Interaktivna računalna grafika.

Zadatci za laboratorijske vježbe.

Marko Čupić Željka Mihajlović

15. ožujka 2013.

Sadržaj

Predgovor

Ovaj dokument predstavlja radnu verziju novih uputa za laboratorijske vježbe iz kolegija Interaktivna računalna grafika: *Interaktivna računalna grafika. Zadatci za laboratorijske vježbe.* Molimo sve pogreške, komentare, nejasnoće te sugestije dojaviti na
Marko.Cupic@fer.hr ili Zeljka.Mihajlovic@fer.hr.

Verzija dokumenta: 0.1.2013-03-15.

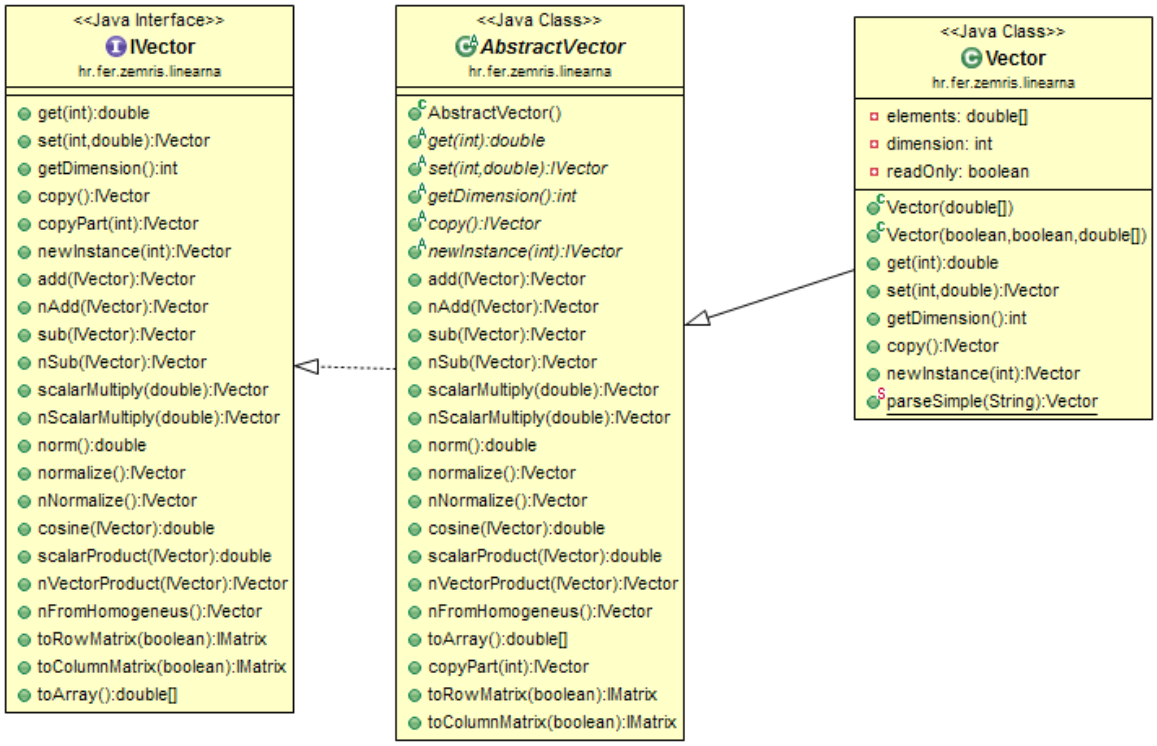
Laboratorijska vježba 1

Izrada pomoćne biblioteke

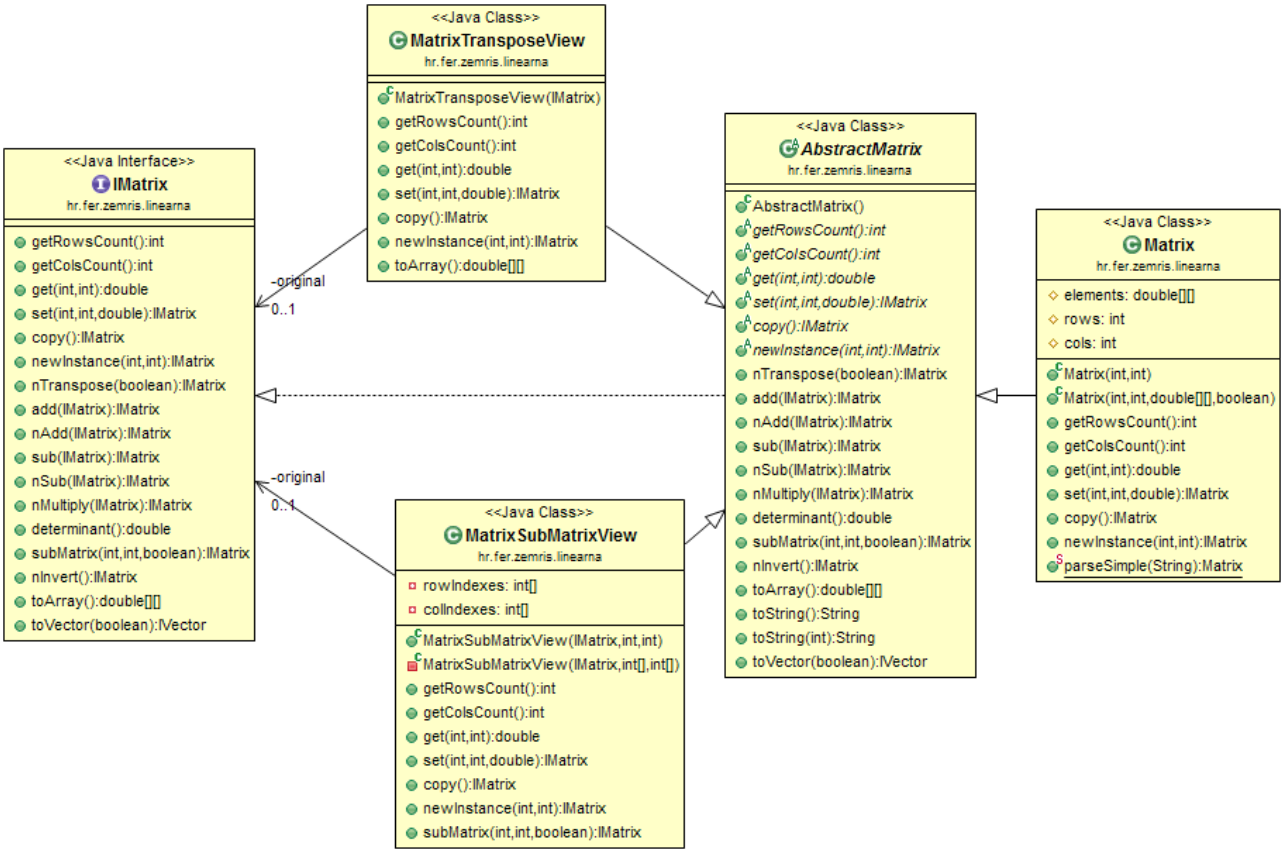
Osnovna matematička podloga interaktivne računalne grafike je linearna algebra. Unutar toga, dva najvažnija pojma su pojam vektora i pojam matrice. U okviru ove vježbe Vaš je zadatak napisati malu biblioteku koja se sastoji od nekoliko razreda i koja omogućava rad s vektorima i matricama. Primjer koji je ovdje dan kao i dokumentacija napisana je u programskom jeziku Java; međutim, kao jezik u kojem ćete raditi možete odabrati bilo koji objektno orijentirani programski jezik u kojem se dobro snalazite. U tom slučaju primjere koji su ovdje dani shvatite kao konceptualni naputak onoga što treba napraviti a konkretnu implementaciju napravite u jeziku po vašem izboru. Međutim, ovdje razvijena biblioteka trebat će Vam u nekim kasnijim vježbama; stoga odaberite jezik u kojem mislite raditi i ostale vježbe.

1.1 Zadatak

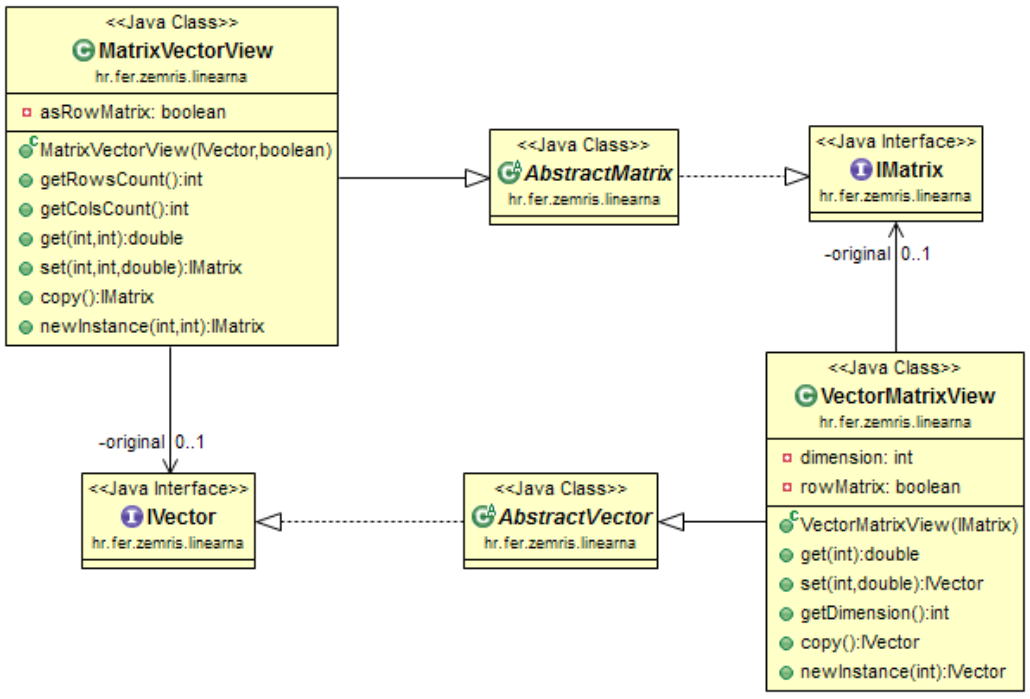
Na slikama 1.1, 1.2 i 1.3 dani su UML dijagrami svih sučelja i razreda koje trebate napisati.



Slika 1.1: Osnovni razredi povezani s modeliranjem vektora



Slika 1.2: Osnovni razredi povezani s modeliranjem matrica



Slika 1.3: Pomoćni razredi za preslikavanje između matrica i vektora

1.1.1 Modeliranje vektora

Modeliranje vektora započinje modeliranjem sučelja IVector (u jezicima koji ne podržavaju sučelja, sučelje se poistovjećuje s čistim apstraktnim razredom). Ovo sučelje sadrži deklaracije svih metode koje mora sadržavati jedan razred koji za sebe tvrdi da je vektor. U ovom sučelju postoje dvije porodice metoda koje kao rezultat opet vraćaju vektor. Metode koje u imenu sadrže samo naziv operacije, tu operaciju provode direktno nad vektorom nad kojim su pozvane i mijenjaju njegov sadržaj (drugim riječima, stari sadržaj se gubi). Metode koje započinju malim slovom `n` nakon čega imaju naziv operacije ne modificiraju trenutni objekt već stvaraju novi u koji zapisuju rezultat. Primjer ovoga su metode add i sub koje direktno mijenjaju vektor nad kojim se pozovu, nasuprot metodama nAdd i nSub koje rezultat operacije zapisuju u novi vektor koji vraćaju. Obje porodice metoda pri tome vraćaju referencu na vektor: u prvom slučaju vektor vraća referencu na samog sebe a u drugom slučaju vraća se referenca na novostvoreni vektor. To je napravljeno tako da bi se omogućilo ulančavanje poziva metoda i time osiguralo kompaktnije zapisivanje. Primjerice, ideja je da se može pisati nešto poput sljedećega:

```
1 IVector a = Vector.parseSimple("3 1 3");
2 IVector b = new Vector(-2,4,1);
3 double n = a.add(new Vector(1,1,1))
4             .add(b)
5             .nScalarProduct(new Vector(2,3,2));
```

Prvi korak u implementaciji razreda koji bi ponudio ovakvo ponašanje jest implementiranje zajedničkih metoda za različite vrste vektora – što je načinjeno u apstraktnom razredu AbstractVector koji implementira sučelje IVector. Taj razred pretpostavlja da postoje metode za dohvat i postavljanje vrijednosti elementa (get i set), za otkrivanje dimenzionalnosti vektora (getDimension) te za kopiranje trenutnog vektora (copy) i stvaranje novog vektora (newInstance); naravno, te metode u tom razredu još ne postoje pa je razred zato i apstraktan, no to nas ne sprječava da ove metode ipak koristimo. Uz pretpostavku da te metode postoje, razred AbstractVector daje implementacije svih preostalih metoda koje su realizirane samo pozivanjem ovih prethodno spomenutih metoda. Kao primjer slijedi implementacija metoda na zbrajanje vektora.

```
1  @Override
2  public IVector add(IVector other) throws IncompatibleOperandException {
3    if(this.getDimension()!=other.getDimension())
4          throw new IncompatibleOperandException();
5    for(int i = this.getDimension()-1; i >= 0; i--) {
6      this.set(i, this.get(i)+other.get(i));
7    }
8    return this;
9  }
10
11 @Override
12 public IVector nAdd(IVector other) throws IncompatibleOperandException {
13   return this.copy().add(other);
14 }
```

Na sličan način možete implementirati sve preostale metode, osim metoda toRowMatrix i toColumnMatrix, jer još nismo modelirali matrice; stoga te dvije metode privremeno zakomentirajte i kasnije se vratite na njih.

Metoda nFromHomogeneus trenutni vektor tretira kao vektor u homogenom prostoru i vraća vektor koji njemu odgovara u radom prostoru. Konkretno, kod vektora u homogenom prostoru posljednja komponenta tretira se kao homogena koordinata. Ako je vektor u homogenom prostoru d-dimenzijski, odgovarajući vektor u radnom prostoru bit će $(d-1)$-dimenzijski pri ćemu će pojedine komponente tog vektora biti jednake odgovarajućim komponentama vektora u homogenom prostoru podijeljenima s homogenom komponentom. Primjerice, ako je vektor u homogenom prostoru $(1, 3, 7, 2)$, odgovarajući vektor u radnom prostoru je $(1/2, 3/2, 7/2)$ odnosno $(0.5, 1.5, 3.5)$.

Metoda copyPart prima broj komponenata n vektora koje treba iskopirati u novi vektor koji će vratiti, pri ćemu se komponente broje od početka. Metoda uvijek vraća novi vektor koji je n-dimenzijski. Ako je n manji od dimenzionalnosti originalnog vektora, novi vektor će imati prvih n njegovih komponenti. Ako je n veći od dimenzionalnosti d trenutnog vektora, novi vektor će na prvih d mjesta imati prekopirane sve komponente originalnog vektora a na preostalih $n-d$ mjesta imat će vrijednost 0. Evo primjera.

```
1 IVector a = new Vector(-2,4,1);
2 IVector b = a.copyPart(2);
3 IVector c = a.copyPart(5);
```

Vektor b bit će vektor $(-2, 4)$ dok će vektor c biti $(-2, 4, 1, 0, 0)$.

Potom napravite konkretan razred Vector koji nasljeđuje apstraktni razred AbstractVector, elemente vektora čuva u privatnom polju decimalnih brojeva, nudi dva konstruktora na raspolaganje i implementira sve preostale apstraktne metode. Prvi konstruktor prima varijabilni broj elemenata (ili polje) i stvara vektor koji se inicijalizira s tim podatcima; ima dimenzija koliko je dobio argumenata a vrijednosti odgovaraju vrijednostima argumenata. Alternativni konstruktor prima dvije zastavice i polje vrijednosti. Prva zastavica regulira je li vektor koji nastaje nepromjenjiv (tj. takav da ga se smije samo čitati; pokušaj pisanja kod takvog vektora mora baciti iznimku). Ako je postavljena na **true**, vektor treba tretirati nepromjenjivim; ako je postavljena na **false**, komponente vektora dozvoljeno je mijenjati. Druga zastavica regulira smije li konstruktor preuzeti predano polje (treći argument); ako je postavljeno na **true**, pretpostavka je da je polje vrijednosti stvoreno baš za uporabu u vektoru koji se stvara te da se izvana neće mijenjati niti oslobađati; konstruktor stoga može preuzeti referencu na to polje i raditi direktno s njime (i na kraju ga, u destruktoru, osloboditi). Ako je druga zastavica postavljena na **false**, konstruktor mora pretpostaviti da će se sadržaj predanog polja izvana mijenjati ili da će se polje osloboditi; stoga za svoje potrebe treba zauzeti novo privatno polje i u njega iskopirati sadržaj predanog polja.

U razredu Vector implementirajte još i statičku metodu parseSimple koja prima jedan argument: string koji sadrži elemente vektora razdijeljene s jednim ili više razmaka. Metoda treba obraditi taj string i vratiti vektor koji odgovara predanom stringu. Prvi primjer u ovom zadatku već sadrži poziv ove metode.

Preporuka (no nije nužno) je u razredu AbstractVector još implementirati i metodu toString(**int** precision) koja stvara lijepo formatiranim string u kojem su svi decimalni brojevi ispisati sa zadanim brojem decimala, te metodu toString() koja poziva prethodnu uz argument 3.

1.1.2 Modeliranje matrice

Modeliranje matrice ostvareno je slično kao i modeliranje vektora: najprije je definirano apstraktno sučelje IMatrix, potom apstraktni razred AbstractMatrix koji implementira većinu funkcionalnosti ali još ne brine o načinu pohrane elemenata matrice, i konačno konkretni razred Matrix koji vrijednosti matrice čuva u dvodimenzijskom polju decimalnih brojeva. Na slici 1.2 slovom `A` su označene metode koje u apstraktnom razredu AbstractMatrix još ne implementiraju. Sve ostale metode implementiraju se njihovom uporabom.

Sučelje IMatrix definira dvije operacije (nTranspose i subMatrix) koje ovisno o vrijednosti predane zastavice liveView trebaju vratiti kopiju podataka ili živi pogled na matricu. Evo primjera koji bi to trebao pojasniti.

```
1 IMatrix m1 = Matrix.parseSimple("1 2 3 | 4 5 6");
2 IMatrix m2 = m1.nTranspose(true);
3
4 System.out.println("m1:");
5 System.out.println(m1.toString());
6 System.out.println("m2:");
7 System.out.println(m2.toString());
8 System.out.println();
```

```
9
10 m2.set(2,1,9);
11
12 System.out.println("m1:");
13 System.out.println(m1.toString());
14 System.out.println("m2:");
15 System.out.println(m2.toString());
```

U kodu se najprije stvara matrica m1 dimenzija 2×3 a potom se zahtjeva stvaranje živog pogleda na transponiranu matricu što se pamti kao m2. Matrica m2 stoga je matrica dimenzija 3×2. Rezultat izvođenja ovog koda prikazan je u nastavku.

```
m1:
[1.000, 2.000, 3.000]
[4.000, 5.000, 6.000]
m2:
[1.000, 4.000]
[2.000, 5.000]
[3.000, 6.000]

m1:
[1.000, 2.000, 3.000]
[4.000, 5.000, 9.000]
m2:
[1.000, 4.000]
[2.000, 5.000]
[3.000, 9.000]
```

Uočite što znači da je pogled "živ": nad transponiranim pogledom smo zatražili postavljanje vrijednosti 9 u treći redak i drugi stupac (odnosno na lokaciju $(2, 1)$ jer numeracija ide od nule). Ta je promjena potom vidljiva u ispisu matrice m2 ali također i u ispisu matrice m1. Da bismo ovo ostvarili, potreban nam je još jedan pomoćni razred: MatrixTransposeView. Taj razred također nasljeđuje AbstractMatrix ali za razliku od razreda Matrix, on svoje elemente ne čuva u privatnom polju. Umjesto toga, ovaj razred u konstruktoru dobiva referencu na originalnu matricu i potom pamti tu referencu. Kada ga se pita koliko ima redaka, on pita originalnu matricu koliko ona ima stupaca, i vrati taj podatak. Kada ga se pita koje je element u retku r i stupcu c, on pita originalnu matricu što piše u retku c i stupcu r i vrati taj podatak; i tako slično za sve ostale metode. Sličnu funkciju obavlja i pomoćni razred MatrixSubMatrixView koji omogućava dobivanje živog pogleda na matricu koja se dobije kada se iz originalne matrice izbaci neki redak i neki stupac (početna matrica, redak koji treba izbaciti i stupac koji treba izbaciti predaju se kao argumenti javnog konstruktora ovog razreda); ova operacija je često potrebna, a dva primjera gdje se javlja jest rekurzivni izračun determinante matrice te izračun matričnog inverza uporabom matrica kofaktora. Još jedna prednost stvaranja živih pogleda jest što izostaje kopiranje podataka čime se operacija stvaranja matrice provodi efikasnije (cijena se plaća kod pristupanja elemenata koje je sada nešto nepovoljnije). Uočite: razredi MatrixTransposeView i MatrixSubMatrixView su pogledi na originalne matrice; to znači da ne kopiraju podatke već su po definiciji "živi" pogledi. Promjena kroz njih automatski se propagira u originalni objekt. Razred MatrixSubMatrixView ima definiran i privatni konstruktor koji omogućava stvaranje pogleda kojemu se zadaju indeksi redaka te indeksi stupaca koji čine ovaj pogled. Primjerice, neka je originalna matrica m dimenzija 5 redaka puta 6 stupaca. **new** MatrixSubMatrixView(m, **new int**[]{2,4}, **new int**[]{4}) će stvoriti kao pogled matricu dimenzija dva retka puta jedan stupac: prvi redak te matrice odgovarat će trećem retku originalne matrice (jer indeksacija ide od nule), drugi redak će odgovarati petom retku originalne matrice; jedini stupac nove matrice odgovarat će petom stupcu originalne matrice.

Razred Matrix opremite još i metodom parseSimple koja prima kao argument jedan string koji predstavlja tekstovni zapis matrice – elementi u retku su razdvojeni s jednim ili više razmaka, a pojedini

retci su razdvojeni znakom |. U prethodnom primjeru ta je metoda korištena za inicijalizaciju matrice m1. Također, preporuča se da se u AbstractMatrix dodaju metode toString() i toString(precision) zbog ostvarivanja preglednog ispisa.

1.1.3 Povezivanje vektora i matrica

Konačno, još je potrebno modelirati dvije operacije koje se u praksi vrlo često koriste. Vektor dimenzija dim često se promatra kao jednoretčana matrica dimenzija $1 \times dim$ ili kao jednostupčana matrica dimenzija $dim \times 1$. Isto tako, jednoretčana matrica ili jednostupčana matrica često se promatra i kroristi kao vektor (pa se računa norma, kosinus kuta i slično). Stoga sučelje IVector definira metode kojima je na temelju trenutnog vektora moguće dobiti matrični pogled (pa čak i živi) – na početku ste ih zakomentirali pa ih sada otkomentirajte, dok sučelje IMatrix definira metode kojima se matrica (ako je jednoretčana ili jednostupčana) može pretvoriti u vektor. Za potrebe ostvarivanja živih pogleda potrebno je koristiti dva pomoćna razreda: MatrixVectorView (to je matrica koja podatke vuče iz vektora) te VectorMatrixView (to je vektor koji podatke vuče iz matrice). Konstruktor razreda MatrixVectorView pri tome prima referencu na originalni vektor te zastavicu koja, ako je postavljena na **true** vektor maskira kao jednoretčanu matricu a u suprotnom kao jednostupčanu matricu.

```
1 public MatrixVectorView(IVector original, boolean asRowMatrix) { ... }
```

Uočite da su oba opisana razreda "pogledi" na odgovarajući objekt. To znači da oni prilikom stvaranja ne kopiraju originalne podatke već se ponašaju kao živa veza prema originalnom objektu na koji gledaju. Stoga u okviru konstruktora oba razreda nema posebno definirane zastavice koja bi trebala definirati radi li se o živom pogledu ili ne.

1.2 Primjeri uporabe

U nastavku slijedi nekoliko primjera koje trebate isprobati bibliotekom koju ste upravo napisali.

1.2.1 Izračun baricentričnih koordinata

Neka su zadane tri vrha trokuta u 3D: $A = (1, 0, 0)$, $B = (5, 0, 0)$ te $C = (3, 8, 0)$, i neka je zadana točka $T = (3, 4, 0)$. Treba odrediti baricentrične koordinate te točke s obzirom na zadane vrhove trokuta. Evo programa.

```
1  IVector a = Vector.parseSimple("1 0 0");
2  IVector b = Vector.parseSimple("5 0 0");
3  IVector c = Vector.parseSimple("3 8 0");
4
5  IVector t = Vector.parseSimple("3 4 0");
6
7  double pov = b.nSub(a).nVectorProduct(c.nSub(a)).norm() / 2.0;
8  double povA = b.nSub(t).nVectorProduct(c.nSub(t)).norm() / 2.0;
9  double povB = a.nSub(t).nVectorProduct(c.nSub(t)).norm() / 2.0;
10 double povC = a.nSub(t).nVectorProduct(b.nSub(t)).norm() / 2.0;
11
12 double t1 = povA / pov;
13 double t2 = povB / pov;
14 double t3 = povC / pov;
15
16 System.out.println("Baricentricne koordinate su: ("+t1+","+t2+","+t3+").");
```

Rezultat bi trebao biti:

```
Baricentricne koordinate su: (0.25,0.25,0.5).
```

1.2.2 Rješavanje sustava jednadžbi

Neka imamo zadan sustav:

$$a_1 x + b_1 y = r_1$$
$$a_2 x + b_2 y = r_2$$

i neka su svi koeficijenti a_i, b_i i r_i zadani. Sustav se može zapisati matrično kao:

$$\mathbb{A} \cdot \vec{v} = \vec{r}$$

gdje je $\mathbb{A}$ matrica 2×2 a $\vec{v}$ i $\vec{r}$ vektori odnosno jednostupčane matrice:

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$$

Slijedi da je rješenje sustava vektor $\vec{v}$:

$$\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$$

Evo konkretnog primjera.

$$3x + 5y = 2$$
$$2x + 10y = 8$$

Slijedi:

$$\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 & 5 \\ 2 & 10 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 2 \\ 8 \end{bmatrix}$$

Program koji to računa prikazan je u nastavku.

```
1 IMatrix a = Matrix.parseSimple("3 5 | 2 10");
2 IMatrix r = Matrix.parseSimple("2 | 8");
3 IMatrix v = a.nInvert().nMultiply(r);
4 System.out.println("Rjesenje sustava je: ");
5 System.out.println(v);
```

Program bi trebao ispisati:

```
Rjesenje sustava je:
[-1.000]
[ 1.000]
```

što je doista korektno. $3 \cdot (-1) + 5 \cdot 1 = 2$ i $2 \cdot (-1) + 10 \cdot 1 = 8$.

1.2.3 Izračun baricentričnih koordinata, drugi način

Riješimo ponovno problem prikazan u odjeljku 1.2.1, ali ovaj puta promatrajući problem kao linearni sustav jednadžbi koji je potrebno riješiti. Uz zadane vrhove trokuta $\vec{A}$, $\vec{B}$ i $\vec{C}$, točku $\vec{T}$ te baricentrične koordinate t_1, t_2 i t_3, znamo da vrijedi:

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

gdje su $\vec{A}$, $\vec{B}$, $\vec{C}$ i $\vec{T}$ vektori u 3D prostoru a t_1, t_2 i t_3 skalari. Prethodni vektorska jednadžba stoga se raspada u tri jednadžbe:

$$\begin{aligned} A_x \cdot t_1 + B_x \cdot t_2 + C_x \cdot t_3 &= T_x \\ A_y \cdot t_1 + B_y \cdot t_2 + C_y \cdot t_3 &= T_y \\ A_z \cdot t_1 + B_z \cdot t_2 + C_z \cdot t_3 &= T_z \end{aligned}$$

odnosno matrično zapisano:

$$\begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ A_z & B_z & C_z \end{bmatrix} \cdot \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}. \tag{1.1}$$

Rješenje sustava tada je:

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ A_z & B_z & C_z \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}. \tag{1.2}$$

U našem konkretnom slučaju ovo se svodi na rješavanje sustava:

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 3 \\ 0 & 0 & 8 \\ 0 & 0 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}. \tag{1.3}$$

Ovaj sustav nema jednoznačno rješenje – matrica koju trebamo invertirati je singularna (determinanta joj je nula jer sadrži redak nula – provjerite to za vježbu). Problem je upravo u trećem retku koji dolazi iz jednadžbe:

$$0 \cdot t_1 + 0 \cdot t_2 + 0 \cdot t_3 = 0$$

što je sustav koji ne ograničava apsolutni ništa – bilo koji t_1, t_2 i t_3 ga zadovoljavaju pa efektivno imamo dvije jednadžbe s tri nepoznanice i od tuda ovaj problem. Međutim, umjesto tog izraza znamo da baricentrične koordinate moraju zadovoljavati izraz:

$$t_1 + t_2 + t_3 = 1. \tag{1.4}$$

Prepišimo stoga sustav jednadžbi (1.3) tako da umjesto originalne treće jednadžbe koristimo jednadžbu (1.4). Slijedi da je potrebno riješiti:

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 3 \\ 0 & 0 & 8 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}. \tag{1.5}$$

Ovaj sustav ima jednoznačno rješenje: $t_1 = \frac{1}{4}$, $t_2 = \frac{1}{4}$ i $t_3 = \frac{1}{2}$. Uporabom biblioteke koju ste napravili napišite program koji će provesti ovaj izračun i ispisati rezultat.

1.2.4 Izračun reflektiranog vektora

Pogledajte u knjizi u poglavlju 2 primjer izračuna reflektiranog vektora oko zadanog vektora. Napišite uporabom razvijene biblioteke program koji korisniku omogućava unos oba vektora te koji računa i ispisuje reflektirani vektor. Program treba prihvaćati i 2D vektore i 3D vektore.

1.3 Demonstracija razvijene biblioteke

U nastavku slijedi nekoliko zadataka koje je potrebno riješiti uporabom razvijene biblioteke.

1. Napišite program koji će izračunati i na zaslon ispisati $\vec{v}_1$, s, $\vec{v}_2$, $\vec{v}_3$, $\vec{v}_4$, $\mathbf{M}_1$, $\mathbf{M}_2$ te $\mathbf{M}_3$.
   - $\vec{v}_1 = (2\vec{i} + 3\vec{j} - 4\vec{k}) + (-1\vec{i} + 4\vec{j} - 3\vec{k})$.
   - $s = \vec{v}_1 \cdot (-1\vec{i} + 4\vec{j} - 3\vec{k})$.
   - $\vec{v}_2 = \vec{v}_1 \times (2\vec{i} + 2\vec{j} + 4\vec{k})$ gdje je $\times$ oznaka za vektorski produkt.
   - $\vec{v}_3 = \mathit{normalize}(\vec{v}_2)$, tj. v_3 treba biti normirani vektor dobiven iz vektora v_2.
   - $\vec{v}_4 = -\vec{v}_2$, tj. v_4 treba biti vektor suprotnog smjera u odnosu na vektor v_2.
   - $\mathbf{M}_1 = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 4 & 5 & 1 \end{bmatrix} + \begin{bmatrix} -1 & 2 & -3 \\ 5 & -2 & 7 \\ -4 & -1 & 3 \end{bmatrix}$.
   - $\mathbf{M}_2 = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 4 & 5 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -3 \\ 5 & -2 & 7 \\ -4 & -1 & 3 \end{bmatrix}^T$.
   - $\mathbf{M}_3 = \begin{bmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{bmatrix}^{-1}$.
2. Napišite program koji će korisniku dopustiti da unese podatke o sustavu tri jednadžbe s tri nepoznanice (nije potrebno raditi parsiranje ulaza; pretpostavite da su varijable uvijek x, y i z, te da će korisnik unijeti redom podatke o prvoj jednadžbi pa podatke o drugoj jednadžbi te konačno podatke o trećoj jednadžbi). Program potom mora ispisati rješenje tog sustava. Primjerice, ako korisnik redom unese 1, 1, 1, 6, −1, −2, 1, −2, 2, 1, 3, 13, sustav koji je time definiran je:

$$\begin{aligned} 1 \cdot x + 1 \cdot y + 1 \cdot z &= 6 \\ -1 \cdot x + -2 \cdot y + 1 \cdot z &= -2 \\ 2 \cdot x + 1 \cdot y + 3 \cdot z &= 13 \end{aligned}$$

   a program treba ispisati da je rješenje $[x, y, z] = [1, 2, 3]$.
3. Napišite program koji će korisniku dopustiti da unese podatke o tri vrha trokuta ($\vec{A}$, $\vec{B}$ i $\vec{C}$) te dodatnoj točki $\vec{T}$ u 3D prostoru (potrebno je unijeti x, y i z koordinatu za svaki vrh odnosno točku). Program treba na zaslon ispisati baricentrične koordinate točke T s obzirom na zadani trokut. Napomena: primijenite direktno prethodno opisani postupak i zanemarite moguće probleme uslijed loše odabranih vrhova odnosno zadane točke.

Laboratorijska vježba 2

Prvi program u OpenGL-u

Jednostavan program koji koristi tehnologiju OpenGL prikazan je i opisan u knjizi u poglavlju 1. Proučite to poglavlje do dijela koji se bavi animacijom (taj dio više ne trebate čitati). Proučite program prikazan u ispisu 1.1; prepišite ga, prevedite i pokrenite. Ako želite, slobodno možete umjesto programskog jezika C ili C++ koristiti neki drugi programski jezik za koji postoji biblioteka koja omogućava uporabu OpenGL-a (primjerice, za Javu postoji JOGL). Ono što je važno jest da za crtanje doista koristite primitive OpenGL-a odnosno potporni skup naredbi koji je dostupan kroz *gl*, *glu* i po potrebi *glut*.

Tijekom ove vježbe morali biste naučiti odgovore na pitanja navedena u nastavku. Ako bilo što od navedenoga nije jasno, svakako se konzultirajte prije no što krenete sa sljedećom vježbom.

2.1 Pitanja

1. Čemu služi naredba `glutInit`?
2. Čemu služi naredba `glutInitDisplayMode`?
3. Čemu služi naredba `glutInitWindowSize`?
4. Čemu služi naredba `glutInitWindowPosition`?
5. Čemu služi naredba `glutCreateWindow`?
6. Čemu služi naredba `glutDisplayFunc`?
7. Čemu služi naredba `glutReshapeFunc`?
8. Čemu služi naredba `glutMainLoop`?
9. Koja je strukture metode `display`, odnosno što se u njoj radi?
10. Koja je strukture metode `reshape`, odnosno što se u njoj radi?
11. Kako se u OpenGL-u crtaju točke, odnosno koji se idiom (slijed naredbi) za to koristi?
12. Kako se u OpenGL-u crta obrub trokuta, odnosno koji se idiom (slijed naredbi) za to koristi?
13. GLUT korisnicima omogućava pisanje programa koji reagira na različite događaje. Koji su događaji podržani?
14. Na koji se način može specificirati dio koda koji je potrebno izvršiti kada korisnik pritisne tipku miša?
15. Na koji se način može specificirati dio koda koji je potrebno izvršiti kada korisnik pritisne tipku na tipkovnici?

2.2 Zadatak

Modificirajte program prikazan ispisom 1.1 tako da omogući crtanje većeg broja ispunjenih trokuta u odabranoj boji. Za crtanje popunjenog trokuta možete koristiti primitiv `GL_TRIANGLES`. Program treba pamtiti trenutnu (aktivnu) boju, i tu boju treba korisniku prikazati u gornjem desnom uglu kao mali kvadratić dimenzija 5×5 koji je ispunjen tom bojom. Za crtanje popunjenog kvadratića možete koristiti primitiv `GL_QUADS`; prije navođenja prve točke zadajte boju kojom kvadrat treba biti obojan (naredba `glColor3f`). Pozadina praznog prozora treba biti ispunjena bijelom bojom.

Zadavanje trokuta u programu potrebno je obavljati mišem. Program za svaki trokut koji je korisnik tako zadao treba pamtiti koordinate triju vrhova te boju koja je u trenutku zadavanja trokuta bila postavljena kao trenutna. Napravite pomoćnu strukturu podataka koja će Vam ovo omogućiti. Tako definirane obojane trokute spremajte u listu obojanih trokuta. Prilikom crtanja trokuta na ekranu, trokute crtajte redosljedom kojim su dodavani u listu.

Dodavanje novog trokuta obavlja se na sljedeći način. Kada korisnik prvi puta klikne negdje na površinu prozora, zapamti se pozicija prvog vrha. Potom se prati pomicanje miša, i iscrtava linija u trenutnoj boji od pokazivača miša do zapamćene prve točke. U trenutku kada korisnik klikne drugi puta mišem, pamti se i drugi vrh trokuta. Sada se opet prate pomaci mišem i pri svakom pomaku crta privremeni trokut određen s prethodna dva zapamćena vrha i trenutnom pozicijom pokazivača miša kao trećim vrhom. Kada korisnik klikne mišem treći puta, pamti se i treći vrh trokuta te se temeljem zapamćene tri lokacije i trenutne boje stvara novi trokut koji se dodaje u listu obojanih trokuta.

Program treba podržavati rad s šest boja: crvenom, zelenom, plavom, cijan, žutom te magentom. Pritiskom na tipku `n` trenutna boja se mijenja u sljedeću a pritiskom na tipku `p` trenutna boja se mijenja u prethodnu. U logičkom smislu, korisnik ima definiran niz od ovih šest boja upravo u tom redosljedu i uporabom uvih dviju tipki ciklički prolazi kroz taj slijed. Promjena trenutne boje treba odmah biti popraćena i vizualnom indikacijom.

2.3 Organizacija programa

Organizirajte program tako da postoji zasebni objekt (odnosno objekti) koji čine stanje programa odnosno model podataka, te zaseban dio koji se bavi crtanjem – takva organizacija upravo je prirodna za OpenGL. Stanje Vašeg programa odnosno model podataka čine strukture podataka koje omogućavaju pamćenje trokuta, lista obojanih trokuta, trenutno odabrana boja, širina i visina samog prozora, je li korisnik već dodao prvu ili drugu točku za definiranje novog trokuta i slično.

Obrada svakog događaja treba biti napravljena tako da promijeni model podataka (evidentira novu trenutnu boju, doda trokut u listu trokuta i slično) i potom pozove glutPostRedisplay. Time će openGL pozvati funkciju koju ste zadužili i registrirali za crtanje slike po površini prozora (tipično metoda display). U toj metodi trebate obrisati sliku, proći kroz listu dodanih trokuta i sve ih nacrtati, pogledati je li u tijeku dodavanje novog trokuta pa ako je, ovisno o tome što je do tada dodano ili povući liniju od prve točke to točke na kojoj je zadnji puta zapamćen pokazivač miša, ili popuniti privremeni trokut definiran sa dvije zadane točke i točkom na kojoj je zadnji puta zapamćen pokazivač miša, te konačno trebate još nacrtati u gornjem desnom uglu kvadratić u trenutnoj boji. Ako ste koristili dvostruki spremnik (kao što je slučaj u ispisu iz knjige, metodu završavate s glutSwapBuffers(); čime će se ta slika aktivirati i prikazati na površini prozora. Pročitajte u knjizi diskusiju o dvostrukom spremniku.

Laboratorijska vježba 3

Crtanje linija na rasterskim prikaznim jedinicama

Dok razmišljamo o računalnoj grafici i računalno generiranim slikama, lako je zaboraviti da je zaslon na kojem se slika prikazuje diskretan, sastavljen od niza slikovnih elemenata, i da na njemu ne možemo beskonačno precizno nacrtati niti linije, niti krivulje niti apstraktne geometrijske likove. Umjesto toga, crtanje različitih pravaca, krivulja i likova u praksi se svodi na uporabu algoritama koji će brzo i efikasno na ekranu upaliti slikovne elemente koji će dati najbolju aproksimaciju linije koju je korisnik htio nacrtati.

U okviru ove vježbe Vaš je zadatak proučiti na koji se način na rasterskim prikaznim jedinicama mogu efikasno crtati linijski segmenti, te na koji je način moguće ograničiti crtanje na dio podprostora kojim prikazna jedinica raspolaže. Da biste riješili ovaj zadatak, proučite u knjizi poglavlje 4 i to podpoglavlje 4.1 koje opisuje Bresenhamov algoritam za crtanje linijskih segmenata, te unutar poglavlja 8 podpoglavlje 8.8 koje opisuje izvedbu odsijecanja algoritmom Cohen Sutherlanda.

3.1 Pitanja

1. Proučite osnovnu verziju Bresenhamovog algoritma za crtanje linije.

   (a) Taj algoritam će korektno iscrtati liniju samo u slučaju da je nagib linije u rasponu od $0°$ do $45°$. Objasnite zašto?

   (b) Što će taj algoritam nacrtati ako se zada crtanje pravca određenog početnom točkom $(0, 0)$ i završnom točkom $(4, 12)$? Skicirajte rezultat. Objasnite zašto je rezultat takav?

   (c) Što bi taj algoritam nacrtao za pravac iz prethodne točke ako bismo redak u kojem se y-koordinata inkrementira zamijenili retkom u kojem se y-koordinata uvećava za izračunati tangens kuta pod kojim je zadan pravac? Skicirajte rezulata. Objasnite ga.

   (d) Vratimo se na osnovnu verziju algoritma gdje se y-koordinata uvećava za 1. Što će taj algoritam nacrtati ako se zada pravac pod kutem od $-30°$, a što ako se zada pravac čiji je tangens nagiba jednak -3? Slicirajte to na konkretnom primjeru i objasnite rezultat.

2. Objasnite kako se izvodi Bresenhamov algoritam s decimalnim brojevima? Koja je osnovna prednost tog algoritma?

3. Objasnite na koji se način Bresenhamov algoritam s decimalnim brojevima prevodi na cjelobrojnu varijantu? Je li ta inačica povoljnija u odnosu na inačicu s decimalnim brojevima? Argumentirajte Vaš odgovor.

4. Objasnite kako radi algoritam za odsijecanje linija?

3.2 Zadatak

Koristeći OpenGL za crtanje, napišite program koji će korisniku omogućiti crtanje proizvoljnog broja linija. Korisnik treba moći linije zadavati mišem – prvi klik definira početak segmenta a drugi kraj segmenta; u tom trenutku segment se dodaje u listu definiranih segmenata. Svi segmenti iscrtavaju se crnom – nije potrebno omogućiti bojanje linija. Prilikom crtanja konačne slike na ekranu, program treba konzultirati dvije pomoćne *boolean* varijable: kontrola i odsijecanje.

Ako je kontrola==**false**, program Vašom implementacijom Bresenhamovog algoritma treba nacrtati svaki segment koji je zapamćen u listi definiranih segmenata. Ako je kontrola==**true**, program Vašom implementacijom Bresenhamovog algoritma treba nacrtati svaki segment koji je zapamćen u listi definiranih segmenata, te za svaki takav segment ugrađenim primitivima OpenGL-a treba nacrtati segment koji je paralelan zadanom i desno je od njega na udaljenosti 4 slikovna elementa (desno kada biste se kretali segmenom linije od početne prema konačnoj točki). Razmislite kako ćete izračunati početnu i konačnu točku takvog segmenta. Ove paralelne segmente crtajte crvenom bojom.

Varijabla odsijecanje omogućava definiranje podprostora u kojem se iscrtavaju linije. Ako je varijabla odsijecanje==**false**, linije se iscrtavaju na čitavoj površini prozora. Ako je odsijecanje==**true**, aktivira se podprostor širine pola prozora i visine pola prozora koji je centriran u prozoru. Vaša implementacija Bresenhamovog algoritma treba provjeravati je li definiran podprostor u kojem se icsrtava, te ako je, treba iscrtati samo dio segmenta koji se nalazi unutar tog podprostora. To treba implementirati na način da se segmenti koji su u cijelosti izvan tog podprostora uopće ne crtaju (naprosto se odbace), a u suprotnom se računa početak i kraj podsegmenta koji je u cijelosti u tom podprostoru i samo se to iscrtava. Ako je ovaj podprostor aktivan, tada se kao prvi korak u iscrtavanju slike na praznu površinu prozora najprije zelenom bojom treba iscrtati rub tog područja (koristite za to OpenGL primitiv za crtanje linije). Ako je istovremeno uz odsijecanje aktivna i varijabla kontrola, odsijecanje se provodi samo nad crtanjem linija Vašom implementacijom Bresenhamovog algoritma; paralelne linije koje se u tom slučaju crtaju trebaju biti u cijelosti nacrtane.

Konačno, program treba pratiti pritiske tipaka na tipkovnici. Ako korisnik pritisne tipku `o`, program treba invertirati trenutnu vrijednost varijable odsijecanje (i naravno osvježiti prikaz). Ako korisnik pritisne tipku `k`, program treba invertirati trenutnu vrijednost varijable kontrola (i osvježiti prikaz).

Laboratorijska vježba 4

Crtanje i popunjavanje poligona

Crtanje i popunjavanje poligona možda je i najčešći zadatak u računalnoj grafici. Danas je ta operacija direktno prisutna u obliku sklopovske implementacije svih modernih grafičkih kartica. Kada govorimo o poligonima, postoji nekoliko zadataka koji mogu biti zadani.

- Crtanje poligona – temeljem zadanih vrhova poligona treba nacrtati poligon; pri tome se misli samo nacrtati obrub poligona, tj. linijama spojiti susjedne vrhove. Ovo je u OpenGL-u direktno podržano preko primitiva GL_LINE_LOOP.
- Popunjavanje poligona – temeljem zadanih vrhova poligona treba popuniti poligon, odnosno njegovu unutrašnjost ispuniti zadanom bojom. Ovo je u OpenGL-u direktno podržano preko primitiva GL_POLYGON.
- Ispitivanje smjera u kojem su zadani vrhovi poligona – u smjeru kazaljke na satu ili u smjeru suprotnom od smjera kazaljke na satu.
- Ispitivanje vrste poligona – je li poligon konveksan ili konkavan.
- Pronalaženje konveksnog poligona – uz niz vrhova zadanih proizvoljnim poretkom traži se pronalaženje poretka vrhova tako da budu u zadanom smjeru (primjerice, u smjeru kazaljke na satu).
- Ispitivanje odnosa točke i poligona – temeljem zadanog poligona i zadane proizvoljne točke potrebno je utvrditi je li zadana točka unutar poligona, na bridu poligona ili izvan poligona.
- Itd.

U okviru ove vježbe, pozabavit ćemo se nekim od tih pitanja. Kako bi se pripremili za vježbu, Vaš je zadatak pažljivo proučiti u knjizi poglavlje 4 i to podpoglavlje 4.2 koje daje matematički tretman poligona i opisuje niz postupaka koji služe crtanju poligona, popunjavanju poligona te otkrivanju različitih svojstava poligona.

4.1 Pitanja

1. Koja je razlika između konveksnog i konkavnog poligona?
2. Što znači da je redosljed vrhova u smjeru kazaljke na satu?
3. Kako se za zadani poligon može ispitati je li mu redosljed vrhova u smjeru kazaljke na satu?
4. Kako se za zadani poligon može ispitati je li mu redosljed vrhova u smjeru suprotnom od smjera kazaljke na satu?

5. Neka je poligon zadan tako da su mu vrhovi zadani u smjeru kazaljke na satu.

   (a) Kako možemo ispitati je li točka T unutar poligona?

   (b) Kako možemo ispitati je li točka T izvan poligona?

   (c) Kako možemo ispitati je li točka T na nekom bridu poligona?

6. Neka je poligon zadan tako da su mu vrhovi zadani u smjeru suprotnom od smjera kazaljke na satu.

   (a) Kako možemo ispitati je li točka T unutar poligona?

   (b) Kako možemo ispitati je li točka T izvan poligona?

   (c) Kako možemo ispitati je li točka T na nekom bridu poligona?

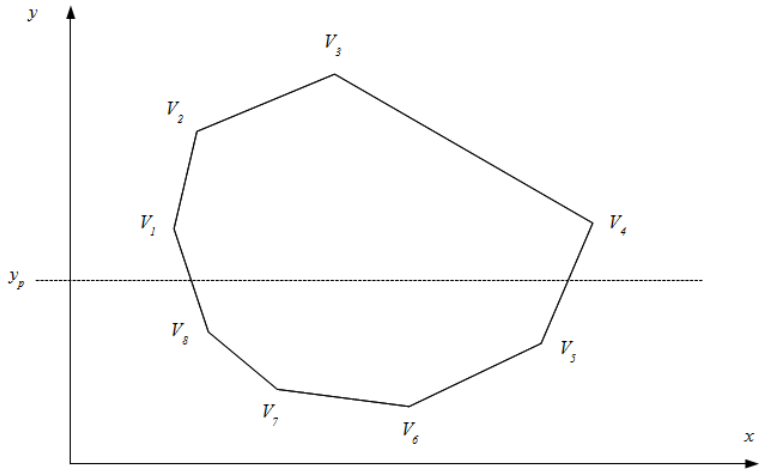
7. Neka je poligon zadan tako da su mu vrhovi zadani na nepoznat način (ne znamo jesu li u smjeru kazaljke na satu ili u smjeru suprotnom od smjera kazaljke na satu). Prilikom odgovaranja na sljedeća podpitanja ne smijete raditi eksplicitnu provjeru načina na koji su bridovi zadani.

   (a) Kako možemo ispitati je li točka T unutar poligona?

   (b) Kako možemo ispitati je li točka T izvan poligona?

   (c) Kako možemo ispitati je li točka T na nekom bridu poligona?

Sljedeća pitanja se odnose na bojanje poligona i sliku 4.1.



Slika 4.1: Primjer poligona i postupak popunjavanja

1. Koliko će se presjecišta računati s pravcem $y = y_p$?

2. Koliko poligon ima lijevih bridova? Kako se definiraju lijevi bridovi?

3. Koliko poligon ima desnih bridova? Kako se definiraju desni bridovi?

4. Koliko će se puta ažurirati točka L a koliko puta točka D nakon što se postave na svoje inicijalne vrijednosti? Koje su uopće njihove inicijalne vrijednosti?

5. Pretpostavite da je poligon zadan tako da mu je redosljed vrhova suprotan od smjera kazaljke na satu. Kako bi se tada modificirao postupak bojanja?

4.2 Zadatak

Koristeći OpenGL za crtanje, napišite program koji će korisniku omogućiti da mišem definira vrhove poligona i koji će potom korisniku prikazati taj poligon, omogućiti mu da dobije prikaz popunjenog poligona, te će omogućiti korisniku da mišem zadaje točke za koje će program u konzolu ispisivati u kakvom je odnosu točka i poligon. Program treba modelirati tako da prolazi kroz dva stanja: stanje 1 u kojem se poligon definira, te stanje 2 u kojem korisnik zadaje točke a sustav ispisuje odnos točaka i definiranog poligona. Radom programa također trebaju upravljati dvije zastavice: popunjavanje te konveksnost.

Inicijalno, program se treba nalaziti u stanju 1. Korisnik počinje definirati poligon tako što klikne na lokaciju prvog vrha. Potom, kako pomiče pokazivač miša, na ekranu se crta spojnica između prvog vrha i trenutne pozicije na kojoj se nalazi pokazivač miša sve do trenutka kada korisnik po drugi puta ne klikne mišem; u tom trenutku poznate su već dvije točke poligona i sada se prilikom pomicanja pokazivača miša iscrtava poligon koji je definiran s te dvije točke i točkom na kojoj se nalazi pokazivač miša. Kada korisnik po treći puta klikne, fiksira se i treća točka poligona te se daljnjim pomicanjem pokazivača miša iscrtava poligon koji je određen s te tri točke i četvrtom koja odgovara lokaciji pokazivača miša. I tako se postupak nastavlja sve dok korisnik želi dodavati nove vrhove. Broj točaka pri tome unaprijed ne smije biti ograničen.

Što se točno na ekranu iscrtava, ovisi o vrijednosti booleove zastavice popunjavanje. Ako je vrijednost te zastavice postavljena na **false**, vrhovi poligona povezuju se linijama. Za ovo povezivanje koristite OpenGL-ov primitiv GL_LINE_LOOP. Ako je vrijednost te zastavice postavljena na **true**, treba se prikazivati popunjeni poligon (crnom) a ne njegov obrub. Proučite u knjizi algoritam za popunjavanje konveksnih poligona, razmotrite kako taj algoritam radi i implementirajte ga. OpenGL smijete koristiti isključivo za bojanje pojedinih slikovnih elemenata ili crtanje linija kada popunjavate poligon (primitiv GL_POINTS za pojedine točke ili GL_LINES za linije); u ovu svrhu nije dozvoljeno koristiti gotove primitive unutar OpenGL-a koji bi posao popunjavanja napravili za Vas.

Zastavica konveksnost regulira prihvaćanje novih vrhova poligona. Kada korisnik klikne mišem s namjerom da doda novi vrh poligona, Vaš program treba najprije provjeriti vrijednost zastavice konveksnost. Ako je ta zastavica postavljena na **false**, vrh se bezuvjetno prihvaća. Međutim, ako je ta zastavica postavljena na **true**, pokreće se provjera kojoj je cilj utvrditi bi li dodavanjem tog vrha poligon i dalje ostao konveksan. Ako je odgovor potvrdan, vrh se prihvaća. U suprotnom se vrh ne prihvaća, i u konzolu se ispisuje odgovarajuća poruka. Zastavica konveksnost dodatno regulira i pozadinsku boju kojom se popunjava slika (ne poligon!); ako je zastavica postavljena na **true**, pozadinska boja treba biti zelena a ako je zastavica postavljena na **false**, pozadinska boja treba biti bijela. Linije kojima se prikazuje obrub poligona (ako se prikazuje) te popunjavanje poligona (ako se popunjava) treba biti crnom bojom.

Promjena stanja kao i ovih zastavica obavlja se preko tipkovnice, kako je opisano u nastavku.

Tipka	Funkcija
`k`	Promjena vrijednosti zastavice konveksnost. Pri tome se ne smije dozvoliti postavljanje vrijednosti na **true** ako u tom trenutku zadani poligon već nije konveksan – u tom slučaju samo treba ispisati poruku u konzolu da promjena zastavice nije moguća. Omogućeno je samo u stanju 1.
`p`	Promjena vrijednosti zastavice popunjavanje. Omogućeno je samo u stanju 1.
`n`	Ciklički prelazak u sljedeće stanje. Iz stanja 1 prelazi se u stanje 2. Iz stanja 2 prelazi se u stanje 1 i brišu se svi podatci o prethodno definiranom poligonu; obje zastavice postavljaju se na početne vrijednosti **false**.

U stanju 2 svaki puta kada korisnik klikne mišem, potrebno je ispitati je li točka na kojoj se nalazi pokazivač miša unutar poligona, na bridu poligona ili izvan poligona. Podatke o točki te utvrđeni status potrebno je ispisati u konzolu. Algoritam za ispitivanje odnosa točke i poligona treba temeljiti na opisu danom u knjizi, u podpoglavlju 4.2.3.

4.2.1 Dodatna pitanja

1. Isprobajte na primjeru kako će algoritam za popunjavanje poligona napraviti popunjavanje ako je zadani poligon konkavan. Objasnite rezultat. Znate li sada rukom na papiru objasniti za proizvoljni konkavan poligon kako će izgledati rezultat popunjavanja ovim algoritmom?

2. Isprobajte na primjeru kako će algoritam koji ste trebali implementirati za ispitivanje odnosa točke i poligona klasificirati točke ako mu se zada konkavan poligon? Hoće li taj algoritam baš za sve raditi krivo? Objasnite.

3. Razmislite biste li problem utvrđivanja odnosa točke i konkavnog poligona riješiti ispucavanjem polupravca iz zadane točke (u bilo kojem smjeru, a možda je najjednostavnije vodoravno u desno) te brojanjem sjecišta s bridovima poligona na koje se naiđe u tom smjeru? Skicirajte pseudokod takvog algoritma.

Laboratorijska vježba 5

3D tijela

Prvi korak u upoznavanju s 3D svijetom računalne grafike jest modeliranje objekata. Dok smo u 2D svijetu često zadovoljni s linijama, kružnicama i drugim krivuljama, u 3D svijetu naješće se bavimo modeliranjem i prikazivanjem različitih tijela. Postoji više načina kako 3D tijelo može biti zadano. Najjednostavnija tijela imaju jasan matematički opis. Tako primjerice, kuglu radijusa R čije je središte u prostoru smješteno u točku (x_c, y_c, z_c) matematički možemo opisati izrazom:

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = R^2.$$

Međutim, nema jednostavnog matematičkog izraza kojim bismo mogli opisati Zagrebačku katedralu ili pak Vašeg omiljenog lika iz igre *Doom* ili *Unreal tournament*. Stoga je u današnje doba velik naglasak stavljen na modeliranje tijela zadavanjem njegovog oplošja. Primjerice, oplošje najobičnije kocke možemo zamisliti da je sastavljeno od šest kvadrata koji u prostoru tako posloženi da u potpunosti zatvaraju volumen kocke. Ovaj pristup primjenjiv je i na modeliranje složenijih tijela – sve što je potrebno jest uzeti veći dio dovoljno malih "pločica" koje se u prostoru poslože tako da aproksimiraju čitavo oplošje objekta. Kod jednostavnih tijela površine tih pločica mogu biti i velike – primjer je upravo kocka ili kvadar proizvoljnih dimenzija za koje je uvijek dovoljno samo šest pločica. Oplošje kugle na ovaj način nikada nećemo uspjeti savršeno opisati, ali uz dovoljan broj pločica rezultat može biti sasvim zadovoljavajući.

Spomenute pločice u praksi mogu biti poligoni, pri čemu je svaki poligon zadan s određenim brojem vrhova koji svi leže u istoj ravnini, ili pak mogu biti površine modelirane na različite načine (poput Bezierovih krpica koje spadaju u parametarske plohe). Međutim, rad s parametarskim plohama je računski vrlo zahtjevan – linearno opisane površine su bitno jednostavnije, tako da se one dosta koriste. Kod linearnih ploha koje su zadane kao poligoni, situacija u kojoj se poligon definira s više od 3 točke mogu biti problematične – što ako četiri ili više zadanih vrhova poligona naprosto ne leže u ravnini? Kako bi se riješio ovaj problem, vrlo često se u praksi koriste najjednostavniji mogući poligoni – trokuti, i njih ćemo koristiti u ovoj vježbi.

Zadamo li tri točke koje međusobno nisu kolinearne, one sigurno leže u ravnini. Zadavanjem tri točke nismo međutim u potpunosti definirali sve parametre ravnine. Prije no što nastavite dalje, pažljivo prođite kroz poglavlje 2 u knjizi, a posebice kroz podpoglavlje 2.5. Vidjet ćete da ono što nam još nedostaje jest normala ravnine – tri točke ne mogu istovremeno odrediti i jednadžbu ravnine i normalu ravnine (odnosno smjer vektora normale ravnine); tri točke su nam dovoljne kako bismo odredili sve točke prostora koje leže u toj ravnini, i ništa više. Da bismo odredili normalu ravnine (a time i smjer u kojem ona gleda), trebamo još nekakav podatak. To bi trebalo biti jasno vidljivo i iz činjenice da je implicitni oblik jednadžbe ravnine u 3D prostoru jednak

$$ax + by + cz + d = 0 \tag{5.1}$$

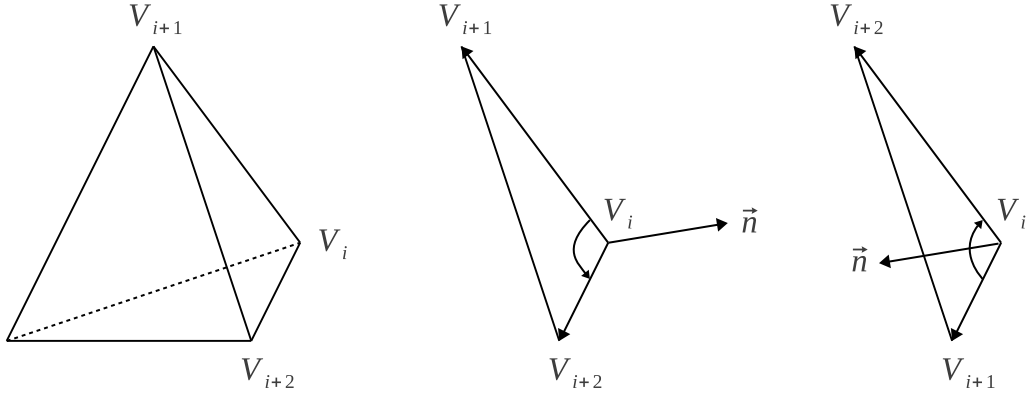
što je jednadžba s četiri nepoznanice. Ako znamo samo tri točke koje leže u toj ravnini (pa time zadovoljavaju prethodni izraz) nemamo dovoljno informacija kako bismo riješili sustav i odredili sve četiri nepoznanice: a, b, c i d.

Ovdje u pomoć priskaču konvencije kojih se želimo držati prilikom opisivanja oplošja tijela trokutima. Svaki trokut koji čini oplošje tijela (odnosno ravnina kojoj on pripada) dijeli prostor u dva podprostora. U jednom od ta dva podprostora smješten je i volumen tijela čije oplošje opisujemo dok u drugom poluprostoru nema ničega. Ovo dakako vrijedi ako je tijelo koje opisujemo konveksno; ako nije, onda izjava sigurno vrijedi barem za dio podprostora koji je s jedne i druge strane samog trokuta – s jedne strane je volumen tijela a s druge strane praznina; kada to ne bi bio slučaj, onda promatrani trokut ne bi mogao biti dio oplošja. Konvencija koje se ovdje želimo držati je sljedeća: želimo da normale ravnina za sve trokute koji čine oplošje tijela konzistentno gledaju ili u unutrašnjost tijela ili prema vanjštini tijela. Potpuno je nebitno odabere li se jedno ili drugo, važno je samo da odabir vrijedi za sve trokute i da ga unaprijed znamo. Normale trokuta za tijela koja ćemo koristiti u ovoj vježbi gledat će prema vanjštini tijela.

No kako ćemo određivati normalne? Pretpostavimo da je trokut zadan s tri točke: V_i, V_{i+1} i V_{i+2} (upravo tim redosljedom). Fiksiramo li točku V_i, možemo izračunati dva vektora koja razapinju ovu ravninu: $V_{i+1} - V_i$ i $V_{i+2} - V_i$. Prema dogovoru ćemo normalu računati kao vektorski produkt ovih dvaju vektora (i to upravo redosljedom kojim smo ih napisali). Dakle, normalu ravnine razapete s tri točke V_i, V_{i+1} i V_{i+2} (zadane tim redosljedom) računat ćemo kao vektorski produkt:

$$\vec{n} = (V_{i+1} - V_i) \times (V_{i+2} - V_i). \tag{5.2}$$

Zamjena redosljeda točaka V_i, V_{i+1} i V_{i+2} može rezultirati dobivanjem kolinearnog vektora (vektora koji će biti jednak po normi ali suprotnog smjera), pa na to svakako treba pripaziti. Prema pravilu desne ruke vektorski produkt jasno određuje smjer u kojem će vektor normale biti orijentiran. I sada imamo sve potrebno da izvedemo zaključak o tome kako vrhovi moraju biti zadani. Prethodno smo rekli kako želimo da normala trokuta gleda u vanjštinu tijela; potom smo rekli da trokut zadajemo s tri vrha V_i, V_{i+1} i V_{i+2} (tim redosljedom) i konačno da normalu računamo vektorskim produktom prema izrazu (5.2). Ako je to istina i ako trokut promatramo iz onog poluprostora u koji pokazuje normala, vrhovi V_i, V_{i+1} i V_{i+2} moraju biti tako zadani da generiraju obilazak koji je u smjeru suprotnom od smjera kazaljke na satu. Ovo je ilustrirano je slici 5.1. Ako je tijelo koje opisujemo konveksno (poput kocke), pravilo je još jednostavnije: gledano izvan tijela prema tijelu trokut mora biti zadan tako da obilaskom njegovih vrhova radimo gibanje koje je u smjeru suprotnom od smjera kazaljke na satu. Primjerice, tijelo prikazano na slici 5.1 je konveksno pa je lako uočiti direktno sa slike da ovaj zaključak također vrijedi.



Slika 5.1: Važnost redosljeda kojim su zadani vrhovi tijela. Lijevo: primjer nepravilnog tetraedra s tri istaknuta vrha. Sredina: izdvojena stranica. Uz prikazani redoslijed normala gleda prema vanjštini tijela. Desno: izdvojena stranica ali uz drugačiji redoslijed vrhova. Uz prikazani redoslijed normala gleda prema unutrašnjosti tijela.

Ponovimo još jednom: zaključak koji je iznesen vrijedi samo ako vrijede i sve prethodno iznesene ograde: želimo da sve normale budu usmjerene konzistentno, želimo da taj smjer bude prema vanjštini tijela, želimo raditi s trokutima koje zadajemo preko tri vrha i konačno, normalu računamo upravo prema izrazu (5.2). Ako bismo odstupili od ovih zahtjeva i promijenili jedan od njih, primjerice, da želimo da sve normale gledaju u unutrašnjost tijela, i naš bi se zaključak promijenio – vrhovi bi morali

biti zadani tako da, ako ih promatramo iz vanjštine tijela, sada činimo obilazak koji odgovara smjeru kazaljke na satu. Međutim, ako promijenimo priču još malo pa kažemo da trokut promatramo iz unutrašnjosti tijela (u koju sada pokazuju i normale), zaključak bi opet bio da vrhovi moraju biti zadani tako da njihovim obilaskom činimo gibanje koje je u smjeru suprotnom od smjera kazaljke na satu. Za konzistentnog ovoga obično se brinu programi koji korisniku omogućavaju 3D modeliranje objekata.

Jednom kad smo odredili normalu, još nam ostaje do kraja odrediti sve koeficijente jednadžbe ravnine. U jednadžbi ravnine, koeficijenti a, b i c upravo odgovaraju komponentama normale. Ako je $\vec{n} = (n_x, n_y, n_z)$, tada vrijedi $a = n_x$, $b = n_y$, $c = n_z$ (pogledajte u knjizi izvod izraza (2.34)). Koeficijent d tada možemo odrediti uporabom bilo kojeg od vrhova. Naime, s obzirom da smo izračunali a, b i c (jer smo izračunali normalu), jedina nepoznanica jest d. Slijedi:

$$d = -ax - by - cz. \tag{5.3}$$

S obzirom da znamo čak tri točke koje zadovoljavaju zadanu jednadžbu ravnine (a to su V_i, V_{i+1} i V_{i+2}), d možemo odrediti preko bilo koje od njih; primjrice, vrijedi:

$$d = -aV_{i,x} - bV_{i,y} - cV_{i,z}. \tag{5.4}$$

Uz pretpostavku da je tijelo čije smo oplošje opisali trokutima uz pridržavanje prethodno definirane konvencije konveksno, ispitivanje odnosa točke i tijela provodi se na identičan način kako smo to radili u vježbi s poligonima. Vrijedi sljedeće: ako je tijelo zadano tako da normale svih trokuta kada ih računamo prema izrazu (5.2) gledaju prema vanjštini tijela, to znači da će za proizvoljnu točku $T = (x, y, z)$ vrijednost izraza $ax + by + cz + d$ biti pozitivna ako točka T ne leži u ravnini već je negdje u poluprostoru u koji pokazuje normala, bit će jednaka 0 ako točka T leži u ravnini (jer tada zadovoljava jednadžbu ravnine), i bit će manja od nule ako točka T leži u poluprostoru u koji normala ne pokazuje (odnosno gdje bi pokazivala $-\vec{n}$). No tada je proizvoljna točka T unutar tijela ako je za svaki trokut i pripadnu ravninu vrijednost izraza $ax+by+cz+d$ negativna (drugim riječima, ako točka za svaki trokut oplošja leži ispod ravnine u kojoj se nalazi taj trokut). Ako je točka T za sve trokute oplošja ispod ravnina u kojima se nalaze odnosni trokuti osim za neke za koje je vrijednost izraza jednaka nuli, točka je na oplošju tijela. Konačno, ako postoji trokut za koji je točka iznad njegove ravnine (odnosno za koji je $ax + by + cz + d$ pozitivno), točka je izvan tijela.

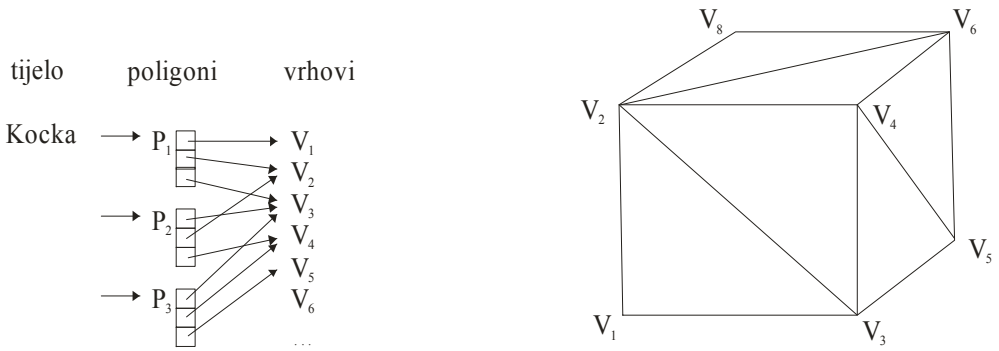
Razmislite kako biste riješili ispitivanje za sličaj konkavnog tijela. Vrijede li tada prethodno opisani zaključci? Biste li zadatak mogli riješiti ispucavanjem polupravca i utvrđivanjem sjecišta s trokutima?

U okviru ove vježbe radit ćemo s modelima tijela koja su definirana zadavanjem njihovog oplošja i to preko niza trokuta. Opis tjela pri tome ćemo čitati iz datoteke. Zapisivanje trodimenzijskih objekata obično je propisano specifikacijama proizvođača programske opreme koji ujedno daju i alate za modeliranje i prikazivanje 3D scena te učitavanje i pohranu tijela u datoteku. Primjeri različitih formata za pohranu 3D tijela su datoteke s ekstenzijom `.3ds`, `.max`, `.dxf`, `.ply`, `.obj`, `.xgl`, `.stl`, `.wrl`, `.iges` i slični. Ovisno o zapisu, zapisani mogu biti trokuti ili poligoni s više vrhova, grupe poligona, boje, teksture, normale u poligonima ili vrhovima, krivulje, površine, način konstrukcije objekata, podaci o sceni, izvori svjetla, podaci o animaciji i slično. U ovoj vježbi bit će korišten zapis `.obj`, odnosno dio tog zapisa. Na mrežnoj stranici kolegija dostupno je nekoliko datoteka s različitim objektima koji će se dalje koristiti u ovim vježbama.

Zapis `.obj` za prikaz tijela koristi trokute. Kako više trokuta može dijeliti isti vrh, u datoteci je najprije dan popis vrhova, a potom slijede definicije trokuta. Opis tijela preko trokuta ilustriran je na slici 5.2.

Konkretan primjer `.obj` datoteke u kojoj je zapisan model kocke prikazan je u nastavku.

```
v 0.0 0.0 0.0
v 0.0 0.0 1.0
v 1.0 0.0 0.0
```



Slika 5.2: Opis tijela trokutima

```
v 1.0 0.0 1.0
v 1.0 1.0 0.0
v 1.0 1.0 1.0
v 0.0 1.0 0.0
v 0.0 1.0 1.0
f 1 3 2
f 3 4 2
f 3 5 4
f 5 6 4
f 5 7 6
f 7 8 6
f 7 1 8
f 1 2 8
f 1 5 3
f 1 7 5
f 2 4 6
f 2 6 8
```

Primjer definira 8 vrhova te 12 trokuta. Vrhovi se nalaze u retcima koji počinju s `v` (od engl. *vertex*); za svaki je vrh navedena x, y i z koordinata. Trokuti su zapisani u retcima koji počinju s `f` (od engl. *face*); svaki trokut navodi indekse pripadnih vrhova pri čemu su indeksi numerirani od 1. Skicirajte ovu kocku i uvjerite se da obilazak vrhova, ako trokut promatrate iz točke koja je izvan kocke, generira gibanje u smjeru suprotnom od smjera kazaljke na satu. Osim slova `v` i `f` moguće je da u `.obj` datotekama naletite i na druga slova i simbole (poput `#` koje označavaju komentar) – sve takve retke zanemarite.

5.1 Pitanja

1. Na koji se način može opisati 3D-tijelo?
2. Kako 3D-tijela opisujemo u ovoj vježbi?
3. Na koji se način temeljem triju točaka dolazi do jednadžbe ravnine? Je li ta jednadžba do kraja definirana?
4. Čime je određen smjer normale trokuta?
5. Koje zahtjeve postavljamo na zapis modela 3D-tijela u ovoj vježbi, odnosno kojih se konvencija pridržavamo?
6. Na koji se način ispituje odnos točke i 3D-tijela ako je tijelo konveksno?

7. Bi li način ispitivanja odnosa točke i 3D-tijela koji je opisan u ovoj vježbi radio za konkavna tijela?

8. Biste li zadatak ispitivanja odnosa točke i konkavnog 3D-tijela mogli riješiti ispucavanjem polupravca i utvrđivanjem sjecišta s trokutima? Objasnite.

9. Kakva je struktura `.obj` datoteke?

5.2 Zadatak

U okviru ove vježbe trebate napraviti program koji će kao argument primiti naziv `.obj` datoteke. Program treba pročitati sadržaj datoteke i u memoriju učitati definirani model tijela. Za svaki trokut program treba izračunati i zapamtiti pripadne koeficijente jednadžbe ravnine. Program potom treba korisniku omogućiti da interaktivno unosi 3D točke; nakon svake unesene točke program treba provjeriti u kakvom su odnosu unesena točka i tijelo te rezultat ispitivanja ispisati na ekran (odnos može biti: točka je unutar tijela, točka je na obodu tijela te točka je izvan tijela).

Program korisniku treba omogućiti i da unosom naredbe `normiraj` (umjesto unosa točke) dobije na zaslon u `.obj` formatu normirani model objekta. Prilikom normiranja modela potrebno je proći kroz sve vrhove modela i utvrditi minimalne i maksimalne iznose svih koordinata: x_{min}, x_{max}, y_{min}, y_{max} te z_{min}, z_{max}. Potom se računa središte objekta kao:

$$\bar{x} = \frac{x_{min} + x_{max}}{2}$$
$$\bar{y} = \frac{y_{min} + y_{max}}{2}$$
$$\bar{z} = \frac{z_{min} + z_{max}}{2}$$

te maksimalni raspon po bilo kojoj od osi:

$$M = \max(x_{max} - x_{min}, y_{max} - y_{min}, z_{max} - z_{min}).$$

Sve je vrhove potrebno translatirati za $(-\bar{x}, -\bar{y}, -\bar{z})$ čime se vrh $V_i = (V_{i,x}, V_{i,y}, V_{i,z})$ preslikava u $(V_{i,x} - \bar{x}, V_{i,y} - \bar{y}, V_{i,z} - \bar{z})$. Konačno, koordinate svih vrhova potrebno je skalirati s $\frac{2}{M}$. Ovim postupkom osigurava se da je raspon koordinata po svim osima za model u rasponu ne većem od $[-1, 1]$ (a za onu os za koju je tijelo izvorno imalo najveći raspon bit će upravo $[-1, 1]$). Nakon normalizacije, Vaš bi program trebao ispisati na zaslon model u `.obj` formatu.

Konačno, ako korisnik umjesto unosa točke zada naredbu `quit`, program treba prestati s radom. U ovom zadatku nije potrebno raditi nikakav grafički prikaz odnosno vizualizaciju učitanog objekta; to je tema sljedećih vježbi.

Za modeliranje tijela prijedlog je da napravite tri razreda: Vertex3D koji pamti koordinate vrha, Face3D koji predstavlja jedan trokut i pamti indekse pridruženih vrhova u cjelobrojnom polju indexes (uz indeksiranje vrhova od nule), te razred ObjectModel koji predstavlja model jednog tijela i sadrži polje vrhova (primjerci razreda Vertex3D) te polje trokuta (primjerci razreda Face3D). U razred ObjectModel trebali biste dodati još tri metode:

- ObjectModel copy(); koja vraća kopiju modela,
- String dumpToOBJ(); koja vraća OBJ reprezentaciju čitavog modela u obliku jednog stringa te
- **void** normalize(); koja modificira trenutni objekt tako da ga normalizira.

Napomena

`.obj` datoteke s modelima različitih objekata (uključivo i datoteku `kocka.obj`) dostupne su na web stranicama kolegija.

Laboratorijska vježba 6

Projekcije

U prethodnoj vježbi upoznali smo se modeliranjem 3D tijela preko opisivanja njegovog oplošja. Također, pogledali smo jedan primjer formata datoteke koji služi za pohranu takvog opisa – format `.obj` datoteke. U ovoj vježbi Vaš je zadatak tijelo koje je pohranjeno u datoteci prikazati na zaslonu; pri tome u okviru ove vježbe trebate prikazati samo žičani model tijela.

Krenimo redom. Bez ikakvog dodatnog podešavanja, OpenGL koristi implicitno definirani volumen pogleda koji je kocka koja se po x-koordinati, y-koordinati i z-koordinati proteže od -1 do $+1$. Od ove kocke važno je izdvojiti njezine dvije plohe, obje paralelne s xy ravninom:

- ploha paralelna s xy ravninom koja leži na $z = -1$ – to je bliža ploha, i vrijednost $z = -1$ još se označava i sa z_{near};
- ploha paralelna s xy ravninom koja leži na $z = +1$ – to je dalja ploha, i vrijednost $z = +1$ još se označava i sa z_{far}.

Na bližoj plohi razapet je dvodimenzijski koordinatni sustav čije su koordinatne osi paralelne s izvornim x- i y- osima, samo što su pomaknute na $z = z_{near}$. Raspon tog 2D koordinatnog sustava je po obje osi i dalje od -1 do $+1$.

Pogledajmo sada što se događa kada OpenGL-u kažemo da nacrta neki slikovni element. Primerice, neka se crta na sljedeći način:

```
1    glBegin(GL_PIXELS);
2    glVertex3f(x,y,z);
3    glEnd();
```

Ako je vrijednost x koordinate, y koordinate ili z koordinate izvan volumena pogleda koji je određen granicama $-1 \leq x \leq 1$, $-1 \leq y \leq 1$ i $-1 \leq z \leq 1$, slikovni element će biti ignoriran – ništa se neće nacrtati. Ako slikovni element leži unutar volumena pogleda, daljnja obrada ovisi o tome je li uključena uporaba z-spremnika ili nije. Ako nije uključena, točka će se preslikati u točku s koordinatama (x, y, z_{near}) – dakle, konceptualno, preslikat će se na bližu plohu i tu će stvarati sliku. Ako je uporaba z-spremnika omogućena, tada će se pogledati kolika je udaljenost te točke od bliže plohe i je li već prethodno na koordinate (x, y) nacrtana neka točka. Ako prethodno niša nije nacrtano na koordinatama (x, y), na bližoj plohi nacrtat će se zadani slikovni element (odnosno točka (x, y, z_{near})) i u z-spremniku će se zapamtiti njezina izvorna udaljenost od bliže plohe. Ako je već nešto nacrtano, tada će se u z-spremniku pogledati na kojoj se je udaljenosti od bliže plohe nalazila ta točka. Novi slikovni element precrtat će se preko starog samo ako je za stari slikovni element u z-spremniku zapisana veća udaljenost no što je udaljenost novog slikovnog elementa, i tada će se udaljenost u z-spremniku korigirati tako da postane jednaka udaljenosti novog slikovnog elementa. Konceptualno, možete zamisliti da je promatrač koji gleda scenu smješten na $-z$-osi (negdje daleko) i gleda prema $+z$-osi; sada je jasno da će vidjeti kao konačni slikovni element onaj koji je najbliži bližoj plohi. z-spremnik u ovoj vježbi još nećemo koristiti. Međutim, upoznat ćemo se s odsijecanjem koje radi OpenGL.

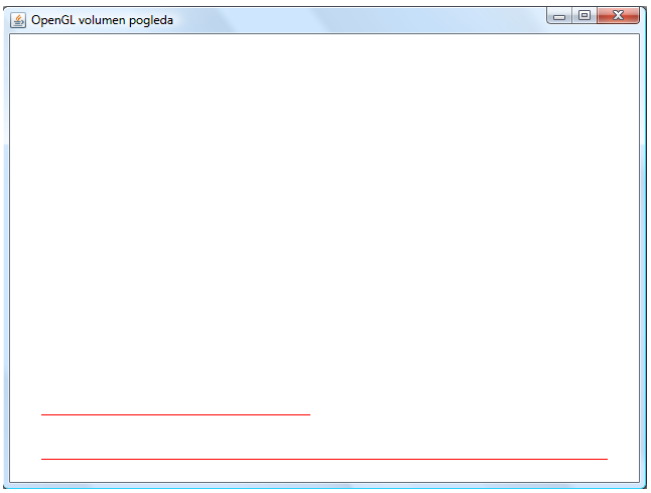
Jednom kada je slika stvorena na bližoj plohi, ona se prenosi na dio zaslona na kojem će biti prikazana. Za ovaj prijenos zaduženo je definiranje otvora (*viewport*): primjerice, ako imamo prozor dimenzija 640×480 i podesimo da je otvor upravo dio prozora koji se po x-u proteže od 0 do 640 a po y-u od 0 do 480, slika dobivena na bližoj plohi bit će razvučena po čitavoj površini prozora. Ako pak uz definirane dimenzije prozora postavimo otvor tako da se po x-u proteže od 320 do 640 a po y-u od 240 do 480, slika dobivena na bližoj plohi bit će prikazana u desnoj donjoj četvrtini prozora. Idemo ovo isprobati na konkretnom primjeru. Napravite osnovni program koji koristi OpenGL za prikaz scene i čije su funkcije reshape i display prikazane u nastavku (primjer je napisan u Javi uporabom biblioteke JOGL).

```
1         @Override
2         public void reshape(GLAutoDrawable glautodrawable, int x, int y,
3             int width, int height) {
4           GL2 gl2 = glautodrawable.getGL().getGL2();
5           gl2.glMatrixMode(GL2.GL_PROJECTION);
6           gl2.glLoadIdentity();
7
8           gl2.glViewport(0, 0, width, height);
9         }
10
11        @Override
12        public void display(GLAutoDrawable glautodrawable) {
13          GL2 gl2 = glautodrawable.getGL().getGL2();
14
15          gl2.glClearColor(1, 1, 1, 0);
16          gl2.glClear(GL.GL_COLOR_BUFFER_BIT);
17
18          gl2.glMatrixMode(GL2.GL_MODELVIEW);
19          gl2.glLoadIdentity();
20
21          // Crtam vodoravnu liniju na blizoj ravnini, pri dnu...
22          gl2.glColor3f(1, 0, 0);
23          gl2.glBegin(GL.GL_LINE_STRIP);
24          gl2.glVertex3f(-0.9f,-0.9f,-0.9f);
25          gl2.glVertex3f( 0.9f,-0.9f,-0.9f);
26          gl2.glEnd();
27
28          // Crtam vodoravnu liniju koja se udaljava, iznad prethodne...
29          gl2.glColor3f(1, 0, 0);
30          gl2.glBegin(GL.GL_LINE_STRIP);
31          gl2.glVertex3f(-0.9f,-0.7f,-0.9f);
32          gl2.glVertex3f( 0.9f,-0.7f,3.1f);
33          gl2.glEnd();
34        }
```

U funkciji reshape projekcijska matrica se inicijalizira na matricu identiteta; u funkciji display matrica modela i pogleda također se inicijalizira na matricu identiteta – posljedica je upravo prethodno opisano početno stanje kod kojeg je volumen pogleda ± 1 za sve tri koordinate. Zadajte kao veličinu prozora dimenzije 640×480 i pokrenite program. Rezultat bi trebao odgovarati onome prikazanom na slici 6.1.

Prikazani program crta dva vodoravna linijska segmenta. Prvi segment (retci 21-26) ima početnu točku $(-0.9, -0.9, -0.9)$ i proteže se do $(+0.9, -0.9, -0.9)$. x-koordinata se, dakle, mijenja gotovo od -1 (skroz lijevo) pa do gotovo $+1$ (skroz desno); uzete su vrijednosti koje su nešto manje tek tako da se ilustrira da je ono što će bez ikakvog podešavanja prikazati OpenGL upravo raspon od -1 do $+1$. Vrijednosti y koordinate je fiksna i iznosi -0.9 – kako je to gotovo -1, linija je pri dnu slike (samo dno bi imalo vrijednost $y = -1$ dok sam vrh slike ima vrijednost $y = +1$). Konačno, z-koordinata je također fiksna i iznosi -0.9 – to je vrlo blizu prednje plohe i nalazi se unutar volumena pogleda.

Drugi segment (retci 27-33) ima početnu točku $(-0.9, -0.7, -0.9)$ i proteže se do $(+0.9, -0.7, 3.1)$. y-koordinata je fiksna i iznosi -0.7 pa opet očekujemo vodoravni segment. x-koordinata se mijenja od -0.9 do $+0.9$ što znači gotovo od krajnje lijevog ruba pa gotovo do krajnje desnog ruba. Međutim, nacrtana linija je kraća. Da bismo ovo razumjeli, treba pogledati još i z-koordinate. Početna točka



Slika 6.1: Demonstracija odsijecanja koje se provodi uslijed definiranog volumena pogleda.

segmenta ima $z = -0.9$ no konačna točka segmenta ima $z = 3.1$: sve točke koje imaju z veći od 1 ispast će iz volumena pogleda i bit će odsiječene. S obzirom da je z-koordinata početne točke manja od z-koordinate završne točke, slijedi da kako crtamo segment, z-koordinata raste od -0.9 za $x = -0.9$ do 3.1 za $x = 0.9$. Vrijednost z postaje jednaka 1 (dolazi do ruba volumena pogleda) za $x = -0.05$ – što poprilici odgovara centru slike (x ide od -1 do $+1$, centar je tada na $x = 0$), i tu prestaje daljnje crtanje linijskog segmenta – OpenGL će ostatak odsijeći.

Napišite čitav program koji će Vam omogućiti da isprobate prethodni primjer. Pokušajte pokrenuti program uz različito definirane *viewport*-e; probajte

- sliku rastegnuti preko čitavog prozora,
- sliku smjestiti u gornju desnu četvrtinu prozora,
- sliku smjestiti u donju lijevu četvrtinu prozora,
- sliku smjestiti tako da zauzima četvrtinu prozora ali da je smještena u centar prozora.

Da biste se pripremili za ovu vježbu, pažljivo proučite poglavlja 5 i 6 u knjizi. Obratite pažnju na dvije konvencije djelovanja operatora: množenje točke matricom operatora te množenje matrice operatora točkom. U prvom slučaju točku ćemo prikazati kao jednoretčanu matricu; u drugom slučaju točku ćemo prikazati kao jednostupčanu matricu. Za matrice uz navedene konvencije vrijedi odnos prikazan izrazom 5.3 u knjzi. U ovoj vježbi koristit ćemo prvu konvenciju (množenje točke matricom operatora) dok OpenGL koristi drugu konvenciju (množi trenutnu matricu operatora točkom). Proučite kako se definiraju matrice operatora translacije, skaliranja i rotacije u 3D sustavu te koje nam naredbe za to nudi OpenGL. Također proučite kako se rade ortografska i perspektivna projekcija, te koje naredbe za to nudi OpenGL, i kako izgledaju pripadne matrice.

6.1 Pitanja

1. Što je to *volumen pogleda*?
2. Kako je određen inicijalni volumen pogleda kod *OpenGL*-a?
3. U kakvoj su vezi volumen pogleda i pojam odsijecanja? Što OpenGL odsijeca? Dajte primjer.
4. Napišite matricu operatora translacije za (d_x, d_y, d_z) uz obje konvencije. Kako glase matrice koje točku vraćaju u početnu?

5. Napišite matricu operatora skaliranja s faktorima s_x, s_y i s_z uz obje konvencije. Kako glase matrice koje točku vraćaju u početnu?

6. Je li transformacija pogleda jednoznačno definirana zadavanjem točke očišta i gledišta? Objasnite.

7. Što je to *view-up* vektor i čemu služi? Leži li on u ravnini projekcije?

8. OpenGL nudi naredbu gluLookAt. Čemu služi ta naredba, kakvu transformaciju pogleda radi, koji su njezini argumenti i kako izgleda pripadna matrica, uz obje konvencije (matricu ne trebate učiti napamet)?

9. OpenGL nudi naredbu glFrustum. Čemu služi ta naredba, kako izgleda volumen pogleda koji definira, koji su njezini argumenti i kako izgleda pripadna matrica, uz obje konvencije (matricu ne trebate učiti napamet)?

10. OpenGL nudi naredbu gluPerspective. Čemu služi ta naredba, kako izgleda volumen pogleda koji definira, koji su njezini argumenti i kako izgleda pripadna matrica, uz obje konvencije (matricu ne trebate učiti napamet)?

11. OpenGL nudi naredbu glViewport. Čemu služi ta naredba i koji su njezini argumenti?

12. Uporabom OpenGL naredbe glMatrixMode do sada ste odabirali dva moda: GL_PROJECTION i GL_MODEL. Što se podešava u jednom modu a što u drugom?

6.2 Zadatak

U okviru ove vježbe napravit ćete tri programa, pri čemu ćete koristiti dijelove koda koje ste već prethodno napisali. Sva tri programa omogućavaju izradu žičanog prikaza 3D tijela. Struktura programa je u sva tri slučaja ista:

1. Korisnik kao argument komandne linije zadaje naziv `.obj` datoteke u kojoj se nalazi pohranjeno tijelo.

2. Program učitava tijelo i radi normalizaciju (ovo ste implementirali u okviru prethodne vježbe, sada to samo iskoristite).

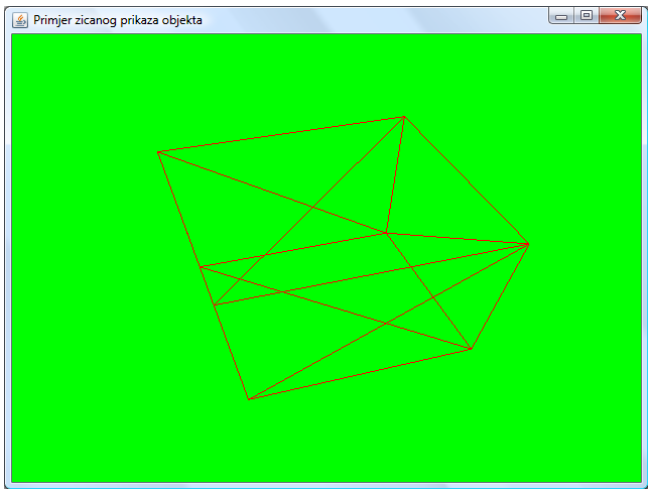
3. Program stvara prozor dimenzija 640×480 u kojem će uporabom OpenGL-a napraviti crtanje tijela.

4. Program treba podesiti matrice za transformaciju pogleda te za perspektivnu projekciju i potom nacrtati svaki od trokuta (koristite primitiv GL_LINE_LOOP jer želimo nacrtati samo rubove trokuta).

6.2.1 Zadatak 1

U ovoj inačici programa sav posao matričnog izračuna prepustit ćete OpenGL-u. Za definiranje transformacije pogleda koristite naredbu gluLookAt a za perspektivnu projekciju naredbu glFrustum. Vrhove trokuta OpenGL-u ćete predavati direktno naredbom glVertex3f.

Na slici 6.2 prikazan je rezultat koji biste morali dobiti prikazivanjem normalizirane kocke ako očište smjestite u točku $(3, 4, 1)$, gledište u točku $(0, 0, 0)$, uzmete *view-up* vektor $(0, 1, 0)$, te podesite volumen pogleda tako da se po x i y osima proteže ± 0.5 te da je bliža ploha na 1 a dalja na 100.



Slika 6.2: Žičani prikaz kocke.

6.2.2 Zadatak 2

U ovoj inačici programa sav posao matričnog izračuna prepustit ćete OpenGL-u. Za definiranje transformacije pogleda koristite naredbu gluLookAt a za perspektivnu projekciju naredbu gluPerspective. Vrhove trokuta predaju se OpenGL-u direktno naredbom glVertex3f. Izračunajte koje parametre trebate predati naredbi gluPerspective da biste prilikom prikaza normalizirane kocke dobili identičan prikaz kao kod prethodne inačice zadatka? Podesite Vaš program tako da inicijalno prikazuje upravo takav prikaz.

6.2.3 Zadatak 3

U ovoj inačici programa Vi ćete obaviti kompletan posao transformiranja i projiciranja. U metodi reshape kao matricu projekcije *morate* postaviti jediničnu matricu i kao *viewport* čitavu površinu prozora. Izradu metode display rješavat ćete u dva koraka.

Prvi korak

U jednoj od prethodnih vježbi napisali ste biblioteku za rad s vektorima i matricama. Dodajte toj biblioteci još i razred IRG (ako koristite objektno orijentirani programski jezik ili ako ne koristite objektno orijentirani programski jezik, ostvarite traženu funkcionalnost na način koji Vam to jezik koji ste odabrali omogućava). Stavite u taj razred statičke metode opisane u nastavku.

- IMatrix translate3D(**float** dx, **float** dy, **float** dz);
  Metoda vraća matricu koja odgovara operatoru translacije uz konvenciju množenja točke s matricom.

- IMatrix scale3D(**float** sx, **float** sy, **float** sz);
  Metoda vraća matricu koja odgovara operatoru skaliranja uz konvenciju množenja točke s matricom.

- IMatrix lookAtMatrix(IVector eye, IVector center, IVector viewUp);
  Metoda vraća matricu koja odgovara transformaciji pogleda koja je zadana očištem, pravcem očište-centar te *view-up* vektorom, uz konvenciju množenja točke s matricom. Ravnina u kojoj se stvara slika sadrži očište i njezina normala je kolinearna s vektorom očište-gledište. Ishodište koordinatnog sustava smješteno je u točku očišta, negativna z-os se proteže iz očišta prema zadanom centru. y-os određena je projekcijom *view-up* vektora na ravninu u kojoj se stvara slika. Konačno, sustav je desni čime je do kraja definiran. Odgovarajuću matricu možete pogledati direktno u dokumentaciji OpenGL-ove naredbe gluLookAt (link je dostupan na kraju upute).

Obratite pažnju na konvenciju uz koju je ta matrica zadana. Također, svi vektori koje zapišete u matricu trebaju biti normirani.

U metodi display podesite da je matrica modela također jedinična. Potom izračunajte matricu transformacije pogleda m (primjer u nastavku je uz iste parametre kao kada je to radio OpenGL).

```
IMatrix tp = IRG.lookAtMatrix(new Vector(3, 4, 1), new Vector(0, 0, 0), new Vector(0, 1, 0));
IMatrix m = tp;
```

Sada svaki vrh trokuta trasformirajte ovom matricom i rezultat predajte OpenGL-u kao 2D točku. Primjerice, ako je f jedan trokut (Face3D iz prethodne vježbe) a om tijelo koje se crta (ObjectModel iz prethodne vježbe), tada će sljedeći kod zadati i-ti vrh koji treba nacrtati.

```
// Dohvati zadani vrh trokuta:
Vertex3D vert = om.vertexes[f.indexes[i]];

// Zamotaj vrh kao vektor s homogenom koordinatom 1
IVector v = new Vector(
  vert.x,
  vert.y,
  vert.z,
  1.0);

// Transformiraj vrh:
IVector tv = v.toRowMatrix(false).nMultiply(m).toVector(false).nFromHomogeneus();

// Predaj x i y koordinatu OpenGL-u
gl2.glVertex2f((float)tv.get(0),(float)tv.get(1));
```

Uz pretpostavku da ste uzeli koordinate očišta, centra i *view-up* vektor kako je prethodno zadano, ova bi matrica vrhove normalizirane kocke trebala preslikati u sljedeće točke.

$$
\begin{aligned}
(-1.0, -1.0, -1.0) &\rightarrow (0.632, 0.372, -6.668)\\
(-1.0, -1.0, 1.0) &\rightarrow (-1.265, -0.124, -6.276)\\
(1.0, -1.0, -1.0) &\rightarrow (1.265, -1.116, -5.491)\\
(1.0, -1.0, 1.0) &\rightarrow (-0.632, -1.612, -5.099)\\
(1.0, 1.0, -1.0) &\rightarrow (1.265, 0.124, -3.922)\\
(1.0, 1.0, 1.0) &\rightarrow (-0.632, -0.372, -3.530)\\
(-1.0, 1.0, -1.0) &\rightarrow (0.632, 1.612, -5.099)\\
(-1.0, 1.0, 1.0) &\rightarrow (-1.265, 1.116, -4.707)
\end{aligned}
$$

Rezultat prikaza slike na zaslonu trebao bi izgledati kao što je prikazano na slici 6.3. Uočite da ovdje još nismo radili perspektivnu projekciju, već smo naprosto za svaku točku odbacili njezinu z-koordinatu (čime smo zapravo napravili ortografsku projekciju).

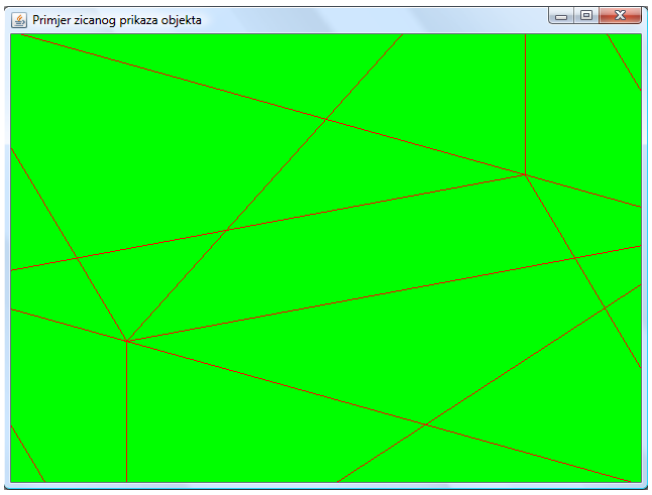
Drugi korak

Dodajte sada u razred IRGjoš jednu statičku metodu koja je opisana u nastavku.

- IMatrix IMatrix buildFrustumMatrix(**double** l, **double** r, **double** b, **double** t, **int** n, **int** f);
  Izvod za elemente pripadne matrice možete pogledati u knjizi izraz 6.5.5

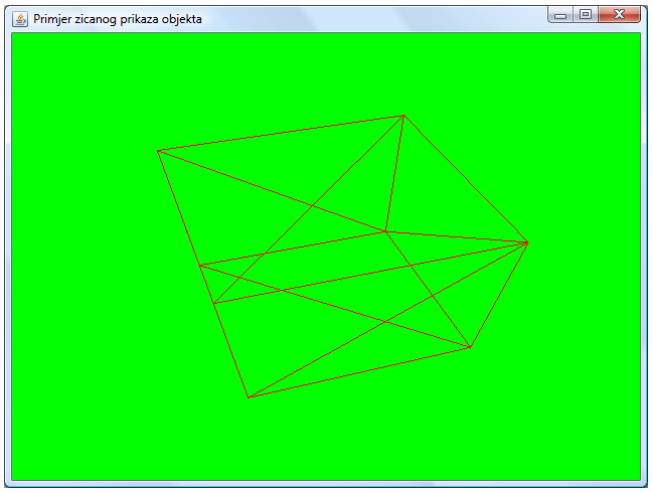
Sada možete modificirati stvaranje matrice m kako je opisano u nastavku.

```
IMatrix tp = IRG.lookAtMatrix(new Vector(3, 4, 1), new Vector(0, 0, 0), new Vector(0, 1, 0));
IMatrix pr = buildFrustumMatrix(-0.5, 0.5, -0.5, 0.5, 1, 100);
IMatrix m = tp.nMultiply(pr);
```



Slika 6.3: Žičani prikaz kocke uz ručno izračunatu matricu transformacije pogleda.

Uz ovako definiranu matricu m dobit ćete prikaz koji odgovara onom sa slike 6.4. Usporedite li taj prikaz i onaj sa slike 6.2, uočit ćete da je rezultat identičan.



Slika 6.4: Žičani prikaz kocke uz ručno izračunatu matricu transformacije pogleda i perspektivnu projekciju.

Uz pretpostavku da ste uzeli koordinate očišta, centra i *view-up* vektor kako je prethodno zadano, ova bi matrica (koja je sada umnožak matrice transformacije pogleda i perspektivne projekcije) vrhove normalizirane kocke trebala preslikati u sljedeće točke.

$$
\begin{aligned}
(-1.0, -1.0, -1.0) &\to (0.190, 0.112, 0.700) \\
(-1.0, -1.0, 1.0) &\to (-0.403, -0.040, 0.681) \\
(1.0, -1.0, -1.0) &\to (0.461, -0.407, 0.636) \\
(1.0, -1.0, 1.0) &\to (-0.248, -0.632, 0.608) \\
(1.0, 1.0, -1.0) &\to (0.645, 0.063, 0.490) \\
(1.0, 1.0, 1.0) &\to (-0.358, -0.211, 0.433) \\
(-1.0, 1.0, -1.0) &\to (0.248, 0.632, 0.608) \\
(-1.0, 1.0, 1.0) &\to (-0.537, 0.474, 0.575)
\end{aligned}
$$

Korak 3

Dodajte u kod tri varijable koje će se moći mijenjati na razini programa:

```
1 double angle = 18.4349488;
2 double increment = 1;
3 double r = 3.16227766;
```

Inicijalno, varijabla angle je postavljena na vrijednost $\arctan(\frac{1}{3})$ (u stupnjevima), varijabla r je postavljena na $\frac{1}{\sin(angle)}$ dok varijabla increment ima vrijednost 1 i kontrolira za koliko će se mijenjati varijabla angle. Evo što želimo: očište smo u svim primjerima smjestili u točku $(3, 4, 1)$ dok je *view-up* vektor uvijek pokazivao u smjeru y-osi globalnog sustava. Stoga ćemo modificirati program tako na pritisak tipki `r` i `l` očište pomiče po kružnici u $x - z$ ravnini čije središte leži na y-osi i čije sve točke pripadaju ravnini $y = 4$ (ovo je odabrano tako jer je izvorno očište bilo smješteno na tu koordinatu). No tada znamo da je točka $x = 3$ i $z = 1$ na obodu te kružnice, i znamo da za nju vrijedi $x^2 + z^2 = r^2$ te $x = r\cos(angle)$ i $z = r\sin(angle)$. Upravo iz ovih izraza izvučene su vrijednosti za početne vrijednosti r i angle; ovi izrazi također se dalje mogu koristiti za određivanje vrijednosti x i z ako je zadan novi kut (radijus kružnice se neće mijenjati).

Modificirajte program tako da mu dodate mogućnost osluškivanja pritisaka na tipke tipkovnice. Program treba podržavati sljedeće operacije.

Tipka	Tražena funkcionalnost
`r`	Kut angle potrebno je povećati za iznos inkrement, izračunati novu vrijednost očišta i zatražiti osvježavanje nacrtane scene.
`l`	Kut angle potrebno je umanjiti za iznos inkrement, izračunati novu vrijednost očišta i zatražiti osvježavanje nacrtane scene.
`ESC`	Kut angle potrebno je resetirati na inicijalnu vrijednost, izračunati novu vrijednost očišta i zatražiti osvježavanje nacrtane scene.

Ovu funkcionalnost (dinamičke promjene očišta) potrebno je dodati u sve tri inačice programa. Pokušajte stoga organizirati Vaš kod tako da ovo napravite na jednom mjestu a ne da iste promjene provodite tri puta. Objektno orijentirana paradigma ovdje Vam može ponuditi vrlo elegantnu tehniku kako to ostvariti.

Linkovi

- gluLookAt - `http://www.opengl.org/sdk/docs/man/xhtml/gluLookAt.xml`
- glFrustum - `http://www.opengl.org/sdk/docs/man/xhtml/glFrustum.xml`
- gluPerspective - `http://www.opengl.org/sdk/docs/man/xhtml/gluPerspective.xml`

Laboratorijska vježba 7

Uklanjanje skrivenih poligona

Kroz prethodne vježbe upoznali smo se s načinom modeliranja tijela zadavanjem njegovog oplošja navođenjem vrhova i poligona, te načinom prikaza takvog modela koji se temelji na transformaciji pogleda i projekciji. Jedan od najjednostavnijih načina prikaza ovakvih tijela jest prikaz žičanom formom što smo napravili u prethodnoj vježbi. Međutim, pogledamo li malo bolje taj prikaz, uočit ćemo da vidimo na ekranu nacrtane apsolutno sve poligone – one koji čine dio oplošja tijela koje je okrenuto prema nama ali i one koji čine dio oplošja tijela koje nije okrenuto prema nama već o obzirom na naš položaj predstavlja stražnji dio tijela. Zbog estetskih razloga, ali i zbog ubrzavanja crtanja scene htjeli bismo preskočiti crtanje poligona koje ne bismo trebali vidjeti. Jednom kada krenemo s bojanjem poligona, odbacivanje stražnjih poligona bit će presudno kako bi se ostvarile visoke performanse prilikom prikazivanja 3D scena. Stoga ćemo se u ovoj vježbi pozabaviti upravo problemom odbacivanja stražnjih poligona.

Svaki poligon koji se nalazi u prostoru možemo okarakterizirati kao prednji poligon ili kao stražnji poligon, s obzirom na poziciju promatrača. Pratimo li zraku svjetlosti od očišta pa do bilo kojeg poligona koje definira oplošje tijela, *prednji poligon* će biti poligon kod kojeg zraka iz okolnog prostora ulazi u unutrašnjost tijela; *stražnji poligoni* su poligoni kod kojih zraka iz unutrašnjosti tijela izlazi u okolni prostor. U scenama koje sadrže samo jedan konveksni objekt, prednji poligoni su ujedno i vidljivi poligoni – njih sve promatrač doista i vidi. Stražnji poligoni su pak uvijek nevidljivi; naime, ako je poligon stražnji, znači da, gledano od promatrača, zraka očište - taj poligon na tom mjestu izlazi iz tijela – onda je prije toga kroz neki drugi poligon morala ući u tijelo pa taj poligon sigurno skriva ovaj stražnji.

Ako tijelo nije konveksno, ili ako u sceni imamo više tijela koja su u djelomično ili u potpunosti smještena jedno iza drugoga gledano od promatrača, stražnje poligone i dalje sigurno možemo uvijek odbaciti – oni će sigurno biti nevidljivi. Međutim, sada više nije jednostavno niti odrediti koji je od prednjih poligona vidljiv. Zamislite samo dvije kocke koje su smještene jedna iza druge i pri tome je ona druga manja od prve tako da je prva potpuno skriva. Kada bismo htjeli nacrtati realističan prikaz ovakve scene, niti jedan prednji poligon druge kocke ne bismo smjeli nacrtati. Međutim, kod crtanja žičanog modela to nije baš jednostavno ostvariv zadatak. Stoga ćemo se u ovoj vježbi pozabaviti algoritmima koji sigurno rade za scenu koja se sastoji od jednog konveksnog tijela. Ako tijelo nije konveksno, uočit ćemo da se crtaju i neki poligoni koje ne bismo htjeli vidjeti; međutim, kod žičanog modela to ćemo ostaviti tako, a kad počnemo bojati poligone, ove probleme rješavat ćemo uporabom z-spremnika.

Algoritmi koje ćemo opisati u nastavku vrijede ako se pridržavamo do sada definiranih konvencija koje nam osiguravaju da su tijela zadana tako da im normale trokuta gledaju prema vanjštini tijela. Od toga prva dva algoritma rade u prostoru scene dok treći radi u prostoru projekcije (u 2D prostoru).

Nakon učitavanja tijela pretpostavit ćemo da su za sve trokute izračunate normale i jednadžbe pripadnih ravnina u skladu s izrazima 5.2 i 5.3 iz ovih uputa. Također, pretpostavit ćemo da je s *eye* označen položaj očišta iz kojeg promatrač gleda scenu. Normale svih poligona moraju gledati prema vanjštini tijela.

Algoritam 1. Poligon je prednji ako se očište nalazi iznad ravnine u kojoj leži poligon. Poligon je stražnji ako se očište nalazi ispod ravnine u kojoj leži poligon. Neka je jednadžba ravnine u kojoj leži poligon jednaka $ax+by+cz+d=0$. Slijedi da je poligon prednji ako je $a\cdot eye_x+b\cdot eye_y+c\cdot eye_z+d>0$, odnosno da je poligon stražnji ako je $a\cdot eye_x+b\cdot eye_y+c\cdot eye_z+d<0$.

Algoritam 2. Neka je s $\vec{c}$ označen centar poligona (tj. trokuta): $\vec{c}=\frac{\vec{V}_i+\vec{V}_{i+1}+\vec{V}_{i+2}}{3}$. Neka je s $\vec{e}$ označen vektor iz centra poligona prema promatraču: $\vec{e}=eye-\vec{c}$. Konačno, neka je $\vec{n}$ normala na ravninu u kojoj leži poligon. Poligon je prednji ako je kut između vektora $\vec{n}$ i $\vec{e}$ manji od 90°. Poligon je stražnji ako je kut između vektora $\vec{n}$ i $\vec{e}$ veći od 90°. Prisjetite se kako se računa kosinus kuta između dvaju vektora: on je jednak skalarnom produktu podijeljenom s umnoškom normi vektora. No, kako su norme uvijek nenegativne, a kosinus kuta na 90° mijenja predznak, dovoljno je ispitati samo predznak skalarnog produkta što je računski efikasnije. Slijedi da je poligon prednji ako je $\vec{n}\cdot\vec{c}>0$, odnosno da je poligon stražnji ako je $\vec{n}\cdot\vec{c}<0$.

Algoritam 3. Neka su točke trokuta $\vec{V}_i$, $\vec{V}_{i+1}$ te $\vec{V}_{i+2}$ iz 3D sustava scene preslikane u 2D sustav projekcije kao $\vec{V}_i^*$, $\vec{V}_{i+1}^*$ te $\vec{V}_{i+2}^*$. Poligon je prednji ako je smjer obilaska točaka $\vec{V}_i^*$, $\vec{V}_{i+1}^*$ te $\vec{V}_{i+2}^*$ suprotan smjeru obilaska kazaljki na satu. Poligon je stražnji ako je smjer obilaska točaka $\vec{V}_i^*$, $\vec{V}_{i+1}^*$ te $\vec{V}_{i+2}^*$ jednak smjeru obilaska kazaljki na satu. Prisjetite se zašto to vrijedi!

Algoritmi 1 i 2 rade u sustavu scene i omogućavaju detekciju stražnjih poligona prije no što se napravi projiciranje koje je računski zahtjevno. Međutim, mana algoritama je da zahtjevaju pamćenje položaja očišta. Algoritam 3 radi direktno u sustavu projekcije i primjenjuje se nakon što su vrhovi projicirani u 2D prostor; međutim, taj algoritam ne treba nikakve dodatne informacije.

7.1 Pitanja

1. Prisjetite se svih pitanja iz vježbi 5 i 6.
2. Kako se definiraju prednji i stražnji poligoni?
3. Jesu li prednji poligoni uvijek vidljivi? O čemu to ovisi? Objasnite.
4. Jesu li stražnji poligoni uvijek skriveni? O čemu to ovisi? Objasnite.
5. Jesu li "prednji poligon" i "vidljivi poligon" te "stražnji poligon" i "nevidljivi poligon" sinonimi? Objasnite.
6. Objasnite kako radi algoritam 1.
7. Što bi trebalo promijeniti u algoritmu 1 ako bi vrijedile sve prethodno ustanovljene konvencije osim što bi poligoni bili zadani tako da pripadne normale ravnina gledaju u unutrašnjost tijela?
8. Objasnite kako radi algoritam 2.
9. Što bi trebalo promijeniti u algoritmu 2 ako bi vrijedile sve prethodno ustanovljene konvencije osim što bi poligoni bili zadani tako da pripadne normale ravnina gledaju u unutrašnjost tijela?
10. Objasnite kako radi algoritam 3.
11. Što bi trebalo promijeniti u algoritmu 3 ako bi vrijedile sve prethodno ustanovljene konvencije osim što bi poligoni bili zadani tako da pripadne normale ravnina gledaju u unutrašnjost tijela?
12. Kako se računa skalarni produkt dvaju vektora?
13. Kako se računa vektorski produkt dvaju vektora?

14. Kako se računa kosinus kuta između dvaju vektora?

15. Kako se računa jednadžba ravnine u kojoj se nalazi zadani trokut?

7.2 Zadatak

U okviru ove vježbe modificirat ćete inačice programa koje ste napravili u vježbi 6 tako što ćete omogućiti da se žičani model tijela nacrta uz odbačene stražnje poligone. Ako još niste, dodajte u strukturu podataka kojom pamtite jedan poligon koeficijente pripadne ravnine. Te koeficijente izračunajte nakon što obavite normalizaciju tijela. Proširite tu strukturu sa zastavicom visible.

7.2.1 Zadatak 1

Uzmite program koji je crtao žičani model tijela uporabom OpenGL naredbi gluLookAt i glFrustum. Kod njega ste trokute crtali uporabom primitiva GL_LINE_LOOP. Poligone nacrtane tim primitivom OpenGL nažalost ne smatra pravim poligonima u smislu da njegovi vrhovi definiraju dio površine koji ima prednju i stražnju stranu već naprosto kao niz linijskih segmenata. Promijenite stoga primitiv kojim crtate iz GL_LINE_LOOP u GL_POLYGON (ostatak se ne mijenja – trebate zadati identične vrhove kao i kod primitiva GL_LINE_LOOP). Primitiv GL_POLYGON OpenGL-u govori da se radi o podskupu ravnine, koji shodno tome ima definiranu prednju i stražnju stranu. Provjeru radi li se o prednjoj ili stražnjoj strani OpenGL radi u sustavu porjekcije prema algoritmu 3. Međutim, omogućeno je podešavanje koji obilazak vrhova poligona definira prednju stranu. Za to se koristi naredba glFrontFace čiji argument može biti konstanta GL_CCW (kratica od engl. *Counter-Clock-Wise* ako je prednji poligon definiran smjerom obilaska koji je suprotan od smjera gibanja kazaljki na satu) ili pak konstanta GL_CW (kratica od engl. *Clock-Wise* ako je prednji poligon definiran smjerom obilaska koji je jednak smjeru gibanja kazaljki na satu). Ako drugačije nije definirano, koristi se početna vrijednost GL_CCW što je baš u skladu s dosad korištenom konvencijom. Za svaki poligon OpenGL prilikom njegovog prikazivanja na zaslonu može nacrtati tri stvari:

- samo točke na mjestima na kojima se nalaze vrhovi – ovo će se dogoditi ako se kao način prikaza poligona odabere GL_POINT;
- linijama može spojiti vrhove – ovo će se dogoditi ako se kao način prikaza poligona odabere GL_LINE i to je način kako smo do sada i prikazivali poligone;
- čitavu površinu poligonu može ispuniti trenutnom bojom – ovo će se dogoditi ako se kao način prikaza poligona odabere GL_FILL.

Za svaki poligon, ovisno o tome radi li se o prednjem ili stražnjem poligonu OpenGL omogućava korisniku da podesi način na koji će se crtati takav poligon; moguće je podesiti kako će se prikazivati prednji poligoni a kako stražnji poligoni. Podešavanje načina radi se naredbom:

void glPolygonMode(GLenum face, GLenum mode);

pri čemu se kao prvi argument predaje GL_FRONT, GL_BACK ili GL_FRONT_AND_BACK dok je drugi argument jedan od opisana tri načina prikaza. Nakon ulaska u metodu display najprije podesite da se za prednje poligone crta samo rub poligona (da se vrhovi spajali linijom). Potom recite OpenGL-u da uključi odbacivanje poligona (koristite naredbu glEnable uz argument GL_CULL_FACE). I konačno, zadajte da se odbacuju stražnji poligoni (koristite naredbu glCullFace; argument može biti GL_FRONT, GL_BACK te GL_FRONT_AND_BACK, ovisno o tome što želite da se odbacuje). Linkovi na pripadnu dokumentaciju dani su na kraju upute za ovu vježbu. Tek nakon što ste ovo napravili krenite u prikazivanje trokuta. Vi ćete ih poslati sve dok će OpenGL odabrati samo prednje i njih će prikazati.

7.2.2 Zadatak 2

Uzmite program koji je crtao žičani model tijela pri čemu ste Vi sami radili sve transformacije i projekcije. Dodajte u razred ObjectModel dvije metode:

- **void** determineFaceVisibilities1(IVector eye) koja uporabom algoritma 1 za svaki trokut postavlja vrijednost zastavice visible na **true** ako je torkut prednji odnosno na **false** ako je trokut stražnji, te
- **void** determineFaceVisibilities2(IVector eye) koja uporabom algoritma 2 za svaki trokut postavlja vrijednost zastavice visible na **true** ako je torkut prednji odnosno na **false** ako je trokut stražnji.

Također, napravite u razredu IRG pomoćnu metodu isAntiClockwise koja prima trokut (vrhovi imaju 2D koordinate) i koja vraća **true** ako je trokut takav da su mu vrhovi zadani u smjeru suprotnom od smjera kazaljke na satu, a inače vraća **false**.

Modificirajte program tako da osluškuje pritiske na tipke 1, 2, 3 te 4. Pritiskom na odgovarajuću tipku program bira jedan od četiri načina odbacivanja stražnjih poligona, kako je opisano u nastavku te na zaslon ispisuje poruku koji je način odabran.

1. *Bez odbacivanja* – program crta sve trokute, neovisno o tome jesu li prednji ili stražnji.
2. *Odbacivanja algoritmom 1* – prije no što se krene u crtanje trokuta poziva se metoda determineFaceVisibilities1. Potom se crtaju samo trokuti koji imaju postavljeno visible==**true**.
3. *Odbacivanja algoritmom 2* – prije no što se krene u crtanje trokuta poziva se metoda determineFaceVisibilities2. Potom se crtaju samo trokuti koji imaju postavljeno visible==**true**.
4. *Odbacivanja algoritmom 3* – za svaki se trokut vrhovi projiciraju u 2D; potom se predaju metodi isAntiClockwise i trokut se šalje na crtanje samo je ta metoda vratila vrijednost **true**.

Provjerite rad algoritma na primjeru kocke; pokušajte malo rotirati očište i promatrajte rade li algoritmi dobro za sve kuteve. Trebali biste dobiti prikaz kakav je dan na slici 7.1. Potom učitajte medvjedića. Je li sve u redu?

Slika 7.1: Žičani prikaz kocke uz odbačene skrivene poligone.

Linkovi

- glBegin - `http://www.opengl.org/sdk/docs/man/xhtml/glBegin.xml`

- glPolygonMode - `http://www.opengl.org/sdk/docs/man/xhtml/glPolygonMode.xml`
- glFrontFace - `http://www.opengl.org/sdk/docs/man/xhtml/glFrontFace.xml`
- glEnable - `http://www.opengl.org/sdk/docs/man/xhtml/glEnable.xml`
- glCullFace - `http://www.opengl.org/sdk/docs/man/xhtml/glCullFace.xml`

Laboratorijska vježba 8

Bezierova krivulja

Rad s krivuljama vrlo je važno područje računalne grafike. Možda najbanalniji primjer kojim možemo ilustrirati raširenost krivulja jest primjer *TrueType* fontova koji se danas koriste svugdje, i koji se temelje na opisu rubova slova krivuljama (najčešće linijama te kvadratnim ili kubnim Bezierovim krivuljama). U okviru ove vježbe isprobat ćemo stoga postupak crtanja Bezierove krivulje.

Proučite u knjizi poglavlje 7 a posebice poglavlje 7.3. Obratite pažnju na način izračuna interpolacije Bezierove krivulje te načine konstrukcije. Pogledajte kako se temeljem zadanih točaka kroz koje mora proći krivulja može doći do aproksimacijske Bezierove krivulje (posebice izraz 7.1 u knjizi, i njegov izvod).

8.1 Pitanja

1. Čime je određena Bezierova krivulja?
2. Što je to *red* Bezierove krivulje i o čemu on ovisi?
3. Koje dvije vrste težinskih funkcija postoje koje se mogu koristiti za izračun točaka Bezierove krivulje?
4. Vezano uz prethodno pitanje, koja je razlika ako se koriste jedne odnosno druge težinske funkcije (što se množi u prvom a što u drugom slučaju)? Napišite izraz kojim je određena proizvoljna točka $\vec{p}(t)$ aproksimacijske Bezierove krivulje.
5. Koja je razlika između aproksimacijskih krivulja i interpolacijskih krivulja?
6. Bezierova krivulja spada u porodicu parametarskih krivulja – što to znači?

8.2 Zadatak

U okviru ove vježbe Vaš je zadatak napraviti program koji će korisniku omogućiti da klikanjem desnog gumba miša zadaje točke kontrolnog poligona u 2D prostoru. Nakon svake novododane točke, program treba osvježiti trenutni prikaz tako što će nanovo nacrtati:

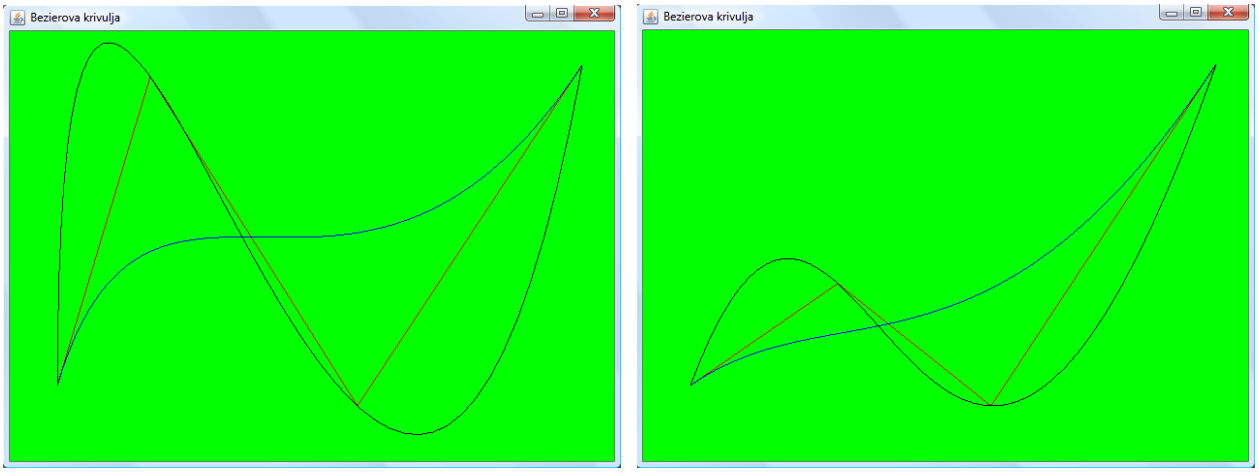
- crvenom bojom kontrolni poligon koji je definiran do tada zadanim točkama,
- aproksimacijsku Bezierovu krivulju koja je definirana do tada zadanim točkama te
- interpolacijsku Bezierovu krivulju koja je definirana do tada zadanim točkama.

Za rad s matricama koristite biblioteku koju ste razvili na početku.

Program treba korisniku omogućiti da lijevim klikom miša dodaje točke kontrolnog poligona (svakim klikom nova se točka nadoda na kraj). Također, program treba omogućiti korisniku da desnim gumbom miša može "uhvatiti" proizvoljni do tada zadan vrh kontrolnog poligona te da ga povlači po prostoru tako dugo dok korisnik drži desni gumb pritisnut i pomiče miš. Naravno, čim se vrh pomakne, potrebno je nanovo nacrtati dobivenu sliku. Ako korisnik pritisne na tipkovnici tipku `ESC`, brišu se sve zapamćene točke kontrolnog poligona i program se ponaša kao da je upravo pokrenut. Da biste ovo realizirali, prisjetite se kako se mogu primati dojave vezane uz miš i tipkovnicu.

Primjer izvođenja programa prikazan je na slici 8.1. Desni dio te slike prikazuje sliku nastalu odvlačenjem drugog vrha kontrolnog poligona prema dolje.

Vaš kod organizirajte tako da dodate dvije funkcije za crtanje koje obje primaju vrhove poligona; prva treba nacrtati poligon a druga aproksimaciju Bezierovu krivulju. U metodi display tada pozovite prvu funkciju kako biste dobili poligon, pozovite drugu funkciju kako biste dobili aproksimacijsku Bezierovu krivulju, izračunajte nove vrhove aproksimacijske krivulje koja interpolira orginalne vrhove i opet pozovite drugu funkciju s tim novim vrhovima.



(a) Izvorna slika.

(b) Slika na kojoj je naknadno pomaknut drugi vrh kontrolnog poligona.

Slika 8.1: Primjer nacrtanih Bezierovih krivulja. Prikazan je kontrolni poligon (crveno), aproksimacijska (plavo) te interpolacijska (crno) Bezierova krivulja.

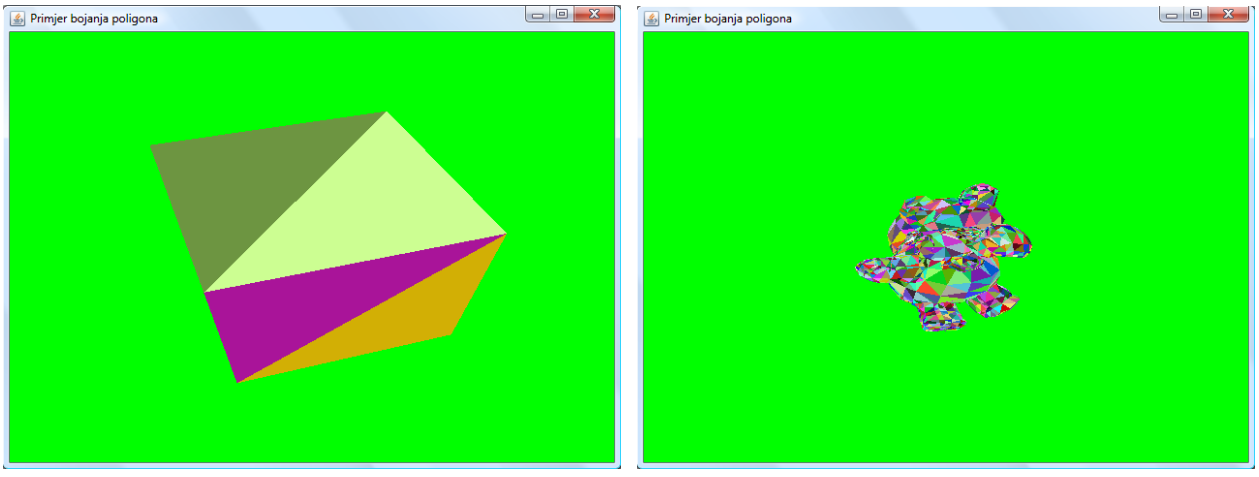
Aproksimacijsku Bezierovu krivulju crtajte na način da izračunate div točaka kroz koje prolazi krivulja (interval parametra t koji je $[0, 1]$ uniformno uzorkujte u div točaka (primjerice, za div=4 računali biste točke za $t = \frac{0}{3}$, $t = \frac{1}{3}$, $t = \frac{2}{3}$ i $t = \frac{3}{3}$). Točke koje ste dobili nacrtajte primitivom GL_LINE_STRIP. Za vrijednost div odaberite neki umjereni broj (primjerice između 20 i 100).

Laboratorijska vježba 9

Sjenčanje

Kako bi se dobio realističan prikaz 3D scene, umjesto žičanog modela objekata potrebno je napraviti vizualizaciju kompletnog tijela. S obzirom da koristimo pristup kod kojeg oplošje tijela modeliramo poligonima (štoviše, trokutima), za svaki poligon znamo gdje se nalazi u sceni te ga možemo obojati nekom bojom. Da bismo to mogli, a uz pretpostavku da trokute i dalje crtamo primitivom GL_POLYGON, potrebno je OpenGL-u narediti da umjesto ruba poligona napravi popunjavanje poligona trenutnom bojom. Prisjetite se vježbe 7 – to ćemo napraviti tako da naredbom glPolygonMode podesimo da se prednji poligoni popunjavaju (koristiti GL_FILL). Potom se prije zadavanja prvog vrha svakog poligona naredbom glColor podesi boja kojom želimo popuniti poligon, i potom se zadaju nizom naredbi glVertex vrhovi poligona; OpenGL će predani poligon uniformno popuniti zadanom bojom.

Primjer ovako prikazane scene dan je na slici 9.1. Pri tome slika 9.1a prikazuje vizualizaciju konveksnog tijela (kocke), i taj prikaz izgleda bitno bolje u odnosu na žičani prikaz, iako je još uvijek daleko od onoga što bismo željeli dobiti. Na slici 9.1b prikazana je vizualizacija konkavnog tijela (medvjedić) i tu možemo uočiti da rezultat nije zadovoljavajući. Problema su dva: uslijed velikog broja premalih poligona, čitav je prikaz neprihvatljivo šaren; drugi je problem taj što vidimo poligone koje ne bismo smjeli vidjeti. Pogledajte sliku 9.1b malo pažljivije – medvjedić je okrenut prema nama no ipak vidimo kroz njegovu glavu dio leđa i repa (vide se poligoni koji jesu prednji pa ih algoritam odbacivanja nije odbacio, no oni bi trebali biti zaklonjeni poligonima koji su bliži pa se ne bi smjeli vidjeti; kako se događa da su u datoteci modela oni navedeni kasnije, njih crtamo zadnje i zato ih ipak vidimo).

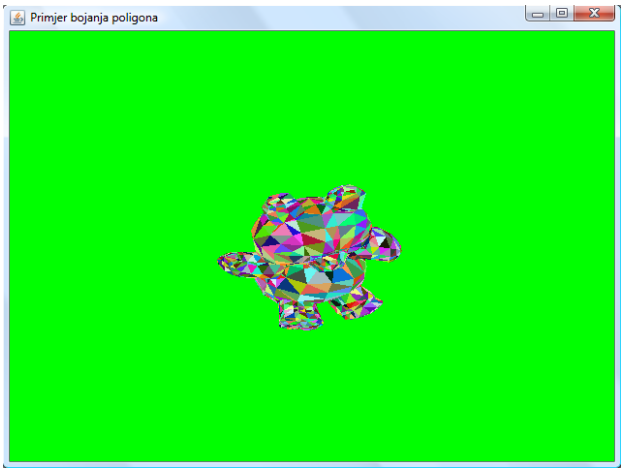


(a) Kocka. (b) Medvjedić.

Slika 9.1: Primjer vizualizacije 3D scene kod kojeg su stražnji poligoni odbačeni a prednji uniformno popunjeni slučajno odabranom bojom.

Ovaj drugi problem rješit ćemo relativno jednostavno – naložit ćemo OpenGL-u da prilikom generiranja prikaza koristi z-spremnik: za svaku točku koju nacrta, mora zapamtiti i njezinu udaljenost od ravnine prikaza. Potom, ako neku drugu točku treba nacrtati na istoj poziciji u ravnini prikaza, to će

biti dozvoljeno samo ako je njezina udaljenost u 3D prostoru od ravnine prikaza manja od udaljenosti prethodno nacrtane točke; na taj način dalji trokuti neće uspjeti pokvariti prikaz kao što je to bio slučaj sa slikom 9.1b. Da bismo koristili z-spremnik, najprije je potrebno uključiti njegovu uporabu – to se radi naredbom glEnable uz predavanja konstante GL_DEPTH_TEST. Potom se, prilikom brisanja pozadine (naredba glClear) treba predati i zahtjev za brisanjem sadržaja z-spremnika (koristeći binarni *ili*-operator dodajte konstantu GL_DEPTH_BUFFER_BIT), nakon čega dalje crtate na uobičajeni način. Ako za stvaranje prozora koristite biblioteku GLUT, prilikom poziva funkcije glutInitDisplayMode trebate predati i zastavicu GLUT_DEPTH tako da zatražite stvaranje prozora koji će podržavati uporabu z-spremnika; bez toga nećete moći omogućiti z-spremnik jer ga prozor neće imati. Rezultat koji se dobije sada će biti bolji – pogledajte sliku 9.2; više se kroz glavu ne vide dijelovi leđa kao niti rep. Boje su nešto drugačije jer se svaki puta generiraju slučajno.



Slika 9.2: Vizualizacija konkavnog tijela uz odbacivanje stražnjih poligona te uporabu z-spremnika.

Ono što sada želimo postići jest vjerniji prikaz modela; prikaz koji sliči onome što bismo vidjeli u prirodi, gdje nam ništa nije vidljivo ako ne postoji izvor svjetlosti koji će obasjati model i tako ga učiniti vidljivim. Što ćemo točno vidjeti ovisi o dva faktora:

- svojstvima izvora svjetlosti te
- svojstvima materijala.

Izvor svjetlosti obično opisujemo RGB modelom pa stoga navodimo intenzitet kojim izvor zrači crvenu komponentu, intenzitet kojim izvor zrači zelenu komponentu te intenzitet kojim izvor zrači plavu komponentu. Izostanak bilo kakvog zračenja označit ćemo intenzitetom vrijednosti 0 dok će maksimalnom intenzitetu odgovarati vrijednost 1. Izvor koji zrači sve tri komponente intenzitetom 1 bit će izvor bijele svjetlosti; izvor koji zrači crvenu komponentu s vrijednošću 1 a zelenu i plavu s vrijednošću 0 bit će izvor crvene svjetlosti i tako dalje.

Što se događa kada svjetlost padne na neku površinu fizikalno je vrlo složeno te se u praksi opisuje pojednostavljenim modelima. Prije no što nastavite dalje, obavezno u knjizi proučite poglavlje 9. Da bismo vidjeli neki objekt, taj objekt mora emitirati svjetlost koja će doći do našeg oka. Da bi se to dogodilo (ako tijelo nije izvor), nužno je da u sceni postoji izvor svjetlosti koji može direktno ili indirektno osvijetliti objekt koji promatramo. Koliko će svjetlosti doći do našeg oka, ovisi o vrsti i količini svjetlosti koja osvjetljava objekt te o svojstvima materijala od kojeg je objekt načinjen. Prema Phongovom modelu, ukupna količina svjetlosti koja do nas dolazi od nekog djelića površine računa se kao suma ambijentne, difuzne i zrcalne komponente. Svojstva materijala određuju u kojoj mjeri materijali apsorbira a u kojoj mjeri zrači pojedine komponente svjetlosti. Primjerice, materijal koji, kada ga obasjamo bijelom svjetlošću djeluje crveno, apsorbira sve komponente svjetlosti osim crvene koju reflektira. Ako bismo takav materijal obasjali zelenom svjetlošću, materijal bi djelovao crno – zelenu svjetlost bi apsorbirao i ništa ne bi reflektirao čime do nas ne bi došla nikakva svjetlost od tog materijala.

U ovoj vježbi koristit ćemo jedan izvor svjetlosti. Njega ćemo modelirati s četiri porodice parametra (slično radi i OpenGL, samo što on nudi još više podesivih parametara) kako je opisano u nastavku.

1. Pozicija u sceni na kojoj se nalazi izvor (izvor ćemo promatrati kao točkasti izvor smješten u jednu točku prostora koji zrači svjetlost zadanih intenziteta).
2. Intenziteti ambijentnih komponenti izvora: crvena, zelena i plava: I_{ar}, I_{ag}, I_{ab}.
3. Intenziteti difuznih komponenti izvora: crvena, zelena i plava: I_{dr}, I_{dg}, I_{db}.
4. Intenziteti zrcalnih komponenti izvora: crvena, zelena i plava: I_{rr}, I_{rg}, I_{rb}.

Materijal od kojeg je sastavljeno tijelo (odnosno površine prednjih poligona) modelirat ćemo s tri porodice parametara – zadat ćemo koeficijente koji govore koliki postotak određene komponente svjetlosti materijal ponovno zrači dalje.

1. Koeficijenti koji definiraju postotak zračenja ambijentnih komponenti izvora za crvenu, zelenu i plavu: k_{ar}, k_{ag}, k_{ab}.
2. Koeficijenti koji definiraju postotak zračenja difuznih komponenti izvora za crvenu, zelenu i plavu: k_{dr}, k_{dg}, k_{db}.
3. Koeficijenti koji definiraju postotak zračenja zrcalnih komponenti izvora za crvenu, zelenu i plavu: k_{rr}, k_{rg}, k_{rb} te koeficijent koji utvrđuje količinu sjaja k_{rn} (u izrazu za zrcalnu komponentu to je potencija na koju se diže kosinusni član).

Prilikom izračuna ukupnog intenziteta kojim će zračiti određeni element površine koristite izraz koji je poopćenje izraza 9.8 navedenog u knjizi. Naime, izraz 9.8 pretpostavlja da je izvor definiran s dvije komponente: jednim intenzitetom koji opisuje ambijentno zračenje te drugim intenzitetom koji se koristi za izračun difuzne i zrcalne komponente. S obzirom da smo prethodno definirali da izvor ima tri komponente intenziteta, koristit ćemo izraz 9.8 iz knjige ali na način da ga primijenimo za svaku komponentu zasebno (posebno za crvenu, posebno za zelenu, posebno za plavu) te tako da za izračun ambijentne komponente koristimo ambijentni intenzitet izvora i ambijentni koeficijent materijala, za difuznu komponentu difuzni intenzitet izvora i difuzni koeficijent materijala te za zrcalnu komponentu zrcalni intenzitet izvora te koeficijent sjaja materijala.

Konačna difuzna komponenta ovisi još i o kutu između normale na površinu i vektora od površine prema izvoru. Ako je ovaj kut veći od 90°, kosinus u izrazu 9.3 u knjizi će biti negativan i u tom slučaju se ta komponenta postaviti na nulu (nema smisla svjetlost umanjivati). U slučaju zrcalne komponente, konačni iznos komponente ovisit će o kutu između reflektirane zrake koja ide od izvora do površine i zrake koja ide od površine do oka promatrača. Pri tome se kao reflektirani vektor uzima vektor koji je usmjeren od površine do izvora. Ako je kut između reflektiranog vektora i vektora prema promatraču veći od 90°, komponentu treba postaviti na nulu.

Možemo koristiti tri načina izračuna intenziteta za svaki slikovni element projiciranog poligona; navodimo ih prema računskoj složenosti, od najjednonstavnijih do najsloženijih.

Konstantno sjenčanje. Za svaki se poligon izračuna njegovo središte (aritmetička sredina svih vrhova). U tom središtu se izračuna intenzitet i čitav se poligon oboja tom bojom. Kao normala se koristi normala ravnine u kojoj leži poligon a kao vektor prema izvoru se računa vektor iz središta poligona prema izvoru.

Gouraudovo sjenčanje. Za svaki se vrh poligona izračuna pripadni intenzitet. Da bi se dobio vizualni kontinuitet, kao normala se ne koristi normala ravnine u kojoj leži poligon, već se u svakom vrhu izračuna aritmetička sredina normala svih poligona koji dijele taj vrh. Pri tome normale poligona prije ulaska u aritmetičku sredinu mogu biti normirane a rezultat aritmetičke sredine treba normirati; tako dobiveni vektor se koristi kao normala u vrhu. Za svaki se vrh izračuna

efektivna normala; za zrcalnu se komponentu koristi vektor iz vrha pa do promatrača. Nakon što se za svaki vrh izračuna pripadni intenzitet, napravi se interpolacija za sve točke površine poligona.

Phongovo sjenčanje. Za svaki se vrh izračuna efektivna normala kako je opisano kod Gouraudovog sjenčanja. Potom se za svaku točku poligona računa interpolirana normala. Potom se tako dobivena normala koristi za izračun intenziteta u točki.

Sumirajmo ovo još jednom: konstantno sjenčanje računa jedan intenzitet i tom bojom oboja čitav poligon; Gouraudovo sjenčanje računa onoliko intenziteta koliko poligon ima vrhova i potom radi interpolaciju za sve točke površine poligona; konačno, Phongovo sjenčanje računa intenzitet za svaku točku površine poligona temeljem interpoliranih normala koje se računaju za sve vrhove poligona.

9.0.1 Osvjetljavanje u OpenGL-u

OpenGL nam nudi mogućnost provođenja opisanih izračuna automatski. Prvi korak jest uključiti uporabu modela osvjetljavanja što se radi naredbom glEnable uz konstantu GL_LIGHTNING. Uključivanjem modela osvjetljavanja OpenGL će dalje ignorirati naredbu glColor jer će boje sam računati. Uključivanjem modela osvjetljavanja, OpenGL inicijalno definira globalni ambijentni intenzitet iznosa $rgb = (0.2, 0.2, 0.2)$ tako da osigura vidljivost tijela čak ako se u scenu ne doda niti jedan izvor svjetlosti. S obzirom da ćemo mi dodati izvor svjetlosti, globalne ambijentne intenzitete postavit ćemo na nulu što se radi naredbom glLightModel čiji će prvi parametar biti konstanta GL_LIGHT_MODEL_AMBIENT a drugi parametar boja (uz alpha vrijednost) – definirat ćemo neprozirnu crnu, pa su vrijednosti koje se predaju kao polje float-ova redom 0 (crvena), 0 (zelena), 0 (plava) te 1 (zadana boja je skroz neprozirna). U Javi (u JOGL-u), poziv izgleda ovako:

```
1 gl2.glLightModelfv(GL2.GL_LIGHT_MODEL_AMBIENT, new float[] {0.0f,0.0f,0.0f,1f}, 0);
```

OpenGL podržava rad s više izvora svjetlosti (8 minimalno) koji se biraju konstantama GL_LIGHT0 do GL_LIGHT7. Za svaki izvor koji se želi koristiti treba podesiti poziciju te pripadne intenzitete. Evo primjera za izvor 0, opet u Javi.

```
1 IVector lightVector = new Vector(4,5,3);
2 gl2.glLightfv(GL2.GL_LIGHT0, GL2.GL_POSITION, new float[] {4f,5f,3f,1f}, 0);
3 gl2.glLightfv(GL2.GL_LIGHT0, GL2.GL_AMBIENT, new float[] {0.2f,0.2f,0.2f,1f}, 0);
4 gl2.glLightfv(GL2.GL_LIGHT0, GL2.GL_DIFFUSE, new float[] {0.8f,0.8f,0f,1f}, 0);
5 gl2.glLightfv(GL2.GL_LIGHT0, GL2.GL_SPECULAR, new float[] {0f,0f,0f,1f}, 0);
```

Redak 1 zadaje poziciju izvora. Predaju se 4 vrijednosti tipa **float** jer OpenGL očekuje poziciju u homogenom prostoru (stoga je zadnja komponenta postavljena na 1 tako da prve tri komponente odgovaraju poziciju u 3D prostoru). Retci 2, 3 i 4 određuju ambijentne, difuzne i zrcalne intenzitete svjetlosti izvora, opet u zapisu rgba (zadnja komponenta određuje neprozirnost boje; 1 znači da je boja skroz neprozirna).

Nakon što su definirana svojstva izvora, izvor je potrebno omogućiti što se radi naredbom glEnable uz odgovarajuću konstantu koja označava izvor. Primjerice, da bismo omogućili uporabu izvora 0 koji smo prethodno konfigurirali, potrebno je pozvati:

```
1 gl2.glEnable(GL2.GL_LIGHT0);
```

Sljedeći korak se može raditi tik prije crtanja tijela, ako su svi poligoni od materijala istih svojstava, ili se može raditi prije crtanja svakog poligona, ako je svaki poligon od drugačijeg materijala – radi se o konfiguriranju svojstava materijala za što se koristi naredba glMaterial. Primjer je prikazan u nastavku.

```
1 gl2.glMaterialfv(GL2.GL_FRONT, GL2.GL_AMBIENT, new float[] {1f,1f,1f,1f}, 0);
2 gl2.glMaterialfv(GL2.GL_FRONT, GL2.GL_DIFFUSE, new float[] {1f,1f,1f,1f}, 0);
3 gl2.glMaterialfv(GL2.GL_FRONT, GL2.GL_SPECULAR, new float[] {0.01f,0.01f,0.01f,1f}, 0);
4 gl2.glMaterialf(GL2.GL_FRONT, GL2.GL_SHININESS, 96f); // prihvatljivo: od 0 do 128
```

Redak 1 definira da je materijal od kojeg je izrađena prednja strana poligona takav da odbija primljenu ambijentnu svijetlost opisanu koeficijentima $(1, 1, 1)$ – drugim riječima, obasjan bijelom ambijentnom svjetlošću, materijal će također djelovati bijelo jer sve tri komponente u potpunosti ponovno odbija dalje. Retci 2 i 3 definiraju koeficijente za difuznu komponentu i zrcalnu komponentu dok redak 4 definira potenciju koja se primjenjuje na kosinusni član u izrazu za zrcalnu komponentu (sjaj).

Konačno, ostalo je definirati treba li OpenGL raditi konstantno sjenčanje ili intenzitet računati u svakom vrhu pa ga potom interpolirati za sve točke površine poligona. Odabir se radi naredbom glShadeModel. Ako se preda konstanta GL_FLAT, OpenGL će raditi konstantno sjenčanje (ali intenzitet neće računati u centru poligona već će uzeti intenzitet jednog od vrhova; pogledati dokumentaciju za detalje). Ako se preda GL_SMOOTH, OpenGL će raditi interpolaciju i dobit ćemo Gouraudovo sjenčanje.

Prilikom zadavanja poligona sada za svaki vrh trebamo napraviti dva poziva: naredbom glNormal3x (primjerice glNormal3f) trebamo najprije predati normalu koju treba koristiti za izračune osvjetljavanja u tom vrhu, pa potom naredbom glVertex3x (primjerice glVertex3f) treba predati koordinate vrha. To znači da će crtanje jednog trokuta izgledati poprilici kako slijedi.

```
1  gl2.glBegin(GL2.GL_POLYGON);
2    // Prvi vrh:
3    gl2.glNormal3f(...);
4    gl2.glVertex3f(...);
5    // Drugi vrh:
6    gl2.glNormal3f(...);
7    gl2.glVertex3f(...);
8    // Treci vrh:
9    gl2.glNormal3f(...);
10   gl2.glVertex3f(...);
11 gl2.glEnd();
```

Važno je napomenuti da OpenGL sam ne računa normale – normale koje treba koristiti moraju mu se eksplicitno predati pozivom naredbe glNormal. Također, OpenGL očekuje da su sve normale koje se predaju normirane. Ako to nije slučaj, izračuni koji će se provoditi dat će pogrešne rezultate. Teoretski, ovo se može ispraviti tako da se u OpenGL-u uključi normalizacija normala no to može dosta usporiti rad pa se preporuča da korisnik sam provede normalizaciju tamo gdje je ona potrebna, kao što smo u prethodnim koracima i napravili.

9.1 Pitanja

1. Objasnite Phongov model osvjetljavanja – koje komponente uzima u obzir te kako se one računaju.

2. Objasnite kako se provodi konstantno sjenčanje. Gdje se računa intenzitet. Možete li se domisliti zašto baš tamo?

3. Kako u OpenGL-u koristeći ugrađeni mehanizam osvjetljavanja postići efekt konstantnog sjenčanja? Što je potrebno konfigurirati?

4. Objasnite kako se provodi Gouraudovo sjenčanje. Gdje se računaju intenziteti? Kako se utvrđuju intenziteti za svaku točku površine poligona?

5. Kako u OpenGL-u koristeći ugrađeni mehanizam osvjetljavanja postići efekt Gouraudovog sjenčanja? Što je potrebno konfigurirati?

6. Objasnite kako se provodi Phongovo sjenčanje. Kako se utvrđuju intenziteti za svaku točku površine poligona?

7. Kako se u OpenGL-u konfiguriraju izvori svjetlosti a kako svojstva materijala?

8. Hoće li OpenGL za nas računati normale?
9. Hoće li OpenGL za nas normirati normale?
10. Kako se računa kosinus kuta između dva vektora?
11. Koji se vektori razmatraju prilikom izračuna difuzne komponente?
12. Koji se vektori razmatraju prilikom izračuna zrcalne komponente?
13. Kako se računa reflektirani vektor?
14. Kada je kosinus kuta između dva vektora jednak skalarnom produktu tih vektora?
15. Kada se za izračun kosinusa kuta između dva vektora ne treba dijeliti s umnoškom normi vektora?
16. Kako se definiraju prednji a kako stražnji poligoni?
17. Na koji se način može postići odbacivanje stražnjih poligona?
18. Kako funkcionira z-spremnik?
19. Možemo li uporabom z-spremnika postići da se kod žičanog modela scene sakriju stražnji poligoni te prednji poligoni koje (teoretski) ne bismo smjeli vidjeti? Objasnite.

9.2 Zadatak

U okviru ove vježbe ostvarit ćete prikaz 3D scene koristeći konstantno sjenčanje te Gouraudovo sjenčanje. To ćete ostvariti na dva načina: bez ugrađenog modela osvjetljavanja (dakle koristeći isključivo naredbu glColor za zadavanje boje) te koristeći ugrađeni model osvjetljavanja. U oba slučaja, vaš program treba pratiti pritiske tipki na tipkovnici. Ako korisnik pritisne tipku `k`, scenu treba prikazati uporabom konstantnog sjenčanja. Ako korisnik pritisne tipku `g`, scenu treba prikazati uporabom Gouraudovog sjenčanja. Hoće li se prilikom iscrtavanja scene koristiti z-spremnik, to treba regulirati posebna boolean zastavica koja ako je postavljena na **`true`** znači da prilikom crtanja scene još treba i koristiti z-spremnik. Pritisak na tipku `z` invertira trenutno stanje te zastavice, u konzolu ispisuje je li z-spremnik uključen ili nije, i nanovo crta scenu. Tipke `r` i `l` trebaju omogućiti rotiranje očišta, kako je to implementirano u prethodnim vježbama.

Modificirajte strukturu Vertex3D tako da omogućite pamćenje normale koja je pridružena tom vrhu. Dodajte u razred ObjectModel metodu koja za svaki vrh računa pridruženu normalu. Kao normalu vrha treba postaviti normiranu aritmetičku sredinu normiranih normala ravnina poligona koji dijele taj vrh. Pozovite tu metodu nakon što ste tijelo normalizirali i nakon što ste izračunali normale ravnina.

9.2.1 Osvjetljavanje scene uporabom OpenGL-a

Iskoristite programski kod koji ste već napravili za prethodnu vježbu u kojem crtate žičani model pri čemu posao transformacije pogleda, projekcije i uklanjanja stražnjih poligona prepuštate OpenGL-u; tamo ste već dodali i funkcionalnost rotiranja očišta. Modificirajte kod tako da dodate mogućnost uporabe z-spremnika. Objekt koji se prikazuje i dalje se zadaje kao argument komandne linije. Uključite OpenGL podršku za osvjetljavanje, postavite globalni ambijentni intenzitet na 0. Iskoristite jedan izvor svjetlosti i podesite ga kao što je u primjeru u ovoj uputi (poziciju i pripadne intenzitete); uključite ga. Svojstva materijala podesite također kao što je u ovoj uputi. Podešavanje pozicije izvora napravite nakon što je definirana matrica modela (tj. nakon što je definirano kako se radi transformacija pogleda). Potom, ovisno treba li raditi konstantno sjenčanje ili Gouraudovo sjenčanje, podesite to pozivom metode glShadeModel. Ovisno o tome treba li koristiti z-spremnik, uključite ga ili isključite pozivima glEnable odnosno glDisable. Za crtanje poligona koristite primitiv GL_POLYGON ali ga konfigurirajte tako da se poligon popunjava bojom, a ne da se crta njegov rub. Konačno, za svaki trokut, prije

no što zadate vrh trokuta zadajte pripadnu normalu (ovisno o modelu koji koristite; za konstantno sjenčanje svaki puta predajte normalu ravnine u kojoj leži poligon, za Gouraudovo sjenčanje za svaki vrh predajte izračunatu normalu vrha).

9.2.2 Osvjetljavanje scene bez uporabe ugrađenog modela

Iskoristite program koji ste razvili za prethodnu vježbu u kojem ste samostalno radili transformaciju pogleda, projekciju te izračun koji su poligoni prednji a koji stražnji. Tamo ste već dodali i funkcionalnost rotiranja očišta. Modificirajte kod tako da dodate mogućnost uporabe z-spremnika; vrhove morate predavati kao 3D točke. Objekt koji se prikazuje i dalje se zadaje kao argument komandne linije.

Pretpostavite da u sceni imate jedan izvor čiji su parametri kao u primjeru u ovoj uputi. Pretpostavite da su poligoni napravljeni od materijala čija su svojstva kao u primjeru u ovoj uputi. Provedite sami sve potrebne izračune. Kod konstantnog sjenčanja, prilikom crtanja poligona, pozvat ćete naredbu glColor kojom ćete definirati boju i potom ćete predati vrhove poligona.

```
1  gl2.glBegin(GL2.GL_POLYGON);
2    // Boja za popunjavanje poligona:
3    gl2.glColor3f(...);
4    // Prvi vrh:
5    gl2.glVertex3f(...);
6    // Drugi vrh:
7    gl2.glVertex3f(...);
8    // Treci vrh:
9    gl2.glVertex3f(...);
10 gl2.glEnd();
```

Kod Gouraudovog sjenčanja izračunat ćete intenzitete u svakom vrhu poligona i potom ćete za svaki vrh najprije pozvati naredbu glColor kojom ćete zadati boju vrha a potom naredbu glVertex kojom ćete zadati koordinate vrha, kako je prikazano u nastavku.

```
1  gl2.glBegin(GL2.GL_POLYGON);
2    // Prvi vrh - boja i koordinate:
3    gl2.glColor3f(...);
4    gl2.glVertex3f(...);
5    // Drugi vrh - boja i koordinate:
6    gl2.glColor3f(...);
7    gl2.glVertex3f(...);
8    // Treci vrh - boja i koordinate:
9    gl2.glColor3f(...);
10   gl2.glVertex3f(...);
11 gl2.glEnd();
```

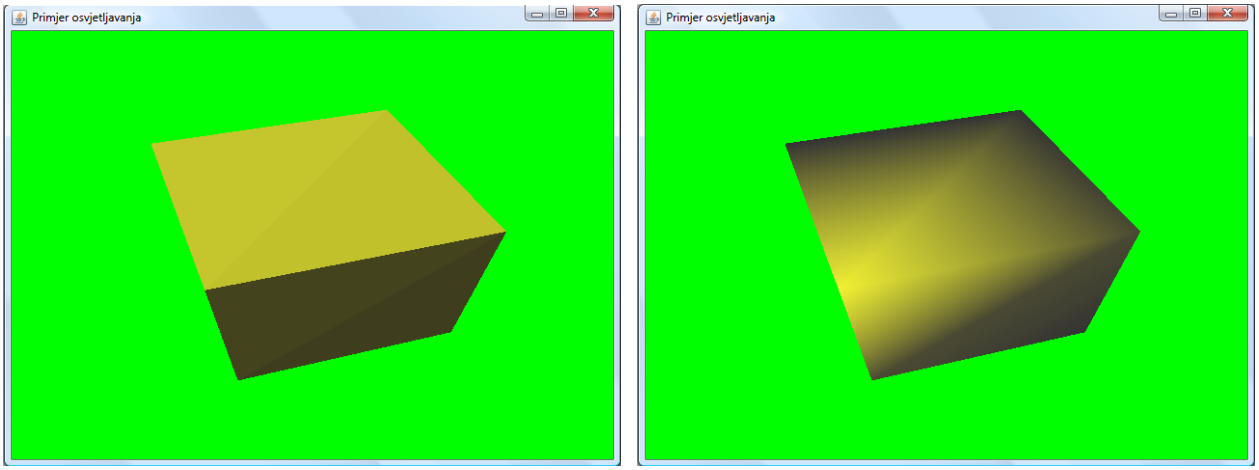
Ako s naredbom glShadeModel konfiguriranje uporabu interpolacije (argument GL_SMOOTH što i jest početna vrijednost pa je ne treba niti mijenjati), OpenGL će sam dalje provesti postupak interpolacije boje kroz čitavu površinu poligona. Kada sami provodite sve izračune, ugrađeno osvjetljavanje ne smije biti uključeno (ne pozivati glEnable(GL_LIGHTNING)), ne radite konfiguraciju izvora niti materijala (metode glLight te glMaterial) i prilikom zadavanja vrhova poligona ne predajete normale; sve izračune radite sami i za bojanje koristite glColor.

9.2.3 Ogledni primjeri

Ako ste sve napravili dobro, rezultat u oba slučaja (uz OpenGL-ovo sjenčanje ili uz Vaše izračune) mora biti isti. Uz uobičajen položaj očišta i gledišta te podešenu perspektivnu projekciju kao u prethodnim vježbama te uz parametre izvora i materijala kao u primjerima u ovoj uputi rezultati su prikazani na slikama 9.3, 9.4 i 9.5.

Slika 9.3 prikazuje scenu u kojoj se nalazi jedan konveksni lik (kocka). U tom slučaju, algoritam uklanjanja stražnjih poligona je dovoljan kako se na slici ne bi pojavili poligoni koji ne bi smjeli biti

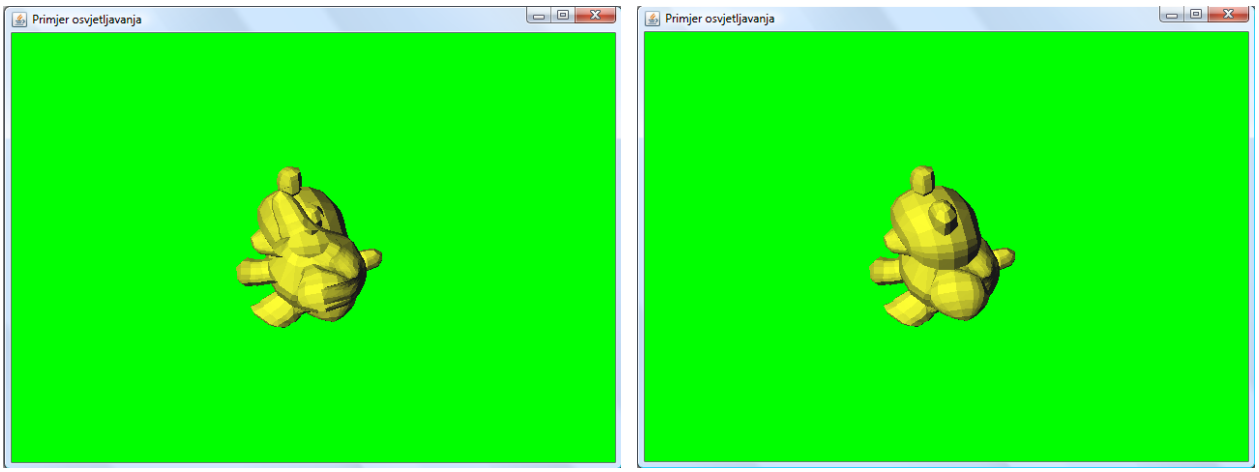
vidljivi. Stoga je rezultat isti neovisno o tome koristi li se z-spremnik ili ne. Slika 9.3a prikazuje uporabu konstantnog sjenčanja; granica između pojedinih poligona je jasno vidljiva, a svaki poligon je popunjen uniformno. Slika 9.3b prikazuje istu scenu uz Gouraudovo sjenčanje. Sada je lijepo vidi kako se intenziteti interpoliraju kroz površinu poligona. Također, granice između susjednih poligona manje su izražene zbog načina kako se računaju normale u vrhovima. Slika 9.4 prikazuje sjenčanje



(a) Uz konstantno sjenčanje. (b) Uz Gouraudovo sjenčanje.

Slika 9.3: Primjer vizualizacije kocke uz različite vrste sjenčanja.

konkavnog tijela (medvjedić) i to uporabom konstantnog sjenčanja. S obzirom da je tijelo konkavno, algoritam uklanjanja stražnjih poligona ne uspjeva ukloniti sve nevidljive poligone; stoga je jasno vidljiva razlika između scene koja je prikazana bez uporabe z-spremnika i scene koja je prikazana uz uporabu z-spremnika. Konačno, slika 9.5 prikazuje sjenčanje konkavnog tijela (medvjedić) i to



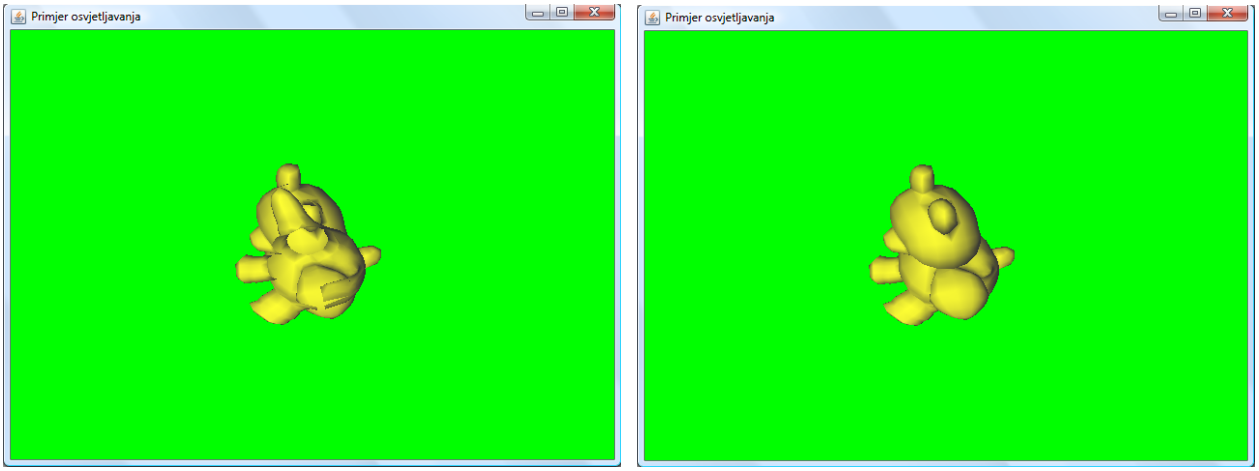
(a) Bez z-spremnika. (b) Uz z-spremnik.

Slika 9.4: Primjer vizualizacije medvjedića uz konstantno sjenčanje.

uporabom Gouraudovog sjenčanja, opet sa i bez uporabe z-spremnika.

Linkovi

- glEnable - `http://www.opengl.org/sdk/docs/man/xhtml/glEnable.xml`
- glDisable - `http://www.opengl.org/sdk/docs/man/xhtml/glDisable.xml`
- glNormal - `http://www.opengl.org/sdk/docs/man/xhtml/glNormal.xml`



(a) Bez z-spremnika. (b) Uz z-spremnik.

Slika 9.5: Primjer vizualizacije medvjedića uz Gouraudovo sjenčanje.

- glLight - `http://www.opengl.org/sdk/docs/man/xhtml/glLight.xml`
- glMaterial - `http://www.opengl.org/sdk/docs/man/xhtml/glMaterial.xml`
- glLightModel - `http://www.opengl.org/sdk/docs/man/xhtml/glLightModel.xml`
- glShadeModel - `http://www.opengl.org/sdk/docs/man/xhtml/glShadeModel.xml`
- glPolygonMode - `http://www.opengl.org/sdk/docs/man/xhtml/glPolygonMode.xml`

Laboratorijska vježba 10

Algoritam praćenja zrake

U okviru ove vježbe napravit ćemo program za vizualizaciju 3D scene uporabom algoritma praćenja zrake. Kao prvi korak na tom putu detaljno proučite poglavlje 10 u knjizi.

Opis scene zadat ćemo preko tekstovne datoteke u kojoj će biti navedeni svi podatci potrebni za pozicioniranje promatrača, svi izvori svjetlosti sa svojim karakteristikama te svi objekti koji se nalaze u sceni sa svojstvima materijala iz kojih su napravljeni. Slijedi primjer datoteke.

```
e 6 10 7 # ociste
v -1 -1 -0.5 # view vektor
vu 0 0 1 # view-up vektor
h 3  # udaljenost ravnine prikaza od ocista
xa 60  # horizontalni kut gledanja
ya 60  # vertikalni kut gledanja
ga 0.1 0.1 0.1 # globalna ambijentna svjetlost, r g b
i 0 -10 1 0.1 0.1 0.1 # x y z r g b
i 10 10 10 0.5 0.5 0.5 # x y z r g b
i 10 0 10 0.5 0.5 0.5 # x y z r g b

# slijede tri kugle...
# kugla: cx cy cz r ar ag ab dr dg db rr rg rb n kref
o s 2 2 2 1 0 0.4 0.8 0 0.4 0.8 0.3 0.3 0.3 96 0.2
o s 6 3 0 2 0 0.6 0.6 0 0.6 0.6 0.3 0.3 0.3 96 0.2
o s 2 6 5 2 0.7 0.7 0 0.7 0.7 0 0.3 0.3 0.3 16 0.2

# slijede dvije krpice...
# krpica cx cy cz v1x v1y v1z v2x v2y v2z wi he ar ag ab dr dg db rr rg rb n kref
                                                ar ag ab dr dg db rr rg rb n kref
# Prethodni (i sljedeca dva) retka su slomljena samo zbog ogranicene sirine skripte.
# U datoteci ce to biti u jednom retku (jedan redak, jedna krpica).
o p 0 0 -2 1 0 0 0 1 0 16 16 0.6 0.2 0.2 0.6 0.2 0.2 0.3 0.3 0.3 32
                             0 1 1 1 1 1 1 0.3 0.3 0.3 32 0
o p -8 0 2 0 1 0 0 0 1 20 20 1 1 1 0.1 0.0 0.0 0.0 0.0 0.0 32 0.7
                             1 1 1 1 1 1 0.3 0.3 0.3 32 1

# kod krpice vektori v1 i v2 moraju biti zadani tako da nisu kolinearni i da je
# normala koja se dobije kao v1 x v2 pokazuje prema zeljenoj prednjoj strani
# ravnine, takoder, prvi skup parametara osvjetljavanja je za prednju stranu,
# drugi skup je za straznju stranu
```

Svaki redak datoteke sastoji se od niza elemenata koji su odvojeni jednim ili više razmakom ili znakom tabulatora. Prvi element u retku definira značenje tog retka.

Znak	Značenje
`#`	Komentar. Ovaj znak može doći bilo gdje i u drugim retcima i sve od tog znaka do kraja retka treba smatrati komentarom.
`e`	Pozicija očišta (*eye*). Slijede x, y i z koordinata očišta.
`v`	Vektor pogleda (*view*). Određuje smjer u kojem gledamo. Ravnina prikaza je okomita na taj vektor. Slijede x, y i z komponente vektora.
`vu`	Vektor pogleda prema gore (*view-up*). Određuje smjer u kojem će se protezati y-os u ravnini prikaza (projekcija tog vektora na ravninu prikaza bit će kolinearna s y-osi). Slijede x, y i z komponente vektora.
`h`	Definira na kojoj se udaljenosti od očišta nalazi ravnina prikaza. Slijedi jedan broj (udaljenost).
`xa`	Definira vodoravno vidno polje i zadano je u stupnjevima. Primjerice, ako je zadan kut 60°, to znači da u odnosu na smjer vektora pogleda vidimo još 30° u lijevo i 30° u desno. Slijedi jedan broj (kut).
`ya`	Definira okomito vidno polje i zadano je u stupnjevima. Primjerice, ako je zadan kut 60°, to znači da u odnosu na smjer vektora pogleda vidimo još 30° prema dolje i 30° prema gore. Slijedi jedan broj (kut).
`ga`	Koliko iznosi globalno ambijentno osvjetljenje. Bilo koji objekt bit će osvjetljen barem toliko, čak ako u sceni nema nikakvih dodatnih izvora svjetlosti. Slijede tri komponente (r, g i b).
`i`	Definicija izvora. Prva tri broja određuju položaj izvora u prostoru (x, y i z koordinatu) a sljedeća tri broja određuju intenzitete kojima izvor zrači crvenu, zelenu i plavu boju (r, g i b).
`o`	Definicija objekta. Sljedeći element određuje vrstu objekta; ako je `s`, radi se o kugli a ako je `p`, radi se o krpici površine.

U okviru ove vježbe radit ćemo s dvije vrste objekata koje mogu biti zadane u tekstovnoj datoteci. Ako je objekt kugla, u retku u kojem je zadan redom će biti definirano: centar kugle (cx, cy i cz), radijus kugle (r), koeficijenti uz ambijentnu komponentu svjetlosti (i to po komponentama, ar, ag i ab), koeficijenti uz difuznu komponentu svjetlosti (i to po komponentama, dr, dg i db), koeficijenti uz zrcalnu komponentu svjetlosti (i to po komponentama, rr, rg i rb), koeficijent koji određuje sjaj (n) te koeficijent koji određuje u kojoj se mjeri reflektira svjetlost koja je došla s drugih objekata.

Krpice površine zadaju se navođenjem centralne točke površine ($\vec{c} = cx, cy, cz$), dvaju nekolinearnih vektora koji razapinju tu površinu ($\vec{v}_1 = v_{1x}, v_{1y}, v_{1z}$ i $\vec{v}_2 = v_{2x}, v_{2y}, v_{2z}$) te širinom wi krpice u smjeru $\vec{v}_1$ i visinom he krpice u smjeru $\vec{v}_2$. Svaka točka krpice jednoznačno je određena izrazom:

$$\vec{p}(\lambda, \mu) = \vec{c} + \lambda \cdot \vec{v}_1 + \mu \cdot \vec{v}_2.$$

Pri tome, krpici pripadaju samo točke za koje je $-\frac{wi}{2} \leq \lambda \leq \frac{wi}{2}$ i $-\frac{he}{2} \leq \mu \leq \frac{he}{2}$.

Normala krpice uvijek se računa kao vektorski produkt $\vec{n} = \vec{v}_1 \times \vec{v}_2$. Smjer u kojem tako izračunata normala pokazuje predstavlja prednju stranu krpice. Za krpicu se potom definiraju parametri za izračun ambijentne, difuzne i zrcalne komponente svjetlosti te koeficijent koji određuje u kojoj se mjeri reflektira svjetlost koja je došla s drugih objekata; svi ovi parametri navedeni su dva puta – prvi puta za prednju stranu krpice a drugi puta za stražnju stranu krpice. Koji skup parametara ćete koristiti ovisit će o tome pogađa li zraka svjetlosti prednju ili stražnju stranu krpice.

U knjizi su izvedeni izrazi kojima se uspostavlja koordinatni sustav te koji za svaku (x, y) koordinatu ravnine određuju točku u 3D prostoru gdje se nalazi taj slikovni element.

Također, podsjetite se poglavlja 2 u knjizi i načina kako se računa reflektirani vektor te načina na koji se računa probodište zrake i kugle. Probodište zrake i krpice možete rješiti na sljedeći način.

Točke koje pripadaju ravnini u kojoj leži krpica određene su izrazom:

$$\vec{p}(\lambda, \mu) = \vec{c} + \lambda \cdot \vec{v}_1 + \mu \cdot \vec{v}_2.$$

Točke koje pripadaju zraci koja kreće iz točke $\vec{s}$ i ima vektor smjera $\vec{d}$ određene su izrazom:

$$\vec{p}(\epsilon) = \vec{s} + \epsilon \cdot \vec{d}.$$

U sjecištu tada vrijedi:

$$\vec{c} + \lambda \cdot \vec{v}_1 + \mu \cdot \vec{v}_2 = \vec{s} + \epsilon \cdot \vec{d},$$

odnosno:

$$\lambda \cdot \vec{v}_1 + \mu \cdot \vec{v}_2 - \epsilon \cdot \vec{d} = \vec{s} - \vec{c}.$$

Kako smo u 3D prostoru, to je sustav od tri jednadžbe s tri nepoznanice (sustav vrijedi posebno za svaku od komponenti x, y i z).

$$\begin{bmatrix} v_{1x} & v_{2x} & -d_x \\ v_{1y} & v_{2y} & -d_y \\ v_{1z} & v_{2z} & -d_z \end{bmatrix} \cdot \begin{bmatrix} \lambda \\ \mu \\ \epsilon \end{bmatrix} = \begin{bmatrix} s_x - c_x \\ s_y - c_y \\ s_z - c_z \end{bmatrix}$$

Napišite kod koji rješava zadani sustav (primjerice, Cramerovim pravilom), i utvrdite λ, μ i ϵ. Sjecište, ako postoji, pripada krpici samo ako vrijedi $-\frac{wi}{2} \leq \lambda \leq \frac{wi}{2}$ i $-\frac{he}{2} \leq \mu \leq \frac{he}{2}$. Sjecište treba odbaciti ako je $\epsilon < 0$ jer je takvo sjecište iza nas i mi ga ne vidimo.

Izrazi u knjizi koji se koriste za uspostavu koordinatnog sustava očekuju definirano gledište te parametre l, r, b i t. Temeljem parametara koji su zadani u datoteci, gledište možemo izračunati kao:

$$\vec{G} = \vec{E}ye + h \cdot \frac{\vec{v}}{\|\vec{v}\|}.$$

Parametre l i r trivijalno je izračunati iz zadane udaljenosti h i zadanog vodoravnog vidnog polja a parametre b i t iz zadane udaljenosti h i zadanog okomitog vidnog polja. S obzirom da su vidna polja u oba smjera simetrična, vrijedi $l = r$ i $t = b$ – izvedite izraz za ove vrijednosti temeljem vrijednosti h i zadanih kuteva.

10.1 Pitanja

1. Kako glasi parametarska jednadžba pravca?
2. Kako glasi jednadžba kugle u 3D prostoru?
3. Koja je razlika između pojmova "pravac" i "zraka"?
4. Kako se definiraju točke koje pripadaju zraci?
5. Kako se računa presjecište zrake i kugle? Koliko ih ima i kako ih kategoriziramo?
6. Kako se računa presjecište zrake i ravnine koja je zadana kao u ovoj vježbi?
7. Kada presjecište zrake i ravnine pripada krpici?
8. Opišite algoritam bacanja zrake.
9. Opišite algoritam praćenja zrake.
10. Koja je razlika između algoritma bacanja zrake i algoritma praćenja zrake?
11. Kako se prema Phongovom modelu osvjetljavanja računa intenzitet kojim neki djelić površine zrači?
12. Kako se računa reflektirani vektor $\vec{r}$ zadanog vektora $\vec{l}$ u odnosu na neki drugi zadani vektor $\vec{n}$?

10.2 Zadatak

U okviru ove vježbe Vaš je zadatak napraviti program koji će kao argument komandne linije primiti naziv datoteke s opisom scene te koji će potom algoritmom praćenja zrake obaviti prikaz zadane scene. Datoteka će uvijek biti formatirana kako je opisano u primjeru, s time da broj likova i izvora može varirati, a i redosljed redaka u datoteci nije bitan (ne mora prvi redak uvijek biti redak s informacijom o položaju očišta). Proučite pseudok algoritma koji je dan u knjizi i napisite rješenje prema njemu. Pri tome morate uzeti u obzir zrake reflektirane zrake, ali nije potrebno računati lomljene zrake.

Preporuka je da napravite sljedeće razrede: Light koji modelira jedan izvor svjetlosti, SceneObject koji je apstraktni model objekta koji je prisutan u sceni, Patch koji predstavlja model krpice, Sphere koji predstavlja model kugle, RTScene koji predstavlja čitavu scenu te Intersection koji predstavlja jedno sjecište. Strukture razreda prikazane su u nastavku.

```
1  class Light {
2    IVector position;
3    double[] rgb;
4  }
```

```
1  abstract class SceneObject {
2    // parametri prednje strane objekta
3    double[] fambRGB;
4    double[] fdifRGB;
5    double[] frefRGB;
6    double fn;
7    double fkref;
8    // parametri straznje strane objekta
9    double[] bambRGB;
10   double[] bdifRGB;
11   double[] brefRGB;
12   double bn;
13   double bkref;
14
15   // Apstraktne metode koje ce definirati konkretni modeli
16   public abstract void updateIntersection(Intersection inters, IVector start, IVector d);
17   public abstract IVector getNormalInPoint(IVector point);
18 }
```

```
1  class Patch extends SceneObject {
2    IVector center;
3    IVector v1;
4    IVector v2;
5    IVector normal;
6    double w;
7    double h;
8
9    public void updateIntersection(Intersection inters, IVector start, IVector d) {...}
10   public IVector getNormalInPoint(IVector point) {...}
11 }
```

```
1  class Sphere extends SceneObject {
2    IVector center;
3    double radius;
4
5    public void updateIntersection(Intersection inters, IVector start, IVector d) {...}
6    public IVector getNormalInPoint(IVector point) {...}
7  }
```

```
1  class RTScene {
2    // Parametri iz datoteke
3    IVector eye;
4    IVector view;
5    IVector viewUp;
6    double h;
```

```
7     double xAngle;
8     double yAngle;
9     double[] gaIntensity = new double[] {0, 0, 0};
10    Light[] sources;
11    SceneObject[] objects;
12
13    // Izracunati parametri
14    IVector xAxis;
15    IVector yAxis;
16    double l;
17    double r;
18    double b;
19    double t;
20
21    // Metoda koja racuna xAxis, yAxsis, l, r, b i t
22    private void computeKS() {...}
23
24    // Metoda koja ucitava scenu i na kraju poziva computeKS();
25    public static RTScene ucitajScenu(Path path) {...}
26 }
```

```
1  class Intersection {
2    SceneObject object; // najblizi objekt s kojim se zraka sjece
3    double lambda; // Za koji se labda to dogada?
4    boolean front; // Je li to sjeciste na prednjoj strani objekta?
5    IVector point; // U kojoj je tocki to sjeciste?
6  }
```

U razredu SceneObject definirana je apstraktna metoda updateIntersection. Ideja metode je da izračuna najbliže presjecište objekta i predane zrake i potom napravi ažuriranje informacije samo ako u predanom objektu intersection još nema evidentiranog sjecišta, ili ako je pronađeno sjecište bliže od onog evidentiranog.

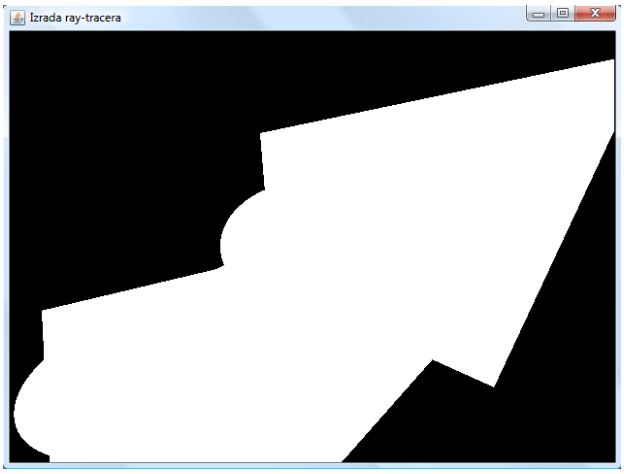
Rješavanju problema pristupite tako da napravite najjednostavniji OpenGL program u kojem projekcijsku matricu podesite na ortografsku punog raspona širine i visine prozora; viewport također treba biti takav da sliku prikazuje preko čitavog prozora. Konačno, matrica modela treba biti jedinična.

Dalje radite prema pseudokodu iz knjige: za svaki x od 0 do širine prozora i za svaki y od 0 do visine prozora izračunajte 3D koordinate $\vec{v}$ pripadne točke (x, y) u ravnini prikaza. Vaša zraka kreće iz očišta i prolazi kroz točku $\vec{v}$ – time je definiran početak zrake i smjer. Napišite metodu slijedi; metodu utvrdi_boju najprije napravite tako da Vam uvijek vrati bijelu boju ($r = g = b = 1$) i pokrenite program. Trebali biste dobiti prikaz kao na slici 10.1a. Na ovaj način radite testiranje računate li dobro presjecišta zrake i objekata u sceni. Tamo gdje zraka koja kreće iz očišta ne siječe niti jedan objekt, imat ćete crni slikovni element, a ako zraka siječe neki objekt, imat ćete bijeli slikovni element. Ako imate kakvih problema, pogledajte pred kraj ove upute gdje se diskutiraju dva tipična problema.

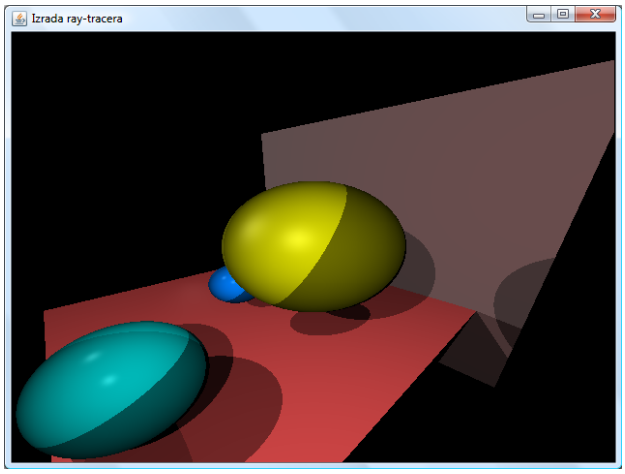
Sada pokušajte doraditi metodu utvrdi_boju. Za svaki izvor svjetlosti stvorite zraku koja ide od pronađenog sjecišta do trenutnog izvora. Pronađite najbliže sjecište te zrake sa svim objektima scene. Ako ste pronašli sjecište koje je ispred polazne točke i koje je bliže polaznoj točki no što je trenutni izvor, preskočite taj izvor – on je skriven objektom koji ste upravo pronašli. Inače izračunajte intenzitet kojim taj izvor osvjetljava izvorno sjecište, i te vrijednosti akumulirajte za sve izvore koji nisu u sjeni. Vratite tako utvrđeni intenzitet. Ako ste to dobro napravili, dobit ćete prikaz kao na slici 10.1b. Ako to ne uspjevate dobiti, popravite kod i ne krećite dalje dok Vaš prikaz ne odgovara onome sa slike.

Konačno, sljedeći korak je omogućiti utjecaj reflektirane zrake. Doradite metodu utvrdi_boju tako što ćete za dolaznu zraku i sjecište te normalu u tom sjecištu izračunati reflektiranu zraku. Rekurzivno pozovite slijeđenje te zrake i rezultat koji dobijete skalirajte pa pridodajte trenutnom intenzitetu. Skaliranje radite s koeficijentom refleksije (kod kugle to je bio zadnji prametar, kod krpice je za svaku stranu bio naveden zadnji). Ako implementirate kontrolu koliko duboko idete u rekurziju (nije nužno), tada ćete moći generirati prikaz kao što je na slici 10.1b uz postavljanje granične dubine na 0 (ne slijede se reflektirane zrake) te prikaz kao što je na slici 10.1c uz postavljanje granične dubine na 1 (samo

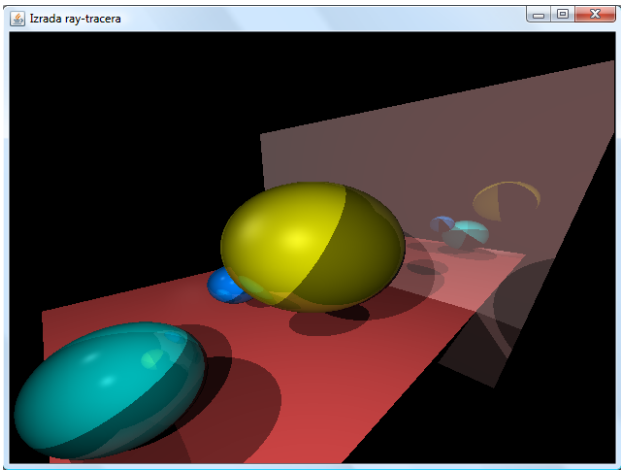
jednom slijedite reflektiranu zraku). Prikaz koji biste morali dobiti bez ikakvih ograničenja prikazan je na slici 10.1d. U datoteci s opisom scene je krpica koja je paralelna s yz-ravninom definirana kao visoko-reflektivna uz male ostale koeficijente tako da ona sama djeluje blago crvenkasto i zapravo se ponaša kao zrcalo.



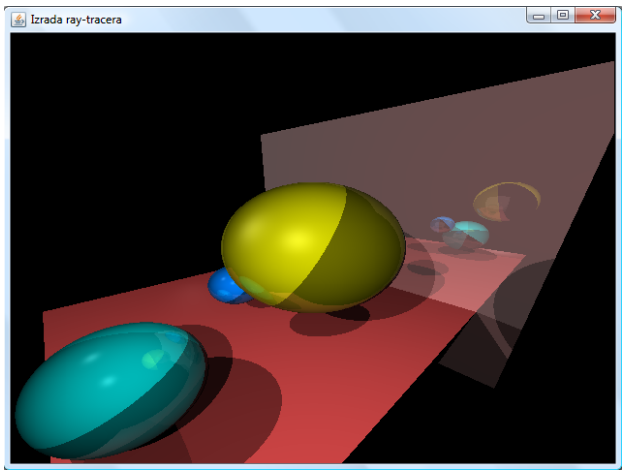
(a) Probodišta označena bijelom bojom.



(b) Prikaz uz dubinu praćenja 0.



(c) Prikaz uz dubinu praćenja 1.



(d) Prikaz uz neograničenu dubinu praćenja.

Slika 10.1: Primjeri prikaza scene algoritmom praćenja zrake uz različite postavke.

10.2.1 Tipični problemi

Ako pozovete gluOrtho2D uz širinu width i visinu height, dobit ćete prikaz kao na slici 10.2a – pojavit će se crne vodoravne crte. U tom slučaju metodi gluOrtho2D kao širinu predajte width−1 a kao visinu height−1; viewport pri tome ne dirajte – on mora ostati pune širine i visine.

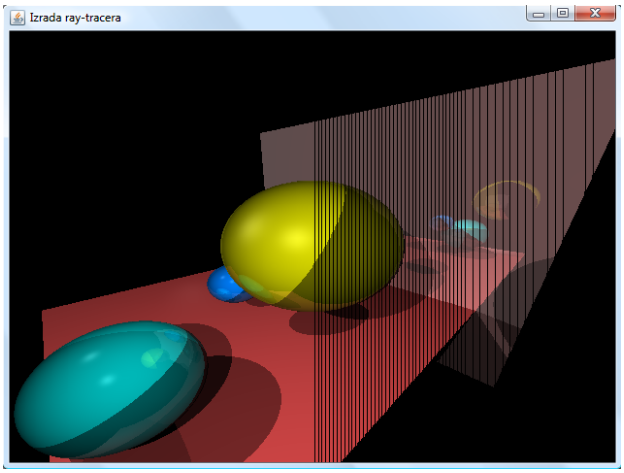
Puno nezgodniji problem prikazan je na slici 10.2b a rezultat je numeričkih problema. Svaki puta kada krećete slijediti neku zraku (a da to nije ona prva zraka iz očišta već je zraka koja kreće iz nekog sjecišta), početnu točku malo pomaknite u smjeru vektora zrake. Umjesto da slijedite zraku:

$$\vec{p}(\epsilon) = \vec{s} + \epsilon \cdot \vec{d}$$

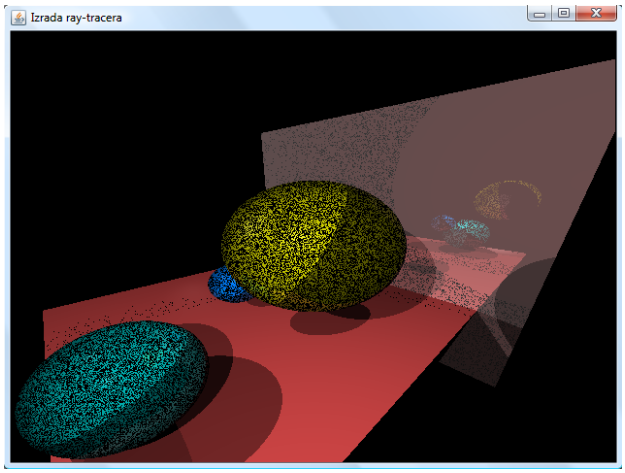
slijedite zraku:

$$\vec{p}(\epsilon) = (\vec{s} + \psi \cdot \vec{d}) + \epsilon \cdot \vec{d}$$

gdje je ψ mali pozitivni broj (primjerice 10^{-4}). Naime, ako to ne napravite, dogodit će Vam se da upravo objekt s kojeg upravo krećete postane najbliži objekt koji siječe zraku što nema smisla. Također, ako to ne napravite sa zrakom prema izvoru, povremeno će se događati "čudne" situacije.



(a) Loše podešen gluOrtho2D.



(b) Numerički problemi na djelu.

Slika 10.2: Dva primjera tipičnioh problema koji se mogu javiti.

Laboratorijska vježba 11

Fraktali

U okviru ove vježbe napravit ćemo program za prikaz nekoliko popularnih fraktala. Kao prvi korak na tom putu detaljno proučite poglavlje 13 u knjizi. Prvi fraktal koji spada u obavezni dio ove vježbe jest Mandelbrotov fraktal. U knjizi je u podpoglavlju 13.3 opisana teorijska podloga te je prikazan kostur implementacije programa koji obavlja prikaz; pažljivo to proučite.

Izborni dio ove vježbe jest prikaz fraktala koji nastaje kao rezultat iteriranja sustava funkcija (tzv. IFS-fraktal) ili fraktala koji nastaje iz L-sustava. "Izbornost" se ovdje koristi u smislu da možete odabrati jedan od ta dva fraktala; međutim, nakon što se odlučite za jedan od njih, morate napraviti program koji obavlja prikaz. Vježba će Vam biti priznata ako ste napravili:

- program za prikaz Mandelbrotovog fraktala i program za prikaz IFS fraktala, ili
- program za prikaz Mandelbrotovog fraktala i program za prikaz L-sustava.

U knjizi su IFS-fraktali opisani u podpoglavlju 13.5 a L-sustavi u podpoglavlju 13.6. Neovisno o tome što odaberete, očekuje se da ćete znati i teorijsku podlogu fraktala koji niste odabrali.

11.1 Pitanja

Krenut ćemo s pitanjima koja su vezana u Mandelbrotov fraktal.

1. Što je Mandelbrotov skup? Možete li ga pokazati na slici 11.1a odnosno na slici 11.1b?
2. Što je Mandelbrotov fraktal? Možete li ga pokazati na slici 11.1a odnosno na slici 11.1b? U kakvoj su vezi Mandelbrotov skup i Mandelbrotov fraktal?
3. Kako je matematički definiran Mandelbrotov skup? Napišite izraz i sve potrebne uvjete.
4. Kod Mandelbrotovog skupa spominje se pojam konvergencije i divergencije. Objasnite.
5. Je li moguće napisati računalni program koji će završiti u konačnom vremenu i koji će nacrtati točnu aproksimaciju Mandelbrotovog skupa? Objasnite.
6. Na koji se način na računalu prikazuje Mandelbrotov skup? U kakvoj su vezi pri tome "ekran računala" i kompleksna ravnina?
7. Proučite kako je u knjizi riješeno bojanje Mandelbrotovog skupa – što predstavljaju područja koja su obojana "šareno"? Što predstavlja pojedina boja?
8. Algoritam za ispitivanje divergencije radi s unaprijed zadanom ogradom na maksimalni broj pokušaja. Što se postiže povećavanjem te ograde odnosno kako se to odražava na dobiveni prikaz (posebice kod bojanja)?

Slijede pitanja vezana uz IFS-fraktal.

1. Na koji se način zadaju IFS-fraktali?
2. Kako tumačimo sadržaj tablice s funkcijama?
3. Što mora vrijediti za vjerojatnosti koje se definiraju uz svaku funkciju?
4. Kakvog je oblika svaka funkcija koja je zadana u tablici? Je li to skalarna funkcija?

Konačno, evo i pitanja vezana uz L-sustave.

1. Što je to L-sustav?
2. Što je DOL-sustav?
3. Kako je definiran DOL-sustav?
4. Uz L-sustave vezan je i pojam *turtle graphics*. O čemu se tu radi? Pojasnite.
5. Na koji se način prikazuje DOL-sustav? Što treba biti zadano osim samog sustava?
6. Što je potrebno ugraditi u sustav za prikazivanje L-sustava koji može crtati fraktale s grananjima?

11.2 Zadatak

U okviru ove vježbe trebate napraviti dva programa; jedan koji radi prikaz Mandelbrotovog fraktala te drugi koji radi prikaz IFS-fraktala ili L-sustava (prema Vašem izboru). U nastavku su dane upute za svaki od njih.

11.2.1 Mandelbrotov fraktal

U knjizi je u podpoglavlju 13.3 dan primjer implementacije funkcije koja ispituje divergenciju točke u kompleksnoj ravnini, a potom i kod koji pozivanjem te funkcije crta Mandelbrotov fraktal. U Vašem programu organizirajte glavnu petlju programa koja obrađuje svaki slikovni element na sljedeći način:

```
1 // za kompleksni broj c koji odgovara tocki (x,y):
2 int n = divergenceTest(c, maxLimit); // ili n = divergenceTest2(c, maxLimit);
3 colorScheme1(n); // ili colorScheme2(n);
4 glVertex2i(x, y);
```

Funkcija divergenceTest ispituje divergenciju s obzirom na uobičajeno definiranu kvadratnu funkciju. Funkcija colorScheme1 ovisno o vrijednosti n kao trenutnu boju postavlja vrijednost $rgb = (0f, 0f, 0f)$ ako nije utvrđena divergencija a inače vrijednost $rgb = (1f, 1f, 1f)$. Ovo će rezultirati crno-bijelim prikazom kao na slici 11.1a. Bojanje možete riješiti na različite načine, i za to treba biti zadužena funkcija colorScheme2. Prijedlog jedne takve funkcije dan je u nastavku.

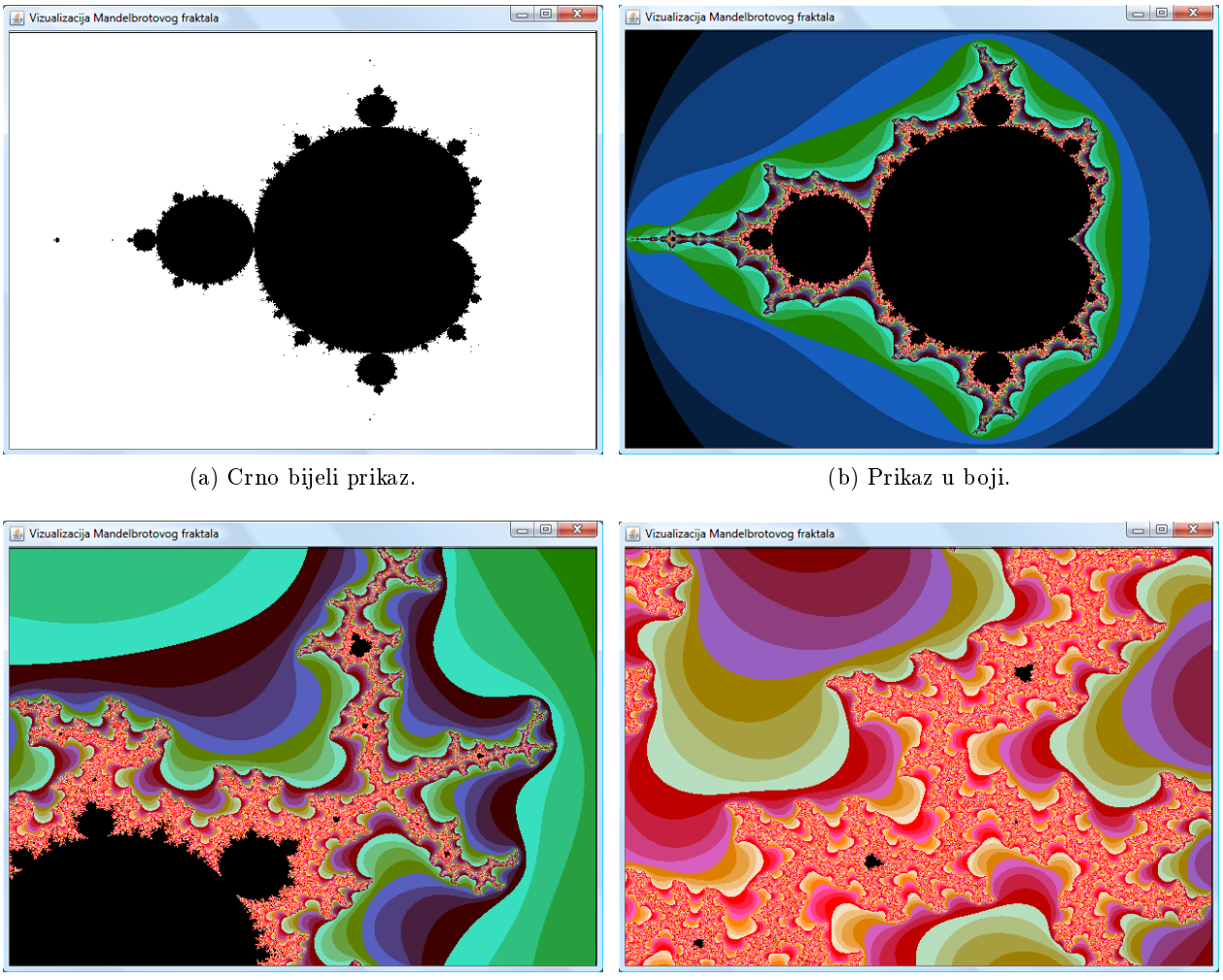
```
1  void colorScheme2(int n) {
2    if(n==-1) {
3      glColor3f(0f, 0f, 0f);
4    } else if(maxLimit < 16) {
5      int r = (int)((n-1)/(double)(maxLimit-1) * 255 + 0.5);
6      int g = 255 - r;
7      int b = ((n-1) % (maxLimit/2)) * 255 / (maxLimit/2);
8      glColor3f((float)(r/255f), (float)(g/255f), (float)(b/255f));
9    } else {
10     int lim = maxLimit < 32 ? maxLimit : 32;
11     int r = (n-1) * 255 / lim;
12     int g = ((n-1) % (lim/4)) * 255 / (lim/4);
```

```
13        int b = ((n-1) % (lim/8)) * 255 / (lim/8);
14        glColor3f((float)(r/255f), (float)(g/255f), (float)(b/255f));
15      }
16    }
```

Konačno, pripremite funkciju divergenceTest2 koja neće ispitivati divergenciju s obzirom na red $z_{n+1} = z_n^2 + c$ već s obzirom na red $z_{n+1} = z_n^3 + c$.

Inicijalno, program treba postaviti vrijednosti $u_{min} = -2$, $u_{max} = 1$, $v_{min} = -1.2$, $v_{max} = 1.2$, $maxLimit = 128$. Da biste dobili čišći kod, prijedlog je da napravite zaseban razred Complex koji će implementirati operacije dodavanja, oduzimanja, množenja, kvadriranja, kubiranja te dohvata modula i kvadrata modula kompleksnog broja. Uz takvu pripremu metode za ispitivanje divergencija bit će trivijalne.

Vaš program treba osluškivati pritiske na tipke tipkovnice te miša. Ako korisnik pritisne tipku `1`, odabire se crtanje fraktala koji se dobije za kvadratnu funkciju, a ako pritisne tipku `2`, odabire se crtanje fraktala koji se dobije za kubnu funkciju. Neovisno o tome, ako pritisne tipku `b` traži se crno-bijeli prikaz a ako pritisne tipku `c` traži se prikaz u boji. Kada korisnik klikne mišem, utvrdite koji kompleksni broj odgovara kliknutom slikovnom elementu i postavite nove vrijednosti u_{min}, u_{max}, v_{min} i v_{max} tako da dobijete područje koje je $\frac{1}{16}$ visine i širine trenutnog područja u kompleksnoj ravnini i koje je centrirano oko kliknute točke; ovo će Vam omogućiti da mišem zumirate željena područja. Prije no što korigirate vrijednosti za u_{min}, u_{max}, v_{min} i v_{max}, trenutne vrijednosti gurnite na stog tako da ostanu zapamćene. Pritiskom na tipku `x` sa stoga se trebaju skinuti vrijednosti za u_{min}, u_{max}, v_{min} i v_{max} čime se korisnik vraća na pogled koji je imao prije jednog klika mišem; ako je stog prazan,



(a) Crno bijeli prikaz.

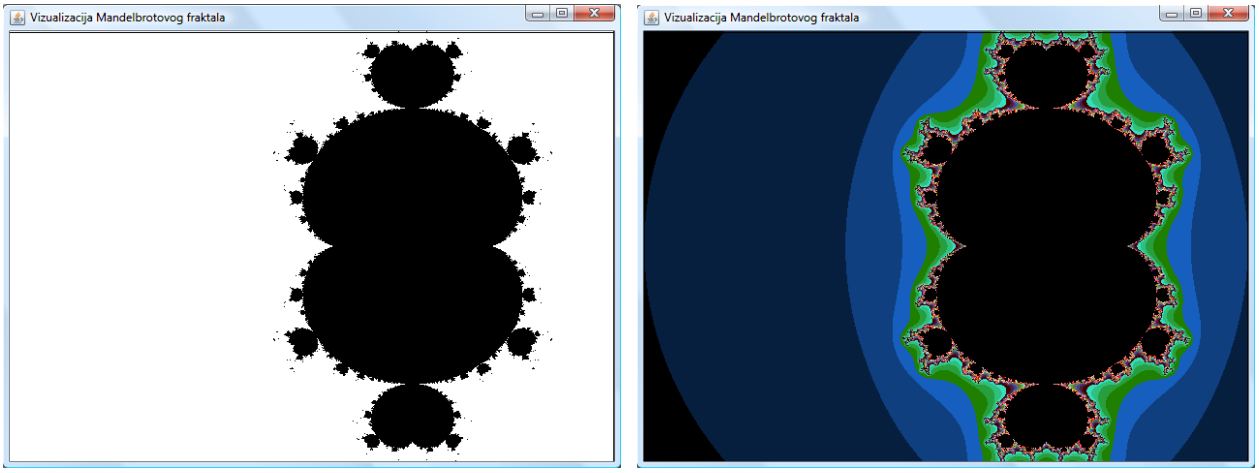
(b) Prikaz u boji.

(c) Prikaz u boji podpodručja.

(d) Prikaz u boji još manjeg podpodručja.

Slika 11.1: Prikazi Mandelbrotovog skupa i fraktala.

zanemarite pritisak. Pritiskom na tipku ESC za u_{min}, u_{max}, v_{min} i v_{max} se uzimaju početne vrijednost (one se nalaze na dnu stoga ako stog nije prazan; ako je, tada trenutni prikaz već jest inicijalni i ne treba napraviti ništa); dodatno, tom tipkom briše se sadržaj stoga. Slika 11.1 daje različite prikaze koji se mogu dobiti ovako napisanim programom. Na slici 11.2 dani su rezultati prikaza ako se ispituje divergencija uz kubni polinom.



(a) Crno-bijeli prikaz. (b) Prikaz u boji.

Slika 11.2: Prikazi fraktala koji nastaje uz kubni polinom.

11.2.2 IFS-fraktal

U knjizi je u podpoglavlju 13.5 dan je detaljni primjer implementacije funkcije koja crta jedan konkretan IFS-fraktal. Algoritam generira pointsNumber točaka (u knjizi u prikazanom algoritmu to je gornja granica petlje u retku 12; umjesto fiksne vrijednosti koristit ćemo varijablu pointsNumber. Za svaku točku radi se limit iteracija (u knjizi u prikazanom algoritmu vrijednost je fiksirana na 25). Nakon tog broja iteracija dobiva se jedna točka koju treba nacrtati na ekranu. Kako se sam fraktal može rasprostirati na relativno malom području, dobivena točka se linearno transformira kako bi se dobio prikaz preko čitavog zaslona. U algoritmu u knjizi to se radi u retku 36; pretpostavite sada da je opći izraz kojim se transformira x zadan kao $x \leftarrow \eta_1 \cdot x + \eta_2$ te da je opći izraz kojim se transformira y zadan kao $y \leftarrow \eta_3 \cdot y + \eta_4$.

Pripremite tekstualnu datoteku sljedećeg formata.

```
200000 # pointsNumber
25 # limit
80 300 # eta1 eta2
60 0 # eta3 eta4
# slijede retci tablice funkcija
 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
```

Komentari se navode znakom ljestvi i treba ih zanemariti. Prva četiri retka uvijek će biti prikazanog formata i strukture. Ostatak datoteke u retcima koji nisu prazni navodi u svakom retku po 7 decimalnih brojeva koji su razdijeljeni s jednim ili više praznina ili znakova tabulatora.

Napravite program koji će kao argument komandne linije primiti ovakvu datoteku i koji će napraviti prikaz tako zadanog fraktala. Ispitajte rad programa barem na primjerima koji su dani u knjizi.

11.2.3 L-sustavi

U knjizi je u podpoglavlju 13.6 dan je detaljni primjer implementacija funkcija koja crtaju fraktale koji se dobivaju iz različitih L-sustava. U okviru ove vježbe napišite program koji omogućava crtanje fraktala koji su u knjizi prikazani na slikama 13.22a, 13.22b, 13.22c te 13.22d.

Vaš program treba pratiti pritiske na tipke tipkovnice. Pritiskom na tipke `1` do `4` program treba generirati i prikazati fraktale sa slika 13.22a do 13.22d.