\log\ r} \\ &= -\lim_{n\to\infty} \frac{\log\ 2^n}{\log\ \frac{a}{2^{n+1}}} \\ &= -\lim_{n\to\infty} \frac{n \cdot \log\ 2}{\log\ \frac{a}{2} - n\ \log\ 2} \\ &= 1 \end{aligned}$$

Zaključak: segment linije je jednodimenzionalan objekt, što je u skladu s našim očekivanjima.

Primjer: 15

Neka je kao objekt zadan kvadrat. Odredite dimenziju dobivenu metodom prebrojavanja kutija tog objekta.



(a) $1 \times a$, $r = \frac{a}{2}\sqrt{2}$ (b) $4 \times a/2$, $r = \frac{a}{4}\sqrt{2}$ (c) $16 \times a/4$, $r = \frac{a}{8}\sqrt{2}$

Slika 13.26: Određivanje dimenzije segmenta kvadrata

Rješenje:

Neka je duljina stranice kvadrata jednaka a (slika 13.26). Čitav kvadrat moguće je obuhvatiti jednim krugom čiji je centar u središtu kvadrata, a radijus $a/\sqrt{2}$. Podijelimo sada kvadrat kvadrate čija je stranica $a/2$. Početni kvadrat ovime će se raspasti na 4 manja kvadrata. Svaki od njih ima stranicu $a/2$, pa ćemo ga moći obuhvatiti u krug radijusa $\frac{a/2}{\sqrt{2}}$. Za pokriti čitav lik trebat ćemo 4 kruga. Ako postupak nastavimo tako da svaki od manjih kvadrata rekurzivno podijelimo, dobit ćemo još manje kvadrate duljine stranice $a/4$. Čitav lik tada će se sastojati od $4 \cdot 4 = 16$ manjih kvadrata, a svaki ćemo moći obuhvatiti jednim krugom radijusa $\frac{a/4}{\sqrt{2}}$. Trend je prikazan u sljedećoj tablici:

Dubina rekurzije	0	1	2	...	n
Broj krugova H_Θ	1	4	16	...	4^n
Radijus kruga r	$a/\sqrt{2}$	$\frac{a/2}{\sqrt{2}}$	$\frac{a/4}{\sqrt{2}}$	...	$\frac{a}{2^n\sqrt{2}}$

Očito je da puštanjem $n \to \infty$ radijus r teži ka nuli. Pogledajmo stoga kamo teži omjer broja krugova i radijusa:

$$\begin{aligned} d_H(\Theta) &= -\lim_{r\to 0} \frac{\log\ N_\Theta(r)}{\log\ r} \\ &= -\lim_{n\to\infty} \frac{\log\ 4^n}{\log\ \frac{a}{2^n\sqrt{2}}} \\ &= -\lim_{n\to\infty} \frac{2n \cdot \log\ 2}{\log\ \frac{a}{\sqrt{2}} - n\ \log\ 2} \\ &= 2 \end{aligned}$$

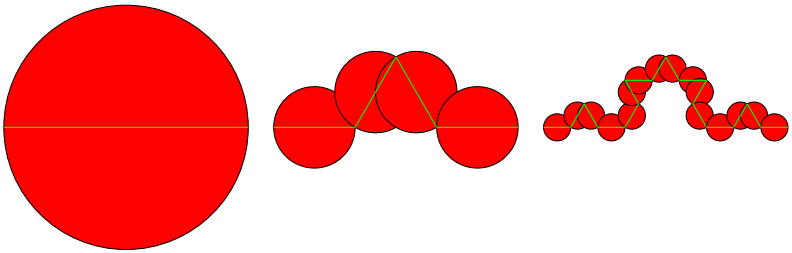
Zaključak: kvadrat je dvodimenzionalan objekt, što je opet u skladu s našim očekivanjima.

Nakon što smo se uvjerili da ovako dobivena dimenzija doista odgovara onome što smo i očekivali za 1D i 2D objekte, pokušajmo sada na isti način izračunati dimenziju dobivenu metodom prebrojavanja kutija Kochine krivulje kao jednog od najjednostavnijih samosličnih fraktala.

Primjer: 16

Neka je kao objekt zadana Kochina krivulja. Odredite dimenziju dobivenu metodom prebrojavanja kutija tog objekta.

Rješenje:



(a) a (b) a/3 (c) a/9

Slika 13.27: Određivanje dimenzije Kochine krivulje

Neka je duljina početnog segment linije iz kojeg kreće konstrukcija jednaka a (slika 13.27). Čitav segment moguće je obuhvatiti jednim krugom čiji je centar u središtu segmenta, a radijus $a/2$. Napravimo li sljedeći korak konstrukcije, jedan segment duljine a zamijenit ćemo s 4 segmenta, svaki duljine $a/3$. Taj objekt moći ćemo obuhvatiti s 4 kruga, svaki radijusa $a/6$. Napravimo li sljedeći korak konstrukcije, jedan segment duljine $a/3$ zamijenit ćemo s 4 segmenta, svaki duljine $a/9$. Čitav objekt tada ćemo moći obuhvatiti s $4 \cdot 4 = 16$ krugova, svaki radijusa $a/18$. Nastavimo li dalje, trend je prikazan u sljedećoj tablici:

Dubina rekurzije	0	1	2	...	n
Broj krugova H_Θ	1	4	16	...	4^n
Radijus kruga r	$a/2$	$\frac{a/2}{3}$	$\frac{a/2}{9}$	...	$\frac{a}{2 \cdot 3^n}$

Očito je da puštanjem $n \to \infty$ radijus r teži ka nuli. Pogledajmo stoga kamo teži omjer broja krugova i radijusa:

$$\begin{aligned} d_H(\Theta) &= -\lim_{r \to 0} \frac{\log\ N_\Theta(r)}{\log\ r} \\ &= -\lim_{n \to \infty} \frac{\log\ 4^n}{\log\ \frac{a}{2 \cdot 3^n}} \\ &= -\lim_{n \to \infty} \frac{n \cdot \log\ 4}{\log\ \frac{a}{2} - n\ \log\ 3} \\ &= \frac{\log\ 4}{\log\ 3} \\ &= 1.2618595... \end{aligned}$$

Zaključak: dimenzija dobivena metodom prebrojavanja kutija Kochine krivulje je poprilici 1.26, što je između 1 i 2, baš kao što smo to i očekivali.

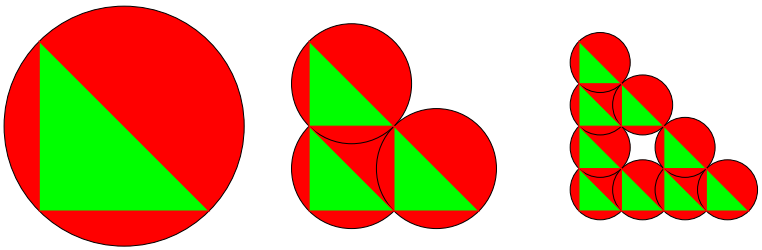
Izračunajmo i dimenziju trokuta Sierpinskog.

Primjer: 17

Neka je kao objekt zadan pravokutni jednakokračni trokut Sierpinskog. Odredite dimenziju dobivenu metodom prebrojavanja kutija tog objekta.

Rješenje:

Neka je duljina stranice početnog pravokutnog jednakokračnog trokuta iz kojeg kreće konstrukcija jednaka a (hipotenuza je tada $a\sqrt{2}$) – slika 13.28. Čitav trokut moguće je obuhvatiti jednim krugom čiji



(a) a (b) a/2 (c) a/4

Slika 13.28: Određivanje dimenzije trokuta Sierpinskog

je centar u središtu hipotenuze trokuta, a radijus $\frac{a\sqrt{2}}{2}$. Napravimo li sljedeći korak konstrukcije, jedan trokut kraka a zamijenit ćemo s 3 trokuta, svaki kraka $a/2$. Taj objekt moći ćemo obuhvatiti s 3 kruga, svaki radijusa $\frac{a\sqrt{2}}{4}$. Napravimo li sljedeći korak konstrukcije, svaki trokut kraka duljine $a/2$ zamijenit ćemo s 3 nova trokuta, svaki duljine kraka $a/4$. Čitav objekt tada ćemo moći obuhvatiti s 9 krugova, svaki radijusa $\frac{a\sqrt{2}}{8}$. Nastavimo li dalje, trend je prikazan u sljedećoj tablici:

Dubina rekurzije	0	1	2	...	n
Broj kugli H_Θ	1	3	9	...	3^n
Radijus kugle r	$\frac{a\sqrt{2}}{2}$	$\frac{a\sqrt{2}}{2\cdot 2}$	$\frac{a\sqrt{2}}{2\cdot 4}$	...	$\frac{a\sqrt{2}}{2\cdot 2^n}$

Očito je da puštanjem $n \to \infty$ radijus r teži ka nuli. Pogledajmo stoga kamo teži omjer broja kugli i radijusa:

$$\begin{aligned} d_H(\Theta) &= -\lim_{r\to 0} \frac{\log\ N_\Theta(r)}{\log\ r} \\ &= -\lim_{n\to\infty} \frac{\log\ 3^n}{\log\ \frac{a\sqrt{2}}{2\cdot 2^n}} \\ &= -\lim_{n\to\infty} \frac{n\cdot \log\ 3}{\log\ \frac{a\sqrt{2}}{2} - n\ \log\ 2} \\ &= \frac{\log\ 3}{\log\ 2} \\ &= 1.5849625... \end{aligned}$$

Zaključak: dimenzija dobivena metodom prebrojavanja kutija trokuta Sierpinskog iznosi poprilici 1.58, što je između 1 i 2.

Izračunajte za vježbu dimenziju dobivenu metodom prebrojavanja kutija tepiha Sierpinskog, prikazanog na slici 13.16.

Umjesto da za prekrivanje fraktalnog objekta koristimo hiperkugle, možemo koristiti i hiperkocke. Prilikom izračuna dimenzije na ovaj način, objekt se ne prekriva krugovima, već se prekriva kvadratnom rešetkom. Neka su dimenzije jednog kvadratića takve rešetke $\epsilon \times \epsilon$. Potrebno je izbrojati preko koliko se kvadratića proteže fraktal, ili rečeno drugačije, koliko nam kvadratića treba da bismo prekrili fraktal. Potom pustimo duljinu stranice da teži ka nuli (rešetka postaje sve finija i finija), i gledamo limes omjera potrebnog broja kvadratića i stranice kvadratića.

Konkretno, ovako definiranu dimenziju nekog skupa S možemo računati kao:

$$d_M(S) = -\lim_{\epsilon \to 0} \frac{\log\ N_S(\epsilon)}{\log\ \epsilon} = \lim_{\epsilon \to 0} \frac{\log\ N_S(\epsilon)}{\log\ \frac{1}{\epsilon}} \tag{13.8}$$

Za korektan izračun ove mjere nije nužno da kvadratići budu pravilno raspoređeni u rešetki. Umjesto toga, moguće je naprosto koristiti potreban broj kvadratića (koji mogu biti i zarotirani) i ručno ih rasporediti tako da pokriju čitav fraktal.

Za vježbu se uvjerite da i ovakav izračun fraktalne dimenzije za Kochinu krivulju daje isti broj kao i kada se dimenzija računa prekrivanjem hiperkuglama što smo prethodno već izračunali (naputak: za pokrivanje koristite četiri pravokutnika – dva "normalna" i dva zarotirana).

13.8 Ponavljanje

1. Što su fraktali?
2. Kako fraktale možemo podijeliti s obzirom na način generiranja?
3. Kako izgleda Kochina krivulja?
4. Kako se generira Mandelbrotov fraktal?
5. Koja je razlika između Mandelbrotovog fraktala i Mandelbrotovog skupa?
6. Na koji se način može obaviti bojanje Mandelbrotovog fraktala?
7. Kako se generira Julijev fraktal? U čemu je razlika između Julijevog i Mandelbrotovog fraktala?
8. Što su i kako se crtaju IFS-fraktali?
9. Kako se zadaje IFS-fraktal? Kako se određuju parametri transformacija za Trokut Sierpinskog odnosno Tepih Sierpinskog?
10. Što su Lindermayerovi sustavi (L-sustavi)? Kako se prikazuju te na koji se način osigurava konstantna veličina prikazanog fraktala s obzirom na dubinu rekurzije?
11. Koji ćemo fraktal dobiti za sljedeći L-sustav: G=(F,+,−,F,F→F+F−−F+F)?
12. Kako se određuju opseg i površina fraktala?
13. Što je fraktalna dimenzija i na koja dva načina smo je računali?

Dodatak A

Dodatno o transformacijama pogleda

A.1 View-up vektor

View-up vektor popularan je naziv za jedan od načina "kroćenja" svih osi sustava (dakle osi x' i y') i njegova uporaba već je opisana u potpoglavlju 6.3.4. Ovdje dajemo način izračuna točke K koji se temelji na skalarnom produktu.

Da bi definirao koordinatni sustav, korisnik može zadati različite kombinacije parametara. Čest slučaj je da se zada očište O i gledište G, informacija da se gradi lijevi koordinatni sustav te da se zada vektor $\vec{v}_{up}$ koji približno pokazuje smjer osi y. Os z koordinatnog sustava određena je vektorom $\vec{h}$:

$$\vec{h} = G - O.$$

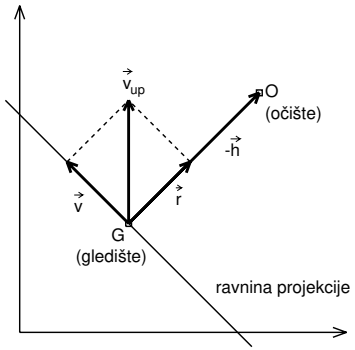
Time je definirano da je pozitivan smjer osi z od ishodišta prema gledištu. Os y mora biti okomita na tu os. Označimo s $\vec{v}$ vektor koji pokazuje u smjeru pozitivne osi y. Taj vektor tada leži u ravnini kojoj pripada očište i čiji je vektor normale upravo $\vec{h}$. Zadavanje baš takvog vektora za korisnike će predstavljati problem. Umjesto toga, dopustit ćemo korisniku da zada vektor $\vec{v}_{up}$ koji je "poprilici" u smjeru osi y a mi ćemo računski pronaći vektor $\vec{v}$ koji je približno u tom smjeru i koji stvarno leži u traženoj ravnini.

Ilustracija koja će nam pomoći prikazana je na slici A.1. Kako vektori u prostoru nisu vezani za točke, ravnina i vektor $\vec{v}_{up}$ pomaknuti su u točku G. Obratite pažnju da je na slici prikazan vektor $-\vec{h}$ (uz smjer koji je nacrtan).

Uočite da vrijedi:

$$\vec{v}_{up} = \vec{r} + \vec{v}.$$

Kut između vektora $-\vec{h}$ i vektora $\vec{v}_{up}$ definiran je izrazom:



Slika A.1: Pronalaženje y-osi temeljem *view-up* vektora

$$\cos(-\vec{h}, \vec{v}_{up}) = \frac{-\vec{h} \cdot \vec{v}_{up}}{||\vec{h}|| \cdot ||\vec{v}_{up}||}.$$

Norma vektora $\vec{r}$ slijedi iz pravokutnog trokuta:

$$||\vec{r}|| = ||\vec{v}_{up}|| \cdot \cos(-\vec{h}, \vec{v}_{up}) = ||\vec{v}_{up}|| \cdot \frac{-\vec{h} \cdot \vec{v}_{up}}{||\vec{h}|| \cdot ||\vec{v}_{up}||} = \frac{-\vec{h} \cdot \vec{v}_{up}}{||\vec{h}||},$$

dok je vektor $\vec{r}$ jednak svojoj normi puta jedinični vektor u smjeru $-\vec{h}$:

$$\vec{r} = ||\vec{r}|| \cdot \frac{-\vec{h}}{||\vec{h}||} = \frac{-\vec{h} \cdot \vec{v}_{up}}{||\vec{h}||} \cdot \frac{-\vec{h}}{||\vec{h}||} = \frac{\vec{h} \cdot \vec{v}_{up}}{||\vec{h}||^2} \cdot \vec{h}.$$

Sada iz jednakosti $\vec{v}_{up} = \vec{r} + \vec{v}$ možemo izvući traženi vektor $\vec{v}$:

$$\vec{v} = \vec{v}_{up} - \vec{r} = \vec{v}_{up} - \frac{\vec{h} \cdot \vec{v}_{up}}{||\vec{h}||^2} \cdot \vec{h}.$$

Izraz se da još pojednostavniti ako se vektor $\vec{h}$ ne definira kao $G - O$ već kao normirani vektor:

$$\vec{h} = \frac{G - O}{||G - O||}.$$

U tom slučaju izraz za $\vec{v}$ se pojednostavljuje u:

$$\vec{v} = \vec{v}_{up} - (\vec{h} \cdot \vec{v}_{up}) \cdot \vec{h}.$$

Koristeći dobiveni izraz točka K koju smo uz O i G koristili za zadavanje koordinatnog sustava kod transformacija pogleda može se izračunati trivijalno:

$$K = G + \vec{v}.$$

Nakon izračunavanja točke K može se krenuti u izgradnju matrica za opću perspektivnu projekciju koristeći pri tome matricu $\mathbf{\Theta}_4$. Uočimo još jednom: prilikom transformacija pogleda i projekcija, *view-up* vektor je vektor koji zadaje korisnik kako bi definirao smjer y-osi lokalnog 2D koordinatnog sustava razapetog u ravnini projekciji s ishodištem u gledištu. Tipično, *view-up* vektor ne leži u ravnini projekcije već "poprilici" pokazuje u smjeru osi y; ravnina projekcije uvijek je okomita na spojnicu očište-gledište. Projekcija *view-up* vektora u ravninu projekcije daje vektor kolinearan jediničnom vektoru y-osi.

A.2 Transformacija pogleda

Zadana su dva koordinatna sustava: osnovni s ishodištem u $O = (0, 0, 0)$ i jediničnim vektorima $\vec{i}$, $\vec{j}$ i $\vec{k}$ te drugi koordinatni sustav čije je ishodište zadano s $O' = (O'_x, O'_y O'_z)$ i jediničnim vektorima $\vec{u}$, $\vec{v}$ i $\vec{w}$. Pretpostavit ćemo podatke o drugom koordinatnom sustavu znamo u terminima podataka o prvom koordinatnom sustavu, odnosno da vrijedi:

$$O' = O'_x \cdot \vec{i} + O'_y \cdot \vec{j} + O'_z \cdot \vec{k},$$

$$\vec{u} = u_x \cdot \vec{i} + u_y \cdot \vec{j} + u_z \cdot \vec{k},$$
$$\vec{v} = v_x \cdot \vec{i} + v_y \cdot \vec{j} + v_z \cdot \vec{k},$$
$$\vec{w} = w_x \cdot \vec{i} + w_y \cdot \vec{j} + w_z \cdot \vec{k}.$$

Pretpostavimo da u koordinatnom sustavu $\vec{u}$-$\vec{v}$-$\vec{w}$ znamo koordinate točke $T' = (x', y', z')$. Koordinate te iste točke u osnovnom koordinatnom sustavu su:

$$\vec{T} = \vec{O'} + x' \cdot \vec{u} + y' \cdot \vec{v} + z' \cdot \vec{w} \tag{A.1}$$

Za potrebe nastavka izvoda usvojit ćemo konvenciju da točke i vektore gledamo kao jednostupčaste matrice. Izraz (A.1) možemo prikazati kao:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix} + \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \tag{A.2}$$

Ako su nam koordinate točke poznate u osnovnom koordinatnom sustavu, koordinate te iste točke u sustavu $\vec{u}$-$\vec{v}$-$\vec{w}$ možemo dobiti tako da izraz (A.2) preuredimo na sljedeći način. Najprije prebacimo koordinate ishodišta s lijeve strane pa još jednom okrenimo izraz:

$$\begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix}.$$

Potom čitav izraz pomnožimo s lijeve strane inverzom matrice vektora koordinatnih osi. S lijeve strane time će se matrica poništiti pa ostaje:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}^{-1} \cdot \left\{ \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix} \right\}. \tag{A.3}$$

Da bismo dobili koordinate točke u zadanom koordinatnom sustavu, trebamo dakle najprije točku translatirati i potom pomnožiti s matricom koja će preračunati koordinate.

Primjetite da izraz (A.3) vrijedi za potpuno općeniti sustav $\vec{u}$-$\vec{v}$-$\vec{w}$ čiji vektori koordinatnih osi mogu biti međusobno okomiti ili ne moraju, te mogu biti jedinični ili ne moraju. Međutim, u tom skroz općenitom slučaju nužno je računati inverz matrice vektora koordinatnih osi.

Uvedemo li dodatna ograničenja, izračun inverza matrice vektora koordinatnih osi može postati trivijalan. Pa pogledajmo dva slučaja.

A.2.1 Slučaj 1: koordinatni sustav s ortonormiranom bazom

Kao prvi slučaj razmotrit ćemo slučaj koji je i najčešći: vektori $\vec{u}$, $\vec{v}$ i $\vec{w}$ su međusobno okomiti (ortogonalni) i svi su jedinični (norma im je 1). U tom slučaju matrica vektora koordinatnih osi je ortogonalna matrica, a ortogonalne matrice imaju svojstvo da je njihov inverz jednak samoj transponiranoj matrici. U tom slučaju, izraz (A.3) prelazi u:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}^{T} \cdot \left\{ \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix} \right\}$$

odnosno

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix} \cdot \left\{ \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix} \right\}. \tag{A.4}$$

A.2.2 Slučaj 2: koordinatni sustav s ortogonalnom ali nejediničnom bazom

Kao proširenje prethodnog slučaja razmotrit ćemo situaciju kada su vektori $\vec{u}$, $\vec{v}$ i $\vec{w}$ međusobno okomiti (ortogonalni) ali nisu nužno jedinični (norma im ne treba biti 1). Matricu vektora koordinatnih osi označit ćemo oznakom Γ:

$$\Gamma = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}.$$

U razmatranom slučaju, matrica Γ je matrica sastavljena od okomitih vektora ali nije ortogonalna (u smislu da vektori nisu jedinični). Označimo s $\|\vec{u}\|$, $\|\vec{v}\|$ i $\|\vec{w}\|$ norme vektora $\vec{u}$, $\vec{v}$ i $\vec{w}$. Konstruirajmo matricu Λ tako da matrici Γ sve komponente prvog stupca podijelimo s $\|\vec{u}\|$, sve komponente drugog stupca podijelimo s $\|\vec{v}\|$ te sve komponente trećeg stupca podijelimo s $\|\vec{w}\|$. Opisano skaliranje možemo postići tako da matricu Γ pomnožimo prikladnom matricom skaliranja S:

$$\begin{aligned} \Lambda &= \Gamma \cdot S \\ &= \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{\|\vec{u}\|} & 0 & 0 \\ 0 & \frac{1}{\|\vec{v}\|} & 0 \\ 0 & 0 & \frac{1}{\|\vec{w}\|} \end{bmatrix} \\ &= \begin{bmatrix} \frac{u_x}{\|\vec{u}\|} & \frac{v_x}{\|\vec{v}\|} & \frac{w_x}{\|\vec{w}\|} \\ \frac{u_y}{\|\vec{u}\|} & \frac{v_y}{\|\vec{v}\|} & \frac{w_y}{\|\vec{w}\|} \\ \frac{u_z}{\|\vec{u}\|} & \frac{v_z}{\|\vec{v}\|} & \frac{w_z}{\|\vec{w}\|} \end{bmatrix} \end{aligned}$$

Za potrebe daljnjeg razmatranja zapišimo matricu Λ kao matricu triju stupčanih vektora:

$$\Lambda = \begin{bmatrix} \vec{\lambda}_1 & \vec{\lambda}_2 & \vec{\lambda}_3 \end{bmatrix}.$$

Tvrdimo da je matrica Λ ortogonalna matrica. Doista, u toj matrici svaki je stupac jedinični vektor i svi su stupci međusobno okomiti. Primjerice, norma vektora koji je u prvom stupcu matrice Λ, dakle vektora $\vec{\lambda}_1$ je:

$$\begin{aligned}\|\vec{\lambda}_1\| &= \sqrt{\left(\frac{u_x}{\|\vec{u}\|}\right)^2 + \left(\frac{u_y}{\|\vec{u}\|}\right)^2 + \left(\frac{u_z}{\|\vec{u}\|}\right)^2} \\ &= \sqrt{\frac{u_x^2 + u_y^2 + u_z^2}{\|\vec{u}\|^2}} \\ &= \frac{\sqrt{u_x^2 + u_y^2 + u_z^2}}{\sqrt{\|\vec{u}\|^2}} \\ &= \frac{\|\vec{u}\|}{\|\vec{u}\|} \\ &= 1.\end{aligned}$$

Na jednak način možemo provjeriti istinitost tvrdnje i za preostale stupce. Provjerimo još okomitost. Skalarni produkt prvog i drugog stupca je:

$$\begin{aligned}\vec{\lambda}_1 \cdot \vec{\lambda}_2 &= \frac{u_x}{\|\vec{u}\|} \cdot \frac{v_x}{\|\vec{v}\|} + \frac{u_y}{\|\vec{u}\|} \cdot \frac{v_y}{\|\vec{v}\|} + \frac{u_z}{\|\vec{u}\|} \cdot \frac{v_z}{\|\vec{v}\|} \\ &= \frac{u_x \cdot v_x + u_y \cdot v_y + u_z \cdot v_z}{\|\vec{u}\| \cdot \|\vec{v}\|} \\ &= \frac{1}{\|\vec{u}\| \cdot \|\vec{v}\|} \cdot (\vec{u} \cdot \vec{v}) \\ &= \frac{1}{\|\vec{u}\| \cdot \|\vec{v}\|} \cdot 0 \\ &= 0.\end{aligned}$$

Skalarni produkt bilo koja dva stupca matrice Λ svodi se na skalirani skalarni produkt originalnih vektora a kako su oni okomiti, rezultat je uvijek 0.

Kako je matrica Λ ortogonalna, njezin inverz jednak je njezinoj transponiranoj matrici:

$$\Lambda^{-1} = \Lambda^T$$

pa uvrštavanjem imamo:

$$(\Gamma \cdot S)^{-1} = (\Gamma \cdot S)^T .$$

Inverz i operacija transponiranja nad umnoškom kvadratnih matrica djeluju slično: okreću redosljed matrica i na njih primjenjuju operaciju. Stoga vrijedi:

$$S^{-1} \cdot \Gamma^{-1} = S^T \cdot \Gamma^T.$$

Množenjem čitavog izraza sa S s lijeve strane dobivamo:

$$\Gamma^{-1} = S \cdot S^T \cdot \Gamma^T.$$

Kako je matrica S dijagonalna (matrica skaliranja!), ona je simetrična pa je njezina transponirana matrica jednak njoj samoj. Stoga za inverz matrice Γ dobivamo konačni izraz:

$$\Gamma^{-1} = S^2 \cdot \Gamma^T. \tag{A.5}$$

U našem slučaju, matrica S je dijagonalna matrica čiji su dijagonalni elementi recipročne norme zadanih vektora. Kvadrat dijagonalne matrice je dijagonalna matrica čiji su elementi kvadrirani a kako je kvadrat norme jednak skalarnom produktu vektora sa samim sobom, slijedi:

$$\Gamma^{-1} = \begin{bmatrix} \frac{1}{\vec{u}\cdot\vec{u}} & 0 & 0 \\ 0 & \frac{1}{\vec{v}\cdot\vec{v}} & 0 \\ 0 & 0 & \frac{1}{\vec{w}\cdot\vec{w}} \end{bmatrix} \cdot \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix}. \tag{A.6}$$

Stoga u slučaju ortogonalnih ali nejediničnih vektora transformaciju koordinata uvrštavanjem izraza (A.6) u (A.3) obavljamo:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \frac{1}{\vec{u}\cdot\vec{u}} & 0 & 0 \\ 0 & \frac{1}{\vec{v}\cdot\vec{v}} & 0 \\ 0 & 0 & \frac{1}{\vec{w}\cdot\vec{w}} \end{bmatrix} \cdot \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix} \cdot \left\{ \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix} \right\}. \tag{A.7}$$

odnosno zapisano vektorski:

$$\vec{T}' = S^2 \cdot \Gamma^T \left(\vec{T} - \vec{O}' \right). \tag{A.8}$$

Primjetite da su svi dosadašnji izvodi rezultirali konvencijom u kojoj matrica operatora množi točku (što je proizašlo iz predstavljanja točaka i vektora kao jednostupčanih matrica). Želimo li okrenuti konvenciju, sve izraze samo treba transponirati: time ćemo automatski dobiti i množenje točaka matricama operatora a točke i vektori će biti predstavljeni jednoretčanim matricama.

A.2.3 Prevođenje između dva zadana koordinatna sustava s ortogonalnim bazama

Ostalo nam je još za pogledati što napraviti ako imamo osnovni koordinatni sustav i u njemu zadana dva druga koordinatna sustava: prvi čije je ishodište O' i matrica vektora koordinatnih osi Γ' te drugi čije je ishodište O'' i matrica vektora koordinatnih osi Γ''. Zadane su koordinate točke T' u prvom koordinatnom sustavu a zanimaju nas njezine koordinate u drugom koordinatnom sustavu.

Transformaciju ćemo napraviti u dva koraka. U prvom koraku točku T' ćemo izrazom (A.1) prebaciti u osnovni koordinatni sustav:

$$\vec{T} = \vec{O}' + \Gamma' \cdot \vec{T}'.$$

Potom ćemo točku T u skladu s izrazom (A.8) poslati u drugi koordinatni sustav:

$$\begin{aligned}\vec{T}'' &= \Gamma''^{-1} \left\{ \vec{T} - \vec{O}'' \right\} \\ &= \Gamma''^{-1} \left\{ \vec{O}' + \Gamma' \cdot \vec{T}' - \vec{O}'' \right\} \\ &= \Gamma''^{-1} \left\{ \Gamma' \cdot \vec{T}' - (\vec{O}'' - \vec{O}') \right\}\end{aligned}$$

Konačan izraz dan je u nastavku.

$$\vec{T}'' = S''^2 \cdot \Gamma''^T \left\{ \Gamma' \cdot \vec{T}' - (\vec{O}'' - \vec{O}') \right\}. \tag{A.9}$$

Primjetite da se i ovaj najopćenitiji slučaj može prikazati u uobičajenom obliku gdje matrica množi translatiranu točku. Evo kako:

$$\begin{aligned}\vec{T}'' &= S''^2 \cdot \Gamma''^T \cdot I \cdot \left\{ \Gamma' \cdot \vec{T}' - (\vec{O}'' - \vec{O}') \right\} \\ &= S''^2 \cdot \Gamma''^T \cdot \Gamma' \cdot \Gamma'^{-1} \left\{ \Gamma' \cdot \vec{T}' - (\vec{O}'' - \vec{O}') \right\} \\ &= S''^2 \cdot \Gamma''^T \cdot \Gamma' \left\{ \vec{T}' - \Gamma'^{-1} \cdot (\vec{O}'' - \vec{O}') \right\} \\ &= S''^2 \cdot \Gamma''^T \cdot \Gamma' \left\{ \vec{T}' - S'^2 \cdot \Gamma'^T \cdot (\vec{O}'' - \vec{O}') \right\}.\end{aligned}$$

Točka $\vec{T}'$ najprije se translatira za vektor $-S'^2 \cdot \Gamma'^T \cdot (\vec{O}'' - \vec{O}')$ i potom množi matricom $S''^2 \cdot \Gamma''^T \cdot \Gamma'$.

Dodatak B

Dodatno o krivuljama

B.1 Presjecišta zrake i krivulje

U pojedinim situacijama važno je odrediti sjeku li se zraka i segment krivulje te ako da, predstavlja li sjecište mjesto na kojem zraka ulazi s lijeve strane i prelazi na desnu ili ulazi s desne strane i prelazi na lijevu. S obzirom da smo u ovoj knjizi u potpoglavlju 7.7.3 opisali postupak popunjavanja unutrašnjosti *glypha* koji se koriste kod vektorski definiranih fontova, u ovom potpoglavlju razmotrit ćemo dvije krivulje koje se koriste pri definiranju segmenata kod *TrueType* fontova: segment pravca te segment kvadratne aproksimacijske Bézierove krivulje. U oba slučaja razmatrat ćemo segment definiran parametarskim oblikom i to isključivo u 2D radnom prostoru.

B.1.1 Presjecište zrake i linijskog segmenta

Zraka je polupravac: to je dio pravca koji možemo definirati početnom točkom $\vec{Z}$ i vektorom smjera $\vec{d}$. Parametarski zapis zrake tada glasi:

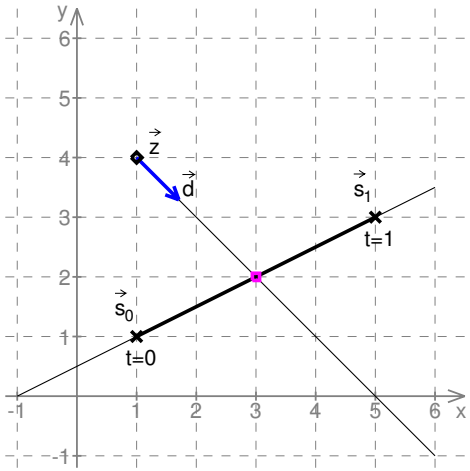
$$\vec{P}(\lambda) = \vec{Z} + \lambda \cdot \vec{d} \tag{B.1}$$

pri čemu je $\lambda \geq 0$. Dio pravca na kojem zraka leži a koji se dobiva za $\lambda < 0$ ne pripada zraci.

Pretpostavimo sada da je linijski segment definiran početnom točkom $\vec{s}_0$ i krajnjom točkom $\vec{s}_1$. Parametarski oblik ovako definiranog linijskog segmenta je:

$$\vec{P}(t) = \vec{s}_0 + t \cdot (\vec{s}_1 - \vec{s}_0) \tag{B.2}$$

pri čemu segmentu pripadaju samo one točke za koje je $0 \leq t \leq 1$. Točke za koje je $t < 0$ i točke za koje je $t > 1$ leže na istom pravcu na kojem leži i segment ali prve se nalaze ispred $\vec{s}_0$ a druge iza $\vec{s}_1$ pa ne pripadaju linijskom segmentu.



Slika B.1: Sjecište zrake i linijskog segmenta.

Slika B.1 ilustrira situaciju i kojoj sjecište zrake i linijskog segmenta postoji. Sjecišta zrake i linijskog segmenta su sve točke za koje vrijedi:

$$\vec{P}(\lambda) = \vec{P}(t) \tag{B.3}$$

uz $\lambda \geq 0$ i $0 \leq t \leq 1$. U slučaju koji razmatramo zraka i linijski segment mogu imati najviše jedno sjecište.

Izjednačavanjem izraza (B.1) i (B.2) dobivamo sustav dvije jednadžbe s dvije nepoznanice (jer radimo u 2D radnom prostoru pa su vektori dvokomponentni; nepoznanice su λ i t):

$$\vec{Z} + \lambda \cdot \vec{d} = \vec{s}_0 + t \cdot (\vec{s}_1 - \vec{s}_0) \tag{B.4}$$

koji je moguće riješiti na uobičajen način. Međutim, u nastavku ćemo pokazati drugačiji način rješavanja koji ćemo iskoristiti i kod kvadratne aproksimacijske Bézierove krivulje. Uvedimo pomoćni vektor $\vec{d}'$ koji je okomit na vektor $\vec{d}$. Ako je vektor $\vec{d} = \begin{bmatrix} d_x & d_y \end{bmatrix}$, tada je vektor $\vec{d}' = \begin{bmatrix} -d_y & d_x \end{bmatrix}$ (uz podsjetnik da smo ograničeni na 2D radni prostor). Pomnožimo sada skalarno izraz (B.4) s $\vec{d}'$. Slijedi:

$$\vec{Z} \cdot \vec{d}' + \lambda \cdot \vec{d} \cdot \vec{d}' = \vec{s}_0 \cdot \vec{d}' + t \cdot (\vec{s}_1 - \vec{s}_0) \cdot \vec{d}'.$$

Kako su vektori $\vec{d}$ i $\vec{d}'$ međusobno okomiti, njihov je skalarni produkt jednak nuli pa slijedi:

$$\vec{Z} \cdot \vec{d}' = \vec{s}_0 \cdot \vec{d}' + t \cdot (\vec{s}_1 - \vec{s}_0) \cdot \vec{d}'$$

odnosno:

$$t = \frac{\vec{Z} \cdot \vec{d}' - \vec{s}_0 \cdot \vec{d}'}{(\vec{s}_1 - \vec{s}_0) \cdot \vec{d}'}$$

što se uz distributivnost skalarnog produkta može kraće zapisati kao:

$$t = \frac{\left(\vec{Z} - \vec{s}_0\right) \cdot \vec{d'}}{(\vec{s}_1 - \vec{s}_0) \cdot \vec{d'}}. \tag{B.5}$$

Prije direktne primjene izraza (B.5) treba provjeriti je li nazivnik jednak 0. Ako je $(\vec{s}_1 - \vec{s}_0) \cdot \vec{d'} = 0$, zraka i linijski segment su paralelni pa sjecište ne postoji i postupak traženja sjecišta završava. U suprotnom, primjenom izraza (B.5) određujemo vrijednost parametra t koja odgovara točki u kojoj se sijeku *pravac na kojem leži zraka* i *pravac na kojem leži linijski segment*. Sada treba provjeriti iznos parametra t: ako smo dobili $t < 0$ ili $t > 1$, sjecište se nalazi na pravcu na kojem leži linijski segment ali ne pripada linijskom segmentu: postupak se opet prekida uz dojavu da sjecište zrake i linijskog segmenta ne postoji.

Ako smo utvrdili da je $0 \leq t \leq 1$, utvrdit ćemo točku potecijalnog sjecišta uvrštavanjem vrijednosti t u izraz (B.2) (kažemo potencijalnog jer još ne znamo na kojem se dijelu pravca na kojem leži zraka to sjecište nalazi). Da bismo završili postupak, moramo još odrediti vrijednost parametra λ. Kako smo izračunali $\vec{P}(t)$ i kako u sjecištu vrijedi (B.3), slijedi da je:

$$\vec{P}(t) = \vec{Z} + \lambda \cdot \vec{d}.$$

Množenjem čitavog izraza s $\frac{\vec{d}}{\|\vec{d}\|^2}$ slijedi:

$$\vec{P}(t) \cdot \frac{\vec{d}}{\|\vec{d}\|^2} = \vec{Z} \cdot \frac{\vec{d}}{\|\vec{d}\|^2} + \lambda \cdot \vec{d} \cdot \frac{\vec{d}}{\|\vec{d}\|^2}$$

što je dalje:

$$\vec{P}(t) \cdot \frac{\vec{d}}{\|\vec{d}\|^2} = \vec{Z} \cdot \frac{\vec{d}}{\|\vec{d}\|^2} + \lambda \cdot \frac{\vec{d} \cdot \vec{d}}{\|\vec{d}\|^2}$$

odnosno:

$$\vec{P}(t) \cdot \frac{\vec{d}}{\|\vec{d}\|^2} = \vec{Z} \cdot \frac{\vec{d}}{\|\vec{d}\|^2} + \lambda \cdot \frac{\|\vec{d}\|^2}{\|\vec{d}\|^2}$$

a kako je $\frac{\|\vec{d}\|^2}{\|\vec{d}\|^2} = 1$ slijedi:

$$\vec{P}(t) \cdot \frac{\vec{d}}{\|\vec{d}\|^2} = \vec{Z} \cdot \frac{\vec{d}}{\|\vec{d}\|^2} + \lambda.$$

Stoga je konačni izraz za parametar λ:

$$\lambda = \frac{\left(\vec{P}(t) - \vec{Z}\right) \cdot \vec{d}}{\|\vec{d}\|^2}. \tag{B.6}$$

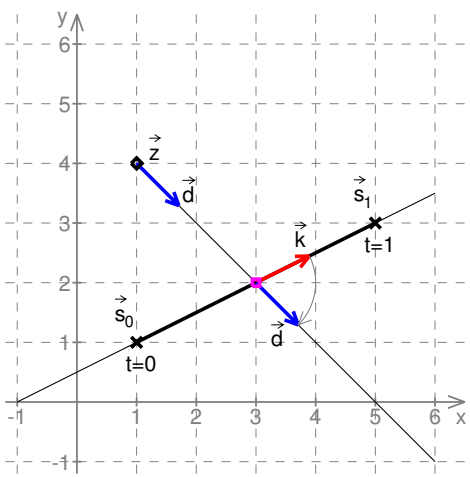
Nakon izračuna vrijednosti λ potrebno je provjeriti je li $\lambda < 0$; ako je, sjecište zrake i linijskog segmenta ne postoji i postupak se prekida.

Ako postupak do sada nije prekinut, utvrdili smo da zraka i linijski segment nisu paralelni, utvrdili smo da postoji sjecište pravca na kojem leži zraka i pravca na kojem leži linijski segment, utvrdili smo da to sjecište pripada linijskom segmentu ($0 \leq t \leq 1$) i konačno, utvrdili smo da to sjecište pripada i zraci ($\lambda \geq 0$).

Na ovom mjestu postupak je sigurno utvrdio postojanje sjecišta: ono što još treba utvrditi jest ulazi li u to sjecište zraka s lijeve strane ili s desne strane, pri čemu stranu gledamo iz referentnog sustava u kojem se šećemo po linijskom segmentu od početne točke prema konačnoj točki. Ovo ćemo utvrditi oslanjajući se na vektorski produkt tangente na "krivulju" u točki sjecišta i vektora smjera zrake. S obzirom da je "krivulja" u ovom slučaju linijski segment, tangenta je u svakoj točki ista pa kao tangentu $\vec{k}$ možemo uzeti:

$$\vec{k} = \frac{\vec{s}_1 - \vec{s}_0}{\|\vec{s}_1 - \vec{s}_0\|}.$$

Ideja je prikazana na slici B.2.



Slika B.2: Utvrđivanje orijentacije sjecišta

Neka je prikaz tangente po komponentama $\vec{k} = \begin{bmatrix} k_x & k_y \end{bmatrix}$. Vektor orijentacije $\vec{o}$ definirat ćemo kao vektorski produkt vektora tangente $\vec{k}$ i vektora smjera zrake $\vec{d}$ proširenih na tri dimenzije postavljanjem treće komponente na vrijednost 0:

$$\vec{o} = \vec{k}_{3D} \times \vec{d}_{3D} = \begin{bmatrix} k_x & k_y & 0 \end{bmatrix} \times \begin{bmatrix} k_x & k_y & 0 \end{bmatrix}. \tag{B.7}$$

Kako oba vektora leže u x-y ravnini i kako vektorski produkt rezultira vektorom koji je okomit na oba vektora, slijedi da će x i y komponente vektora $\vec{o}$ nužno biti

$o_x = o_y = 0$. Jedino će komponenta o_z biti različita od nule. Po pravilu desne ruke i gledajući sliku (B.2) na kojoj zraka ulazi s lijeve strane, rezultantni vektor gledat će u papir odnosno komponenta o_z će biti negativna. Da je zraka dolazila s desne strane, vektorski produkt bi rezultirao vektorom koji gleda prema gore (izlazi iz papira) pa bi komponenta o_z bila pozitivna.

Čitav postupak možemo sažeti na sljedeći način.

1. Provjeri jesu li zraka i linijski segment paralelni. Ako jesu, prekini postupak; sjecište ne postoji.
2. Uporabom (B.5) utvrdi t koji odgovara sjecištu. Ako je $t < 0$ ili $t > 1$, prekini postupak; sjecište ne postoji.
3. Izračunaj $\vec{P}(t)$.
4. Izračunaj λ prema (B.6).
5. Ako je $\lambda < 0$, prekini postupak; sjecište ne postoji.
6. Utvrdi tangentu $\vec{k}$ u točki sjecišta.
7. Izračunaj orijentacijski vektor $\vec{o} = \vec{k}_{3D} \times \vec{d}_{3D}$.
8. Orijentacija sjecišta utvrđuje se na način opisan u nastavku.
   - Ako je $o_z < 0$, zraka ulazi s lijeve strane.
   - Ako je $o_z > 0$, zraka ulazi s desne strane.
   - Situacija u kojoj je $o_z = 0$ u opisanom algoritmu nije moguća.

B.1.2 Presjecište zrake i kvadratne aproksimacijske Bézierove krivulje

Pretpostavimo da je zraka definirana na jednak način kao u prethodnom potpoglavlju. Segment kvadratne aproksimacijske Bézierove krivulje čiji kontrolni poligon čine redom vektori $\vec{r}_0$, $\vec{r}_1$ te $\vec{r}_2$ definiran je izrazom:

$$\vec{P}(t) = (1-t)^2 \cdot \vec{r}_0 + 2 \cdot (1-t) \cdot t \cdot \vec{r}_1 + t^2 \cdot \vec{r}_2 \tag{B.8}$$

uz $0 \leq t \leq 1$. Ovo se dalje može raspisati kako slijedi:

$$\vec{P}(t) = (1 - 2t + t^2) \cdot \vec{r}_0 + (2t - 2t^2) \cdot \vec{r}_1 + t^2 \cdot \vec{r}_2$$

odnosno

$$\vec{P}(t) = t^2 \cdot (\vec{r}_0 - 2\vec{r}_1 + \vec{r}_2) + t \cdot (-2\vec{r}_0 + 2\vec{r}_1) + \vec{r}_0. \tag{B.9}$$

Prije no što potražimo sjecišta ovog segmenta sa zrakom potrebno je skrenuti pažnju na krivulju koju dobijemo ako t pustimo od $-\infty$ do $+\infty$: kvadratna aproksimacijska Bézierova krivulja uz ograničenje $0 \leq t \leq 1$ predstavlja samo jedan segment *parabole*. Stoga, prilikom traženja sjecišta zrake i kvadratne aproksimacijske Bézierove krivulje možemo očekivati da ćemo dobiti od 0 do 2 potencijalna sjecišta. Nekoliko situacija koje se mogu pojaviti prikazane su na slici B.3. Na slikama je prikazana parabola na kojoj leži segment kvadratne aproksimacijske Bézierove krivulje, taj segment (podebljano) te zraka s kojom se traži sjecište.

Slika B.3a prikazuje situaciju u kojoj postoji 0 potencijalnih sjecišta. Slika B.3b prikazuje situaciju u kojoj postoje dva potencijalna sjecišta od kojih će prvo biti odbačeno jer se nalazi prije prve točke segmenta (ispred $\vec{r}_0$, pa će odgovarajući t biti negativan) a drugo će biti prihvaćeno kao sjecište jer pripada i zraci i segmentu (prihvaćanje/odbacivanje označeno je i različitim bojama – zeleno su prikazana sjecišta koja se prihvaćaju). Slika B.3c prikazuje situaciju u kojoj postoje dva potencijalna sjecišta i oba su doista i sjecišta zrake i segmenta. Slika B.3d prikazuje situaciju u kojoj postoje dva potencijala sjecišta ali se oba odbacuju jer ne pripadaju segmentu krivulje (prvo ima $t < 0$ a drugo $t > 1$). Konačno, slika B.3e prikazuje situaciju u kojoj postoje dva potencijalna sjecišta ali se prvo odbacuje jer ne pripada zraci (postiže se za $\lambda < 0$).

Kao radni primjer uzet ćemo situaciju prikazanu na slici B.4.

U točki sjecišta vrijedi:

$$\vec{P}(\lambda) = \vec{P}(t).$$

Izjednačavanjem dobivamo:

$$\vec{Z} + \lambda \cdot \vec{d} = (1-t)^2 \cdot \vec{r}_0 + 2 \cdot (1-t) \cdot t \cdot \vec{r}_1 + t^2 \cdot \vec{r}_2.$$

Množenjem čitavog izraza vektorom $\vec{d}'$ koji je okomit na vektor $\vec{d}$ (baš kako smo napravili i kod traženja sjecišta zrake i linijskog segmenta) dobivamo:

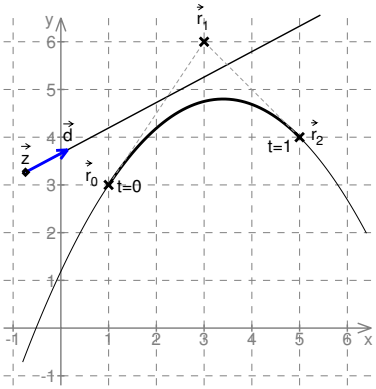
$$\vec{Z} \cdot \vec{d}' + \lambda \cdot \vec{d} \cdot \vec{d}' = t^2 \cdot (\vec{r}_0 - 2\vec{r}_1 + \vec{r}_2) \cdot \vec{d}' + t \cdot (-2\vec{r}_0 + 2\vec{r}_1) \cdot \vec{d}' + \vec{r}_0 \cdot \vec{d}'.$$

S obzirom da je $\vec{d} \cdot \vec{d}' = 0$, ostale članove možemo presložiti tako da dobijemo kvadratnu jednadžbu po parametru t:

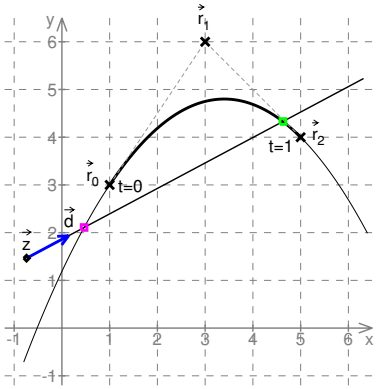
$$t^2 \cdot (\vec{r}_0 - 2\vec{r}_1 + \vec{r}_2) \cdot \vec{d}' + t \cdot (-2\vec{r}_0 + 2\vec{r}_1) \cdot \vec{d}' + \left(\vec{r}_0 - \vec{Z}\right) \cdot \vec{d}' = 0 \tag{B.10}$$

Ovo je klasična kvadratna jednadžba oblika $at^2 + bt + c = 0$ čija je diskriminanta $D = b^2 - 4ac$ što u našem slučaju daje:

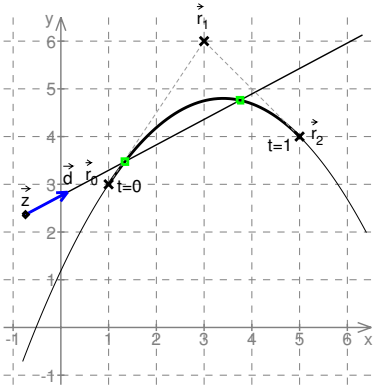
$$D = \left[(-2\vec{r}_0 + 2\vec{r}_1) \cdot \vec{d}'\right]^2 - 4 \cdot \left[(\vec{r}_0 - 2\vec{r}_1 + \vec{r}_2) \cdot \vec{d}'\right] \cdot \left[\left(\vec{r}_0 - \vec{Z}\right) \cdot \vec{d}'\right]. \tag{B.11}$$



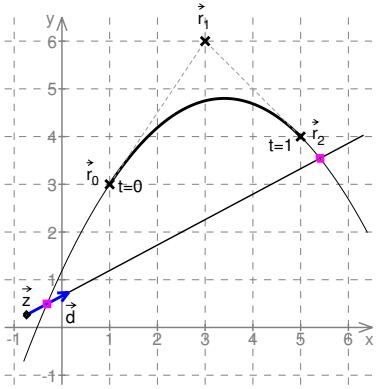
(a) Nema sjecišta



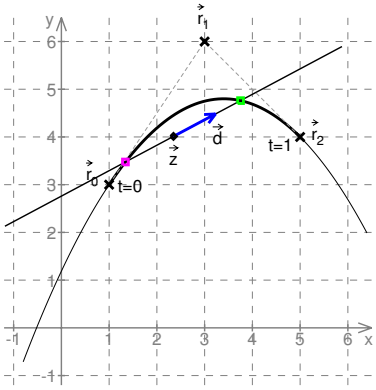
(b) Jedno sjecište



(c) Dva sjecišta

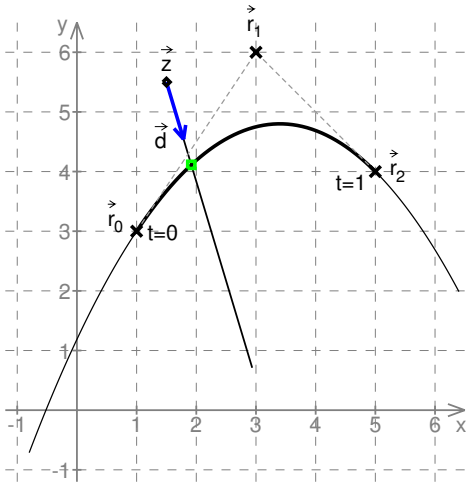


(d) Nema sjecišta



(e) Jedno sjecište

Slika B.3: Izbor različitih situacija koje se mogu pojaviti prilikom utvrđivanja sjecišta zrake i segmenta kvadratne aproksimacijske Bézierove krivulje.



Slika B.4: Utvrđivanje sjecišta zrake i kvadratne aproksimacijske Bézierove krivulje

Najprije ćemo izračunati diskriminantu D i pogledati njezin iznos. Ako je $D < 0$, pravac na kojem leži zraka i parabola na kojoj leži segment nemaju sjecišta (pa tada sigurno niti zraka i segment nemaju sjecišta) – postupak se prekida jer sjecište ne postoji. Ovo je situacija koju imamo na slici B.3a.

Ako je $D \geq 0$ nastavljamo postupak: računamo vrijednosti parametra t za koji se pravac na kojem leži zraka i parabola na kojoj leži segment sijeku (potencijalna sjecišta):

$$t_{1,2} = \frac{-\left[(-2\vec{r}_0 + 2\vec{r}_1) \cdot \vec{d}'\right] \pm \sqrt{D}}{2 \cdot (\vec{r}_0 - 2\vec{r}_1 + \vec{r}_2) \cdot \vec{d}'} \tag{B.12}$$

Točke potencijalnih sjecišta izračunat ćemo uvrštavanjem vrijednosti t_1 i t_2 u izraz (B.9). Označimo prvo potencijalno sjecište s $\vec{P}_1$ a drugo s $\vec{P}_2$. Koristeći izraz (B.6) odredit ćemo λ_1 koji odgovara potencijalnom sjecištu $\vec{P}_1$ i λ_2 koji odgovara potencijalnom sjecištu $\vec{P}_2$. Sada, bez gubitka općenitosti, pretpostavimo da je $\lambda_1 < \lambda_2$ (ako nije, zamijenite ih, zamijenite točke $\vec{P}_1$ i $\vec{P}_2$ i zamijenite t_1 i t_2).

Za sjecište $\vec{P}_i$ sada znamo λ_i i t_i. Sjecišta za koja je bilo $\lambda_i < 0$ (pripadaju pravcu na kojem leži zraka ali ne i samoj zraci) bilo $t_i < 0$ ili $t_i > 1$ (pa leže na paraboli na kojoj leži segment ali ne pripadaju segmentu) potrebno je odbaciti. Primjerice, pogledamo li sliku B.3b, za prvo sjecište ćemo imati $\lambda_1 > 0$, $t_1 < 0$ (pripada zraci ali ne i segmentu) a za drugo $\lambda_2 > 0$, $0 \leq t_2 \leq 1$. U situaciji prikazanoj na slici B.3c imat ćemo $\lambda_1 > 0$, $\lambda_2 > 0$, $0 \leq t_1 \leq 1$, $0 \leq t_2 \leq 1$.

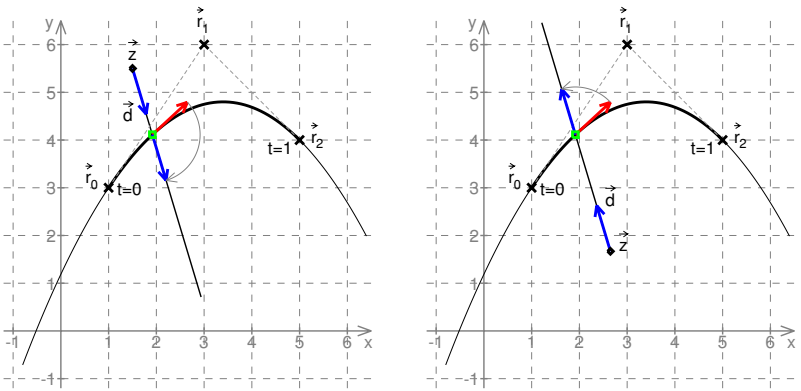
U situaciji prikazanoj na slici B.3d imat ćemo $\lambda_1 > 0$, $\lambda_2 > 0$, $t_1 < 0$, $t_2 > 1$. U situaciji prikazanoj na slici B.3e imat ćemo $\lambda_1 < 0$, $\lambda_2 > 0$, $0 \leq t_1 \leq 1$, $0 \leq t_2 \leq 1$.

Nakon ovog postupka odbacivanja ostalo nam je nula, jedno ili dva sjecišta (i u ovom posljednjem slučaju, sjecišta su poredana tako da prvo odgovara manjem λ). Ako nije preostalo niti jedno sjecište, postupak se prekida. Ako su preostala dva sjecišta, postupak se također može prekinuti kao da sjecište ne postoji (ovo ovisi o konkretnom algoritmu koji treba ovaj rezultat – ako zraka uđe u prostor ispod krivulje i zatim izađe iz tog prostora, algoritam može zanemariti oba sjecišta jer je zraka u konačnici i dalje u istom podprostoru) ili se pak može analizirati bliže sjecište (ono na koje zraka prvo naiđe).

U svakom slučaju, na ovom mjestu imamo jedno sjecište $\vec{P}_i$ za koje razmatramo orijentaciju (ulazi li u njega zraka s lijeve ili desne strane). Postupak utvrđivanja orijentacije napravit ćemo na isti način kao u slučaju linijskog segmenta: odredit ćemo vektor tangente u točki sjecišta, napraviti vektorski produkt tog vektora i vektora smjera zrake proširenih na 3D i potom pogledati z-komponentu dobivenog vektora. Tangentu ćemo utvrditi kao gradijent parametarskog prikaza (B.9):

$$\vec{k} = \nabla \vec{P}(t) = 2t \cdot (\vec{r}_0 - 2\vec{r}_1 + \vec{r}_2) + (-2\vec{r}_0 + 2\vec{r}_1)\,. \tag{B.13}$$

Konačno, orijentacijski vektor izračunat ćemo prema (B.7). Dvije ilustrativne situacije prikazane su na slici B.5.



(a) Zraka ulazi s lijeva (b) Zraka ulazi s desna

Slika B.5: Različite orijentacije sjecišta.

Na slici B.5a zraka ulazi s lijeve strane: vektorski produkt tangente i vektora smjera gleda u papir ($o_z < 0$). Na slici B.5b zraka ulazi s desne strane: vektorski produkt tangente i vektora smjera gleda iz papira ($o_z > 0$).

Postupak za utvrđivanje orijentacije sjecišta možemo sažeti na sljedeći način.

0. *Opcionalan korak u svrhu ubrzanja postupka.*
   Provjeriti postoji li sjecište zrake i konveksne ljuske kontrolnih točaka Bézierove krivulje. Ako ne postoji, prekini postupak. Inače nastavi sa sljedećim korakom.
1. Izračunaj diskriminantu D prema izrazu (B.11). Ako je $D < 0$, prekini postupak; niti jedno sjecište ne postoji.
2. Uporabom (B.12) utvrdi t_1 i t_2 koji odgovaraju potencijalnim sjecištima.
3. Uporabom (B.9) izračunaj $\vec{P}(t_1)$ i $\vec{P}(t_2)$.
4. Uporabom (B.6) izračunaj λ_1 i λ_2.
5. Poredaj (P_1, λ_1, t_1) i (P_2, λ_2, t_2) tako da je $\lambda_1 < \lambda_2$.
6. Odbaci sjecišta za koja je $\lambda < 0$ ili $t < 0$ ili $t > 1$.
7. Ako nije ostalo niti jedno sjecište (ili ako su ostala 2, ovisno o algoritmu koji treba ovaj rezultat), prekini postupak; javi da sjecište ne postoji.
8. Neka je i indeks koji odgovara onom sjecištu P_i koje je preostalo eliminacijski postupak i ima manji λ.
9. Uporabom B.13 utvrdi tangentu $\vec{k}$ u točki sjecišta $\vec{P}_i$.
10. Izračunaj orijentacijski vektor $\vec{o} = \vec{k}_{3D} \times \vec{d}_{3D}$.
11. Orijentacija sjecišta utvrđuje se na način opisan u nastavku.
    - Ako je $o_z < 0$, zraka ulazi s lijeve strane.
    - Ako je $o_z > 0$, zraka ulazi s desne strane.
    - Ako je $o_z = 0$, zraka tangira krivulju: prekini postupak i javi da sjecište ne postoji.

Dodatak C

Prevođenje programa koji koriste GLUT

U ovom poglavlju osvrnut ćemo se na različite prevodioce i različite operacijske sustave, te dati naputak kako najjednostavnije doći do koda koji se može prevesti i pokrenuti. Cilj ovog poglavlja je prevesti program prikazan u ispisu 1.1 prikazanom u poglavlju 1.

C.1 Operacijski sustav Windows i primjeri u jeziku C++

Prvi korak je nabavka biblioteke *glut*[1] ili biblioteke *freeglut*[2] (preporučamo ovu posljednju). U oba slučaja potrebno je skinuti ZIP arhivu koja sadrži potrebne zaglavne datoteke, opis biblioteke (`*.lib`) te samu biblioteku (`*.dll`).

Potom je potrebno nabaviti prevodioc. Pogledat ćemo slučajeve uporabe alata *Microsoft Visual Studio*, zatim `gcc`-a (skinite i raspakirajte *MinGW – Minimalist GNU for Windows*[3]), te konačno *Free Borland C++ Compiler*[4].

C.1.1 Microsoft Visual Studio

Postupak izrade programa uporabom biblioteke *freeglut* na operacijskom sustavu Windows i uporabom alata *Microsoft Visual Studio* opisan je kroz nekoliko koraka u nastavku.

1. Sa stranice: `http://www.transmissionzero.co.uk/software/freeglut-devel/` skinuti "freeglut 3.0.0 MVSC Package".

[1] `http://www.xmission.com/~nate/glut.html`
[2] `http://freeglut.sourceforge.net/`
[3] `http://sourceforge.net/projects/mingw/`
[4] `http://edn.embarcadero.com/article/20633`

2. Sadržaj ZIP-arhive raspakirati u proizvoljan direktorij (npr. u `C:\libs`).

3. Pokrenite *Visual Studio* i stvorite novi projekt (Win32 Console Application).

4. U opcijama tog projekta potrebno je podesiti sljedeće opcije:

   (a) Pod sekcijom "VC++ Directories" u "Include Directories" potrebno je dodati putanju do `include` direktorija raspakirane arhive. U našem primjeru to bi bila putanja:
   `C:\libs\freeglut\include`.

   (b) Pod sekcijom "Linker" odabrati podsekciju "General" te pod "Additional Library Directories" dodati putanju do `lib` direktorija. U ovom primjeru to bi bila putanja:
   `C:\libs\freeglut\lib`.

   (c) Dodatno, pod sekcijom "Linker" odabrati podsekciju "Input" i ovdje pod "Additional Dependencies" dodati `freeglut.lib`.

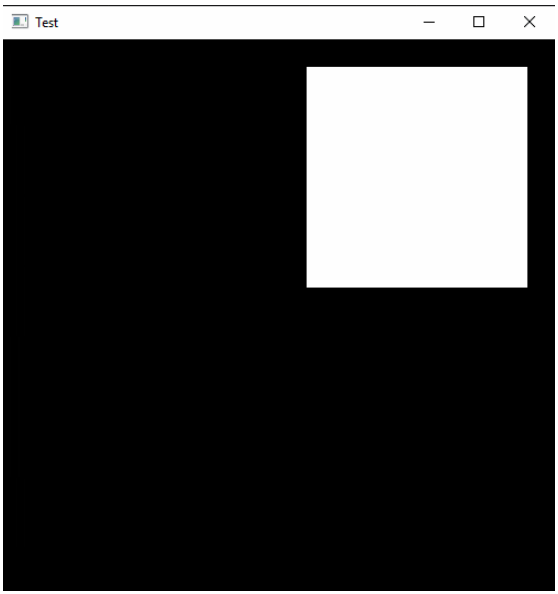
5. Nakon što su sve opcije podešene, iz direktorija `C:\libs\freeglut\bin` potrebno je prekopirati datoteku `freeglut.dll` u vršni direktorij projekta (do tog direktorija je najjednostavnije doći desnim klikom na projekt i odabirom opcije "Open Folder in File Explorer").

6. Nakon što su svi prethodni koraci obavljeni, probajte prekopirati i pokrenuti program prikazan u nastavku. Ako je sve dobro podešeno trebali bi dobiti prozor prikazan na slici C.1 u nastavku.

Ispis C.1: Primjer OpenGL programa u jeziku C

```
#include <GL/glut.h>

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
    glVertex3f(0.1, 0.1, 0.0);
    glVertex3f(0.1, 0.9, 0.0);
    glVertex3f(0.9, 0.9, 0.0);
    glVertex3f(0.9, 0.1, 0.0);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
```

```
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Test");
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Slika C.1: Primjer pokrenute aplikacije.

C.1.2 Gcc

Priprema

Pretpostavit ćemo da ste skinuli *MinGW* distribuciju, te da ste je raspakirali u direktorij `D:\usr\MinGW`. Također, pretpostavit ćemo da nam je cilj prevesti program `prvi.cpp` u napravljeni direktorij `D:\primjeri\primjer1`. Unutar direktorija `primjer1` napravite poddirektorije `include\GL` i `lib`. Otvorite *Command Prompt* i pozicionirajte se u taj direktorij.

```
D:
cd D:\primjeri\primjer1
```

Ako to već niste napravili u varijablama okruženja samog operacijskog sustava, dodajte `gcc` u `PATH`.

```
SET "PATH=%PATH%;D:\usr\MinGW\bin"
```

Uporaba biblioteke *glut*

Iz ZIP arhive *glut*-a u direktorij `include\GL` iskopirajte `glut.h`, u direktorij `lib` iskopirajte `glut32.lib`, a u direktorij `primjer1` iskopirajte samu biblioteku `glut32.dll`. Trebali biste dobiti sljedeću strukturu direktorija.

```
primjer1
  include
    GL
      glut.h
  lib
    glut32.lib
  glut32.dll
  prvi.cpp
```

Konačno, pokrenite prevođenje programa `prvi.cpp`.

```
gcc -Iinclude -Llib -o prvi.exe prvi.cpp -lglut32 -lopengl32
```

Program ćete potom pokrenuti sljedećim pozivom.

```
prvi.exe
```

Uporaba biblioteke *freeglut*

Iz ZIP arhive biblioteke *freeglut* iskopirati direktorije `include` i `lib` te biblioteku `freeglut.dll` u direktorij `primjer1`. Trebali biste dobiti strukturu direktorija kako je prikazano u nastavku.

```
primjer1
  include
    GL
      freeglut.h
      freeglut_ext.h
      freeglut_std.h
      glut.h
  lib
    freeglut.lib
  freeglut.dll
  prvi.cpp
```

Pokrenite prevođenje programa `prvi.cpp`.

```
gcc -Iinclude -Llib -o prvi.exe prvi.cpp -lfreeglut -lopengl32
```

Program ćete potom pokrenuti sljedećim pozivom.

```
prvi.exe
```

C.1.3 Bcc

Priprema

Pretpostavit ćemo da ste skinuli *Free Borland C++ Compiler* distribuciju, te da ste je raspakirali u direktorij `D:\usr\BCC55`. Također, pretpostavit ćemo da nam je cilj prevesti program `prvi.cpp` u napravljeni direktorij `D:\primjeri\primjer1`. Unutar direktorija `primjer1` napravite poddirektorije `include\GL` i `lib`. Otvorite *Command Prompt* i pozicionirajte se u taj direktorij.

```
D:
cd D:\primjeri\primjer1
```

Ako to već niste napravili u varijablama okruženja samog operacijskog sustava, dodajte `bcc32` u `PATH`.

```
SET "PATH=%PATH%;D:\usr\BCC55\bin"
```

Uporaba biblioteke *glut*

Iz ZIP arhive *glut*-a u direktorij `include\GL` iskopirajte `glut.h`, u direktorij `lib` iskopirajte `glut32.lib`, a u direktorij `primjer1` iskopirajte samu biblioteku `glut32.dll`. Trebali biste dobiti sljedeću strukturu direktorija.

```
primjer1
  include
    GL
      glut.h
  lib
    glut32.lib
  glut32.dll
  prvi.cpp
```

Opis biblioteke koji dolazi u ZIP arhivi nažalost nije kompatibilan s Borlandovim, pa je datoteku `glut32.lib` potrebno nanovo generirati. Taj posao napravit ćemo uporabom naredbe `implib` koja dolazi u Borlandovom paketu. Evo što treba napraviti.

```
cd lib
implib glut32.lib ..\glut32.dll
cd ..
```

Konačno, pokrenite prevođenje programa `prvi.cpp`.

```
bcc32 -Iinclude;D:\usr\bcc55\include
      -Llib;D:\usr\bcc55\lib;D:\usr\bcc55\lib\psdk
      -tWM glut32.lib prvi.cpp
```

Program ćete potom pokrenuti sljedećim pozivom.

```
prvi.exe
```

Uporaba biblioteke *freeglut*

Iz ZIP arhive biblioteke *freeglut* iskopirati direktorije `include` i `lib` te biblioteku `freeglut.dll` u direktorij `primjer1`. Trebali biste dobiti strukturu direktorija kako je prikazano u nastavku.

```
primjer1
  include
    GL
      freeglut.h
      freeglut_ext.h
      freeglut_std.h
      glut.h
  lib
    freeglut.lib
  freeglut.dll
```

Opis biblioteke koji dolazi u ZIP arhivi nažalost nije kompatibilan s Borlandovim, pa je datotku `freeglut.lib` potrebno nanovo generirati. Taj posao napravit ćemo uporabom naredbe `implib` koja dolazi u Borlandovom paketu. Evo što treba napraviti.

```
cd lib
implib freeglut.lib ..\freeglut.dll
cd ..
```

Konačno, pokrenite prevođenje programa `prvi.cpp`.

```
bcc32 -Iinclude;D:\usr\bcc55\include
      -Llib;D:\usr\bcc55\lib freeglut.lib prvi.cpp
```

Program ćete potom pokrenuti sljedećim pozivom.

```
prvi.exe
```

C.2 Operacijski sustav Linux i primjeri u jeziku C++

Sve potrebno prikazat ćemo na primjeru danas relativno popularne distribucije Ubuntu (konkretno, verzija 9.10). Ovdje nećemo niti razmatrati uporabu osnovne biblioteke *glut* jer se u paketnom sustavu proglašena zamijenjenom bibliotekom *freeglut3*. Stoga ćemo sve primjere raditi upravo korištenjem biblioteke *freeglut3*. Nakon instalacije Ubuntu-a trebalo je instalirati 3 paketa:

1. *prevodilac za c++*, naredbom

   ```
   sudo apt-get install g++
   ```

2. *biblioteku freeglut*, naredbom

   ```
   sudo apt-get install freeglut3
   ```

3. *zaglavne datoteke biblioteke freeglut*, naredbom

   ```
   sudo apt-get install freeglut3-dev
   ```

Potom napravite direktorij u koji ćemo smjestiti izvorni kod programa (primjerice `primjer1`), te u njega smjestite izvorni kod. Trebali biste dobiti sljedeću strukturu datoteka:

```
primjer1
  prvi.cpp
```

Sada je dovoljno ući u direktorij `primjer1`, i pokrenuti prevođenje, kako je prikazano u nastavku.

```
cd primjer1
g++ -o prvi prvi.cpp -lglut
```

Napomena: u izvornom kodu programa uključeno je i zaglavlje `windows.h`. Prije prevođenja programa na Linux-u taj je redak potrebno izbrisati. Nakon što prevođenje završi, program ćemo pokrenuti kako je opisano u nastavku.

```
./prvi
```

Tim pozivom prikazat će se novi prozor u kojem će biti nacrtan trokut i tri točke.

C.3 Primjeri u jeziku Java

U programskom jeziku Java nije moguće direktno pozivati OpenGL, s obzirom da je to zasebna tehnologija koja nije pisana u Javi. Međutim, iz programskog jezika Java funkcije koje nudi OpenGL moguće je pozivati uporabom tehnologije JNI koja Java programima omogućava da izvode dijelove koda koji su pisani nativno za operacijski sustav. Tehnologija JNI omogućava da se u Javi definiraju razredi koji međutim ne nude implementacije svih metoda već se metode označe ključnom riječi `native`. Takvi razredi potom trebaju deklarirati gdje se nalazi implementacija takve metode – na Windowsima će to biti u nekom DLL-u, na Linux-u u nekoj `.so` datoteci; spomenute datoteke mogu pak nastati direktno iz programskog jezika C ili C++ čime mogu pozivati sve nativne resurse operacijskog sustava.

Kako bi se programerima u Javi omogućio što jednostavniji pristup funkcijama koje nudi OpenGL, postoji već nekoliko biblioteka koje kroz tehnologiju JNI nude mogućnost uporabe nekog podskupa OpenGL-a. Jedna od najpopularnijih takvih biblioteka je JOGL – *Java-binding for OpenGL*. JOGL se slobodno može skinuti s Interneta; osnovna stranica preko koje se može doći i do dokumentacije, uputa i drugih resursa je `http://jogamp.org`.

Kako biste krenuli s uporabom JOGL-a, najjednostavnije je skinuti arhivu `http://jogamp.org/deployment/jogamp-current/archive/jogamp-all-platforms.7z`. Raspakirajte arhivu i dobit ćete direktorij `jogamp-all-platforms` s nizom poddirektorija. Sve što će Vam trebati nalazi se u poddirektoriju `jar`. Trebat ćete četiri `jar`-datoteke:

- `gluegen-rt.jar`
- `jogl-all.jar`
- gluegen-rt-natives-*OS-ARHITEKTURA*.jar
  (npr. `gluegen-rt-natives-linux-i586.jar`) te
- jogl-all-natives-*OS-ARHITEKTURA*.jar
  (npr. `jogl-all-natives-linux-i586.jar`).

Od ove četiri datoteke, prve dvije sadrže isključivo Java kod, dok druge dvije sadrže zapakirane nativne biblioteke (npr. u `gluegen-rt-natives-linux-i586.jar` će se nalaziti `.so` biblioteka, dok će i inačici za Windows operacijski sustav, u datoteci `gluegen-rt-natives-windows-i586.jar` biti odgovarajuća `.dll` datoteka).

Pogledajmo sada kako ćemo prevesti i pokrenuti jednostavan Java program koji koristi JOGL za pozivanje OpenGL-a i to direktno iz komandne linije. Napravit ćemo sljedeću strukturu direktorija:

```
demo
  +---- lib
         +---- gluegen-rt.jar
         +---- jogl-all.jar
         +---- gluegen-rt-natives-linux-i586.jar
         +---- jogl-all-natives-linux-i586.jar
  +---- src
         +---- test
                +---- SwingJOGLExample.java
  +---- bin
```

Ovaj primjer napravit ćemo na Linuxu. U direktorij `lib` potrebno je iskopirati sve četiri biblioteke. Ako radite na Windowsima, umjesto nativnih Linux biblioteka iskopirat ćete one koje odgovaraju Vašoj inačici Windowsa. Pretpostavimo sada da imate otvorenu konzolu, i da ste pozicionirani u direktorij `demo` (to je trenutni direktorij). Pripremili smo direktorij `src` u kojem će se nalaziti sve izvorne datoteke te direktorij `bin` u koji ćemo pohraniti generirane `.class` datoteke (prevedeni Java program). Naš ogledni program sastojat će se od razreda `SwingJOGLExample` koji će biti smješten u paket `test`; stoga na disku mora postojati datoteka `src/test/SwingJOGLExample.java` koja će sadržavati izvorni kod. Ovaj izvorni kod prikazan je u nastavku.

Ispis C.2: Primjer OpenGL programa u jeziku Java

```
 1  package test;
 2
 3  import java.awt.BorderLayout;
 4  import java.awt.event.KeyAdapter;
 5  import java.awt.event.KeyEvent;
 6  import java.awt.event.MouseAdapter;
 7  import java.awt.event.MouseEvent;
 8  import java.awt.event.MouseMotionAdapter;
 9  import java.awt.event.WindowAdapter;
10  import java.awt.event.WindowEvent;
11
12  import javax.media.opengl.GL;
13  import javax.media.opengl.GL2;
14  import javax.media.opengl.GLAutoDrawable;
15  import javax.media.opengl.GLCapabilities;
16  import javax.media.opengl.GLEventListener;
17  import javax.media.opengl.GLProfile;
18  import javax.media.opengl.awt.GLCanvas;
19  import javax.media.opengl.glu.GLU;
20  import javax.swing.JFrame;
21  import javax.swing.SwingUtilities;
22  import javax.swing.WindowConstants;
23
24  public class SwingJOGLExample {
```

```
25    static {
26      GLProfile.initSingleton();
27    }
28
29    public static void main(String[] args) {
30
31      SwingUtilities.invokeLater(new Runnable() {
32        @Override
33        public void run() {
34          GLProfile glprofile = GLProfile.getDefault();
35          GLCapabilities glcapabilities =
36            new GLCapabilities(glprofile);
37          final GLCanvas glcanvas = new GLCanvas(glcapabilities);
38
39          // Reagiranje na pritiske tipki na misu...
40          glcanvas.addMouseListener(new MouseAdapter() {
41            @Override
42            public void mouseClicked(MouseEvent e) {
43              System.out.println("Mis je kliknut na: x=" +
44                e.getX() + ", y=" + e.getY());
45              // Napravi nesto
46              // ...
47              // Posalji zahtjev za ponovnim crtanjem...
48              glcanvas.display();
49            }
50          });
51
52          // Reagiranje na pomicanje pokazivaca misa...
53          glcanvas.addMouseMotionListener(new MouseMotionAdapter() {
54            @Override
55            public void mouseMoved(MouseEvent e) {
56              System.out.println("Mis pomaknut na: x=" +
57                e.getX() + ", y=" + e.getY());
58              // Napravi nesto
59              // ...
60              // Posalji zahtjev za ponovnim crtanjem...
61              glcanvas.display();
62            }
63          });
64
65          // Reagiranje na pritiske tipaka na tipkovnici...
66          glcanvas.addKeyListener(new KeyAdapter() {
67            @Override
68            public void keyPressed(KeyEvent e) {
69              if (e.getKeyCode() == KeyEvent.VK_R) {
70                e.consume();
71                // Napravi nesto
72                // ...
73                // Posalji zahtjev za ponovnim crtanjem...
74                glcanvas.display();
75              }
```

```
76            }
77          });
78
79          // Reagiranje na promjenu velicine platna, na zahtjev za
80          // crtanjem i slicno...
81          glcanvas.addGLEventListener(new GLEventListener() {
82            @Override
83            public void reshape(GLAutoDrawable glautodrawable,
84                int x, int y, int width, int height) {
85              GL2 gl2 = glautodrawable.getGL().getGL2();
86              gl2.glMatrixMode(GL2.GL_PROJECTION);
87              gl2.glLoadIdentity();
88
89              // coordinate system origin at lower left with
90              // width and height same as the window
91              GLU glu = new GLU();
92              glu.gluOrtho2D(0.0f, width, 0.0f, height);
93
94              gl2.glMatrixMode(GL2.GL_MODELVIEW);
95              gl2.glLoadIdentity();
96
97              gl2.glViewport(0, 0, width, height);
98            }
99
100           @Override
101           public void init(GLAutoDrawable glautodrawable) {
102           }
103
104           @Override
105           public void dispose(GLAutoDrawable glautodrawable) {
106           }
107
108           @Override
109           public void display(GLAutoDrawable glautodrawable) {
110             GL2 gl2 = glautodrawable.getGL().getGL2();
111             int width = glautodrawable.getWidth();
112             int height = glautodrawable.getHeight();
113
114             gl2.glClear(GL.GL_COLOR_BUFFER_BIT);
115
116             // draw a triangle filling the window
117             gl2.glLoadIdentity();
118             gl2.glBegin(GL.GL_TRIANGLES);
119             gl2.glColor3f(1, 0, 0);
120             gl2.glVertex2f(0, 0);
121             gl2.glColor3f(0, 1, 0);
122             gl2.glVertex2f(width, 0);
123             gl2.glColor3f(0, 0, 1);
124             gl2.glVertex2f(width / 2, height);
125             gl2.glEnd();
126           }
```

```
127           });
128
129           final JFrame jframe = new JFrame(
130               "Primjer prikaza obojanog trokuta");
131           jframe.setDefaultCloseOperation(
132             WindowConstants.DO_NOTHING_ON_CLOSE);
133           jframe.addWindowListener(new WindowAdapter() {
134             public void windowClosing(WindowEvent windowevent) {
135               jframe.dispose();
136               System.exit(0);
137             }
138           });
139           jframe.getContentPane().add(
140             glcanvas, BorderLayout.CENTER);
141           jframe.setSize(640, 480);
142           jframe.setVisible(true);
143           glcanvas.requestFocusInWindow();
144         }
145       });
146     }
147   }
```

Prevođenje programa tada se obavlja naredbom:

```
javac -sourcepath src -d bin
      -cp lib/gluegen-rt.jar:lib/jogl-all.jar
      src/test/SwingJOGLExample.java
```

(čitavu naredbu potrebno je unijeti kao jedan redak) čime će u `bin` direktoriju nastati poddirektorij `test` i unutar njega datoteka `SwingJOGLExample.class`. Pokretanje programa radi se naredbom:

```
java -cp lib/gluegen-rt.jar:lib/jogl-all.jar:bin
     -Dsun.java2d.noddraw=true test.SwingJOGLExample
```

U ova dva poziva, ako ste na operacijskom sustavu Windows, umjesto dvotočke za razdvajanje više staza koristit ćete točku-zarez, a umjesto kose crte za razdvajanje poddirektorija koristit ćete obrnutu kosu crtu; ništa drugo se ne mijenja.

Ako se projekt radi u nekom razvojnom okruženju, potrebno je napraviti sličnu strukturu direktorija (to će tipično napraviti i sama razvojna okolina prilikom stvaranja novog projekta), i potom je potrebno u *classpath* (ili *build path*, ovisi kako se u kojem okruženju zove) dodati ove `.jar` datoteke.

Izvorni kod ovog primjera opširniji je od minimalnog kako bi demonstrirao način na koji se obavlja nekoliko često korištenih zadataka. U logičkom smislu možemo ga podijeliti u tri dijela. Retci 34-37 služe za stvaranje komponente po kojoj ćemo crtati OpenGL-om; komponenta koju koristimo primjerak je razreda GLCanvas i ovdje direktno stvaramo jedan primjerak te komponente. Međutim, u trenutku stvaranja, ta komponenta, kao i svaka druga Swing/AWT komponenta

mora biti dodana u neki prozor da bi postala vidljiva; stvaranje prozora (primjerak razreda JFrame i podešavanje obavlja se u retcima 129-143. Komponenta u koju ćemo crtati u prozor se dodaje u retcima 139-140 i bit će rastegnuta preko čitave površine prozora (izuzev naslovne trake prozora).

Središnji dio koda,retci 39-127 prikazuju kako se na stvorenu komponentu mogu dodati promatrači koje će komponenta obavještavati o različitim događajima; tako je u retcima 39-50 prikazano dodavanje promatrača miša kojim se mogu dobiti informacije o pritiscima tipki miša i slične (pogledajte detaljnije sučelje `MouseListener` koje definira sve metode; ovdje je iskorišten pomoćni razred MouseAdapter). U retcima 52-63 prikazano je dodavanje promatrača kojemu će se dojavljivati informacije o pomicanjima pokazivača miša (pogledajte detaljnije sučelje MouseMotionListener). U retcima 65-77 prikazano je dodavanje promatrača kojemu će se dojavljivati informacije o pritiscima tipki na tipkovnici. Da bi ovo funkcioniralo, komponenta mora imati fokus, i redak 143 upravo pokušava osigurati da se, odmah nakon otvaranja prozora, ova komponenta fokusira. Ako Vam iz nekog razloga informacije o pritisnutim tipkama ipak ne dolaze, pokušajte mišem kliknuti na platno. Konačno, u retcima 79-127 prikazano je dodavanje nama najvažnijeg promatrača – onog kojem će se dojavljivati da je komponenta promijenila dimenzije te da je potrebno ponovno nacrtati sliku koju komponenta prikazuje – ove dvije metode odgovaraju onim klasičnim metodama reshape i display koje smo u C-ovskim verzijama OpenGL primjera koristili od početka knjige.

Treba napomenuti da je komponenta GLCanvas po defaultu koristi dvostruki spremnik i, osim ako se to eksplicitno ne isključi, automatski nakon što naš promatrač nacrta scenu zove swap_buffers kako bi prikazala nacrtanu sliku. Stoga to nije potrebno raditi ručno.

C.4 Primjeri u jeziku Python

Postoji više načina kako koristiti OpenGL u Pythonu. Mi ćemo se ovdje osvrnuti na uporabu biblioteke *pyglet*[5]. Skinite i instalirajte biblioteku prema uputama s mrežnog mjesta. Program koji ilustrira uporabu ove biblioteke prikazan je u nastavku. Uz pretpostavku da je program pohranjen u datoteci `program.py`, program ćete pokrenuti naredbom:

```
python program.py
```

Ispis C.3: Primjer OpenGL programa u jeziku Python

```
1 from pyglet.gl import *
2 from pyglet.window import key
3 from pyglet.window import mouse
```

[5] `http://www.pyglet.org/`

```
4
5  # Otvaranje prozora
6  window = pyglet.window.Window()
7  triCol = [1.0, 0.0, 0.0]
8
9  @window.event
10 def on_mouse_press(x, y, button, modifiers):
11     global triCol
12     if button & mouse.LEFT:
13         triCol = triCol[2:] + triCol[:2]
14
15 @window.event
16 def on_key_press(symbol, modifiers):
17     global triCol
18     if modifiers & key.MOD_SHIFT:
19         triCol = triCol[1:] + triCol[0:1]
20
21 @window.event
22 def on_draw():
23     glClearColor(0.0, 0.0, 0.0, 1.0)
24     glClear(GL_COLOR_BUFFER_BIT)
25     glLoadIdentity()
26     glColor3f(triCol[0], triCol[1], triCol[2])
27     glBegin(GL_TRIANGLES)
28     glVertex2f(0, 0)
29     glVertex2f(window.width, 0)
30     glVertex2f(window.width/2.0, window.height)
31     glEnd()
32
33 @window.event
34 def on_resize(width, height):
35     glViewport(0, 0, width, height)
36     glMatrixMode(gl.GL_PROJECTION)
37     glLoadIdentity()
38     glOrtho(0, width, 0, height, -1, 1)
39     glMatrixMode(gl.GL_MODELVIEW)
40
41 pyglet.app.run()
```

Bibliografija

[1] John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, and Kurt Akeley. *Computer graphics: principles and practice (3rd ed.).* Addison-Wesley Professional, Boston, MA, USA, July 2013.

[2] Eric Lengyel. *Mathematics for 3D Game Programming and Computer Graphics, Third Edition.* Course Technology Press, Boston, MA, United States, 3rd edition, 2011.

[3] D. Shreiner and OpenGL Architecture Review Board. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2.* Graphics programming. Addison-Wesley, 2006.

[4] Dave Shreiner, Graham Sellers, John M. Kessenich, and Bill M. Licea-Kane. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3.* Addison-Wesley Professional, 8th edition, 2013.

[5] Donald D. Hearn, M. Pauline Baker, and Warren Carithers. *Computer Graphics with Open GL.* Prentice Hall Press, Upper Saddle River, NJ, USA, 4th edition, 2010.

[6] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[7] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering 3rd Edition.* A. K. Peters, Ltd., Natick, MA, USA, 2008.

[8] S. Turk. *Računarska grafika: osnovi teorije i primjene.* Školska knjiga, 1983.

[9] N. Guid. *Računalniška grafika.* Tehniška fakulteta, 1988.

[10] Paul S. Heckbert, editor. *Graphics Gems IV.* Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[11] Alan W. Paeth, editor. *Graphics Gems V: MacIntosh Versiion.* Academic Press, Inc., Orlando, FL, USA, 1995.

[12] Edward Angel. *Interactive Computer Graphics: A Top-Down Approach With OPENGL Primer Package-2Nd Edition.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2nd edition, 2001.

Kazalo