Osnove virtualnih okruženja

Laboratorijske vježbe

Vježba 4
Osnovna programska sučelja: OpenGL

FER – ZTE – Igor S. Pandžić
Suradnja na pripremi vježbe:
Ognjen Dobrijević
Mario Weber

1. Uvod

OpenGL je programsko sučelje prema grafičkom hardware-u. Ovo sučelje sadrži oko 120 izravnih naredbi, koje se koriste za specifikaciju objekata i operacija potrebnih za kreiranje 3-dimnezionalnih interaktivnih aplikacija.

Ono daje pristup svim funkcijama grafičkog protočnog sustava, te omogućava iscrtavanje poligona, crta, točaka, korištenje osnovnog modela kamere, transformacija koordinatnog sustava, materijala, svjetla, modela osvjetljenja Gouraud ili plošnog sjenčanja, mapiranja teksture itd. OpenGL može raditi sa primitivnim oblicima, kao što su točka, crta, poligon itd.

OpenGL je napravljen kao hardverski neovisno sučelje predviđeno za korištenje na bilo kojoj platformi. U sklopu OpenGL-a ne postoje naredbe za rad sa prozorima ili korištenje ulaznih parametara zadanih od strane korisnika; za to se koriste specifične naredbe vezane uz određeni hardware koji se koristi. OpenGL ne pruža naredbe više razine za opis modela sastavljenih od 3-dimenzionalnih objekata.

OpenGL radi kao automat stanja. Prolazeći različitim stanjima, ona ostaju nepromijenjena sve dok ih sam korisnik ne promijeni.

Cilj vježbe je upoznavanje sa sintaksom OpenGL-a kroz ove upute i službenu dokumentaciju, a rješavanjem konkretnih zadataka dobit će se bolji uvid u iscrtavanje poligona, geometrijske transformacije i uvod u osnove animacije.

2. Alati potrebni za izvođenje vježbe

Za izvođenje vježbe potreban je programski jezik C++ (npr. MS Visual Studio ili MS Visual Studio .Net), te glut-3.7.6 biblioteka. U PCLAB-u instalirana je glut biblioteka samo za MS Visual Studio 6.0.

2.1 Izvođenje vježbe na vlastitom računalu

Biblioteka glut-3.7.6 dostupna je na stranicama predmeta. Datoteku je potrebno otpakirati tako da se njen sadržaj kopira u direktorij *X:\Program Files\Microsoft Visual Studio\VC98\Include\GL*, osim datoteke glut32.dll koja se pohranjuje u direktorij *X:\WINNT\system* ili *X:\WINDOWS\system32* (ovisno da li se radi pod Win 9x ili Win NT sustavima) i datoteke glut32.lib koja se pohranjuje u *...\VC98\Lib*. Ako se koristi MS Visual Studio .Net (2003) onda se sadržaj te biblioteke kopira u direktorij *X:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\include\GL.* Ako u direktoriju *...\Vc7\include* ne postoji direktorij *\GL*, potrebno ga je stvoriti. Datoteka glut32.lib u ovom slučaju se kopira u direktorij *...\Vc7\lib*.

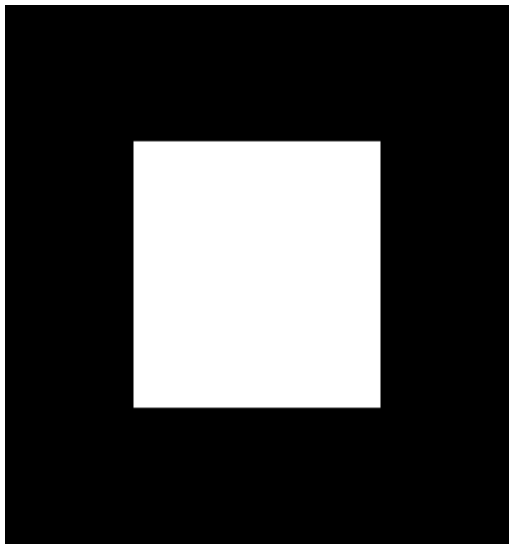
Detaljnije upute za instalaciju glut biblioteke možete naći na: http://www.cs.purdue.edu/homes/sun/Teach/530_04F/howTo.html.

3. Teorijska podloga

3.1 Jednostavan OpenGL program

OpenGL nije jezik sam za sebe, već dodatna kolekcija metoda i funkcija koje se koriste u izradi aplikacija u nekom od programskih jezika. U ovom poglavlju bit će dan teoretski uvod na temelju jednostavnog OpenGL programa.

Primjer 1 iscrtava bijeli kvadrat na crnoj podlozi, kao što je prikazano na Slici 1.



Slika 1: Bijeli kvadrat na crnoj podlozi

Primjer 1: Jednostavan OpenGL program

```
#include <sveŠtoJePotrebno.h>

main() {

   MolimOtvoriteProzor();

   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex2f(-0.5, -0.5);
      glVertex2f(-0.5, 0.5);
      glVertex2f(0.5, 0.5);
      glVertex2f(0.5, -0.5);
   glEnd();
   glFlush();

   ZadržiteProzorOtvorenimNaTrenutak();
}
```

Prva linija **main()** funkcije otvara prozor na ekranu: funkcija MolimOtvoriteProzor() zamjenjuje se za otvaranje prozora u sustavu na kojem se izrađuje aplikacija. Sljedeće dvije linije su OpenGL naredbe koje postavljaju boju pozadine (prozora) u crno: **glClearColor()** inicijalizira tu boju, a **glClear()** ju i stvarno postavlja. Nakon što se boja inicijalizira, boja

pozadine se postavlja svaki put nakon poziva funkcije **glClear()**. Ta boja se može promijeniti ponovnim pozivom funkcije **glClearColor()**. Analogno, naredba **glColor3f()** postavlja boju kojom se crtaju objekti – u našem slučaju boja je bijela. Svi objekti koji se crtaju nakon te naredbe koriste tu boju, sve dok se ona ne promijeni pozivom funkcije **glColor3f()**.

Sljedeća naredba koja se koristi, **glOrtho()**, specificira projekciju koju OpenGL koristi kod iscrtavanja konačne slike na ekran. Sljedeći poziv, ograničen sa **glBegin()** i **glEnd()**, definira objekt će biti iscrtan – u ovom primjeru to je poligon, sa četiri stranice. Njegovi vrhovi definirani su naredbama **glVertex2f()**. Po koordinatama je lako zaključiti da se radi o pravokutniku (kvadratu).

Naposljetku, **glFlush()** osigurava izvršavanje naredbi za crtanje. **ZadržiteProzorOtvorenimNaTrenutak**() funkcija održava prozor otvorenim kako se slika ne bi nestala odmah nakon njena iscrtavanja.

3.2 Sintaksa OpenGL naredbi

Kao što se može uočiti u prethodnom programu, OpenGL naredbe koriste prefiks **gl** i veliko slovo za svaku riječ naredbe (npr., naredba **glClearColor()**). Analogno tome, OpenGL definira konstante sa GL_ prefiksom, koriste samo velika slova, i podvlake za odvajanje riječi (npr., GL_COLOR_BUFFER_BIT).

Isto tako je moguće uočiti i slova dodana imenima nekih naredbi (npr., **3f** u **glColor3f()**). Ti dodaci se koriste kako bi se omogućila upotreba naredbi sa različitim tipovima argumenata. **3** označava broj argumenata naredbe; neke druge **Color** naredbe prihvaćaju i četiri argumenata. **f** dio sufiksa ukazuje da su argumenti tipa *floating-point*. Neke OpenGL naredbe prihvaćaju I do osam različitih tipova podataka kao argumente. Slova koja se koriste kao sufiksi dani su u Tablici 1, zajedno sa odgovarajućim definicijama tipova podataka.

Tablica 1: Sufiksi naredbi i definicije tipova podataka

Sufiksi	Tip podataka	Odgovarajući C-tip podataka	Definicija OpenGL tipa podataka
B	8-bit integer	signed char	Glbyte
S	16-bit integer	short	Glshort
I	32-bit integer	long	GLint, Glsizei
F	32-bit floating-point	float	GLfloat, Glclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned	unsigned short	GLushort

	integer		
ui	32-bit unsigned integer	unsigned long	GLuint, GLenum, GLbitfield

Stoga su sljedeće dvije naredbe

```
glVertex2i(1, 3);
glVertex2f(1.0, 3.0);
```

jednake, osim što prva specificira koordinate vrhova kao 32-bitne *integere* a druga ih specificira kao *floating-point* brojeve jednostruke preciznosti.

Neke OpenGL naredbe koriste slovo **v**, koje ukazuje da naredba prihvaća pokazivač na vektor (ili polje) vrijednosti. Mnoge naredbe imaju verzije koje prihvaćaju argumente i u obliku vektora i kao pojedinačne argumente, dok neke od njih prihvaćaju ili pojedinačne argumente ili skup vrijednosti u obliku vektora. Sljedeće linije pokazuju kako se koriste verzije koje prihvaćaju argumente i u obliku vektora i kao pojedinačne argumente a koji postavljaju trenutnu boju:

```
glColor3f(1.0, 0.0, 0.0);

float color_array[] = {1.0, 0.0, 0.0};
glColor3fv(color_array);
```

3.3 OpenGL animacija

OpenGL pruža jednostavno rješenje u obliku dvostrukog *spremnika* - hardware ili software koji omogućava dva različita *spremnika* za boje. Sadržaj jednog se prikazuje, dok se u drugom slika crta. Kad se završi sa crtanjem okvira, dolazi do zamjene *spremnika*. Korištenjem dvostrukog *spremnika*, svaki se okvir prikazuje tek kada je crtanje završeno; nikada se ne može iscrtati djelomično završen okvir.

Kako bi omogućili jednostavnu zamjenu gore navedenih *spremnika*, koristi se funkcija **swap_the_buffers()**, koja čeka dok se sadržaj jednog od *spremnika* potpuno ne iscrta na ekran. Naprimjer ako sustav osvježava ekran 60 puta u sekundi, to znači da je najveća brzina iscrtavanja okvira koja se može postići 60 okvira u sekundi, te ako se svi okviri mogu iscrtavati unutar 1/60 sekundi, animacija će se izvršavati bez ikakvih zastoja pri toj brzini.

Kod većine animacija, objekti u sceni se jednostavno iscrtavaju koristeći različite transformacije. Ako se značajna modifikacija provodi na strukturi svakog okvira, dostupna brzina okvira često pada. Ipak, treba imati na umu da se neiskorišteno vrijeme između poziva funkcije **swap_the_buffers()** može iskoristiti za računanje transformacija.

OpenGL ne sadrži naredbu **swap_the_buffers()** jer ona nije dostupna na svim sustavima. U zadacima, gdje je to potrebno, koristiti će se funkcija iz glut.h kolekcije:

```
void glutSwapBuffers();
```

3.4 Brisanje prozora

Memorija u koju je spremljena slika obično sadrži zadnje nacrtanu sliku, tako da je potrebno definirati novu boju pozadine prije iscrtavanja sljedeće scene.

Radi konzistentnosti opet navodimo naredbe za postavljanje boje prozora u crno:

```
glClearColor(0.0, 0.0, 0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
```

Boja u koju želimo obojati pozadinu prozora zadaje se kao kombinacija **crvene**, **zelene** i **plave (RedGreenBlue mod)**, i **alpha** vrijednosti koje se kreću u intervalu [0,1]. Podrazumjevana boja pozadine (0, 0, 0, 0) je crna.

void **glClear**(GLbitfield **mask**);

Postavlja vrijednost u spremniku boje na zadanu vijednost. Argument *mask* je kombinacija vrijednosti navedenih u Tablici 2.

Spremnik	Ime
Color buffer	GL_COLOR_BUFFER_BIT
Depth buffer	GL_DEPTH_BUFFER_BIT
Accumulation buffer	GL_ACCUM_BUFFER_BIT
Stencil buffer	GL_STENCIL_BUFFER_BIT

3.5 Definiranje boja

Opis objekta koji se crta je neovisna o specifikaciji boje. Uvijek kada se određeni objekt crta, crta se trenutno definiranim bojama. Nakon postavljanja boje objekti se crtaju istom bojom sve dok se ona ne promijeni.

Na primjer, pseudokod

```
postavi_trenutnu_boju(crvena);
crtaj_objekt(A);
crtaj_objekt(B);
postavi_trenutnu_boju(zelena);
postavi_trenutnu_boju(plava);
crtaj_objekt(C);
```

crta objekte A i B u crvenoj boji, a objekt C u plavoj boji. Naredba postavi_trenutnu_boju(zelena) nema učinka na sliku.

Nareba **glColor3f()** služi za postavljanje boje. Parametri su definirani, po navedenom redoslijedu, kao crvena, zelena, i plava komponenta boje. Naredba

```
glColor3f(1.0, 0.0, 0.0);
```

stvara najsvjetliju nijansu crevene boje bez zelene ili plave komponente. Sljedeće naredbe definiraju osnovne boje:

```
glColor3f(0.0, 0.0, 0.0);                    crna
glColor3f(1.0, 0.0, 0.0);                    crvena
glColor3f(0.0, 1.0, 0.0);                    zelena
glColor3f(1.0, 1.0, 0.0);                    žuta
glColor3f(0.0, 0.0, 1.0);                    plava
```

```
glColor3f(1.0, 1.0, 1.0);                    bijela
```

glClearColor() prima četiri parametra, ali nam četvrti nije bitan pa ga se može postaviti u 0.0.

3.6 Iscrtavanje slike

OpenGL posjeduje naredbu **glFlush()**, koja uzrokuje početak izvođenja naredbi, kako bi se iscrtavanje izvelo u konačnom vremenu.

void **glFlush**(void);

Slična naredba je **glFinish()**, koja ima istu funkciju kao i **glFlush()**, samo što ona čeka da se sve prethodne naredbe završe sa izvođenjem.

void **glFinish**(void);

3.7 Opis točaka, linija, i poligona

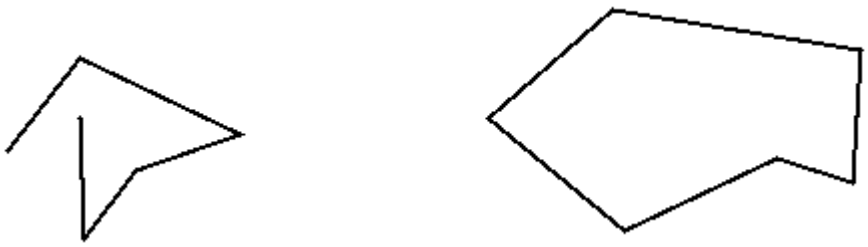
Svi geometrijski primitivi su opisani pomoću *vrhova* - koordinata koje definiraju same točke, krajeve linija, ili vrhove poligona.

3.7.1 Točka

Točka je predstavljena pomoću skupa *floating-point* brojeva koji se nazivaju *vertex* (vrhovi). Sve interne operacije računanja se rade sa 3-dimenzionalnim vrhovima. Vrhovi specificirani kao 2-dimenzionalni (npr., samo sa *x* i *y* koordinatama) dopunjuju se sa *z* koordinatom postavljenom u nula.

3.7.2 Linija

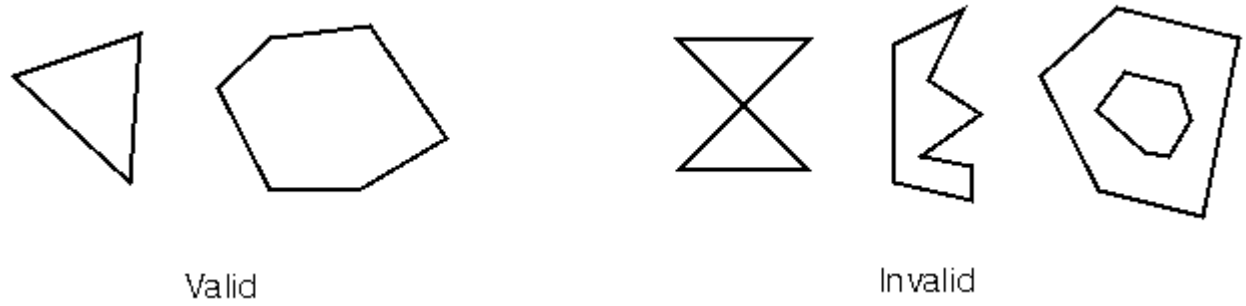
U OpenGL-u, linija znači *segment linije*, a ne beskonačni pravac.



Slika 2 : dva primjera segmenta linije

3.7.3 Poligoni

Poligoni su područja ograničena zatvorenim segmentima linija, a oni su definirani vrhovima svojih krajeva. S obzirom da poligoni mogu biti kompliciran OpenGL zahtjeva stroga ograničenja kod kreiranja poligona. Stranice poligona se ne smiju presjecati i poligon mora biti konveksan.



Slika3 : dozvoljeni i nedozvoljeni poligoni

3.7.4 Pravokutnici

Kako je pravokutnik vrlo čest u grafičkim aplikacijama OpenGL omogučuje crtanje pravokutnika. Mnoge verzije OpenGL-a imaju optimizirane naredbe za crtanje **glRect() :**

void **glRect**{sifd}(**TYPE***x1*, **TYPE***y1*, **TYPE***x2*, **TYPE***y2*);
void **glRect**{sifd}**v**(**TYPE***v1*, **TYPE***v2*);

One crtanju pravokutnik definiran točkama (x1, y1) i (x2, y2).

3.7.5 Krivulje

Bilo koja glatka zakrivljena krivulja ili površina se može aproksimirati, do bilo koje proizvoljne točnosti, sa vrlo malim segmentima linije ili malim područjem poligona.

Slika 4 : Aproksimiranje krivulje

3.8 Definiranje vrhova

Sa OpenGL-om su svi geometrijski objekti u konačnosti opisani kao uređen skup vrhova. Naredba **glVertex*()** nam služi za definiranje vrhova kod opisa geometrijskih objekata. Mogu se navoditi do 4 (x, y, z, w) koordinate za pojedini vrh uz koristenje odgovarajuće naredbe. Ako z ili w nisu definirani, z se podrazumjeva kao 0, a w kao 1. Pozivi naredbe **glVertex*()** bi trebali biti između **glBegin()** i **glEnd()** para.

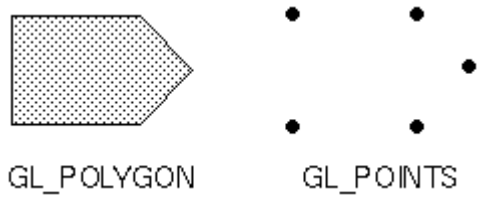
Primjeri:

```
glVertex2s(2, 3);
glVertex3d(0.0, 0.0, 3.1415926535898);
glVertex4f(2.3, 1.0, -2.2, 2.0);
```

3.9 Crtanje geometrijskih primitiva u OpenGL-u

Da bi nacrtali poligon u OpenGL-u trebamo mu definirati vrhove između poziva naredbi **glBegin()** i **glEnd()**. Argument naredbe **glBegin()** određuje kakav će se tip geometrijskog primitiva nacrtati iz vrhova.

```
glBegin(GL_POLYGON);
   glVertex2f(0.0, 0.0);
   glVertex2f(0.0, 3.0);
   glVertex2f(3.0, 3.0);
   glVertex2f(4.0, 1.5);
   glVertex2f(3.0, 0.0);
glEnd();
```



Da smo koristili GL_POINTS umjesto GL_POLYGON, dobili bi pet točaka kao primitiv. Tablica 3 nam daje 10 mogućih argumenata naredbe **glBegin().**

Table 3 : Imena i značenja geometrijskih primitiva

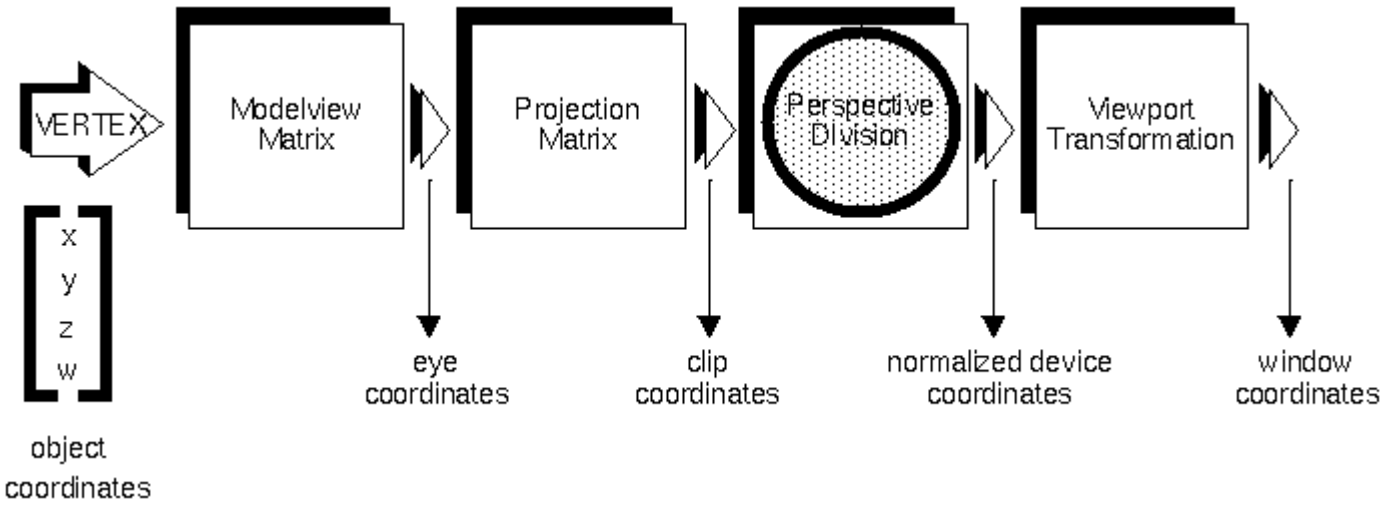
Vrijednost	Značenje
GL_POINTS	individual points
GL_LINES	pairs of vertices interpreted as individual line segments
GL_POLYGON	boundary of a simple, convex polygon
GL_TRIANGLES	triples of vertices interpreted as triangles
GL_QUADS	quadruples of vertices interpreted as four-sided polygons
GL_LINE_STRIP	series of connected line segments
GL_LINE_LOOP	same as above, with a segment added between last and first vertices
GL_TRIANGLE_STRIP	linked strip of triangles
GL_TRIANGLE_FAN	linked fan of triangles
GL_QUAD_STRIP	linked strip of quadrilaterals

3.10 Analogija kamere (transformacije)

Transformacijski proces kreiranja odgovarajuće scene je analogan slikanju fotoaparatom. Vrši se u sljedećim koracima:

- Postavljanje tronošca i usmjeravanje kamere prema sceni (transformacija pogleda).
- Postavljanje predmeta u sceni u odgovarajuću kompoziciju (transformacija modela).
- Izbor leće kamere i postavljanje zoom-a (projekcija).
- Određivanje veličine konačne slike (transformacija u prozor).

Nakon ovih koraka scena se može iscrtati.

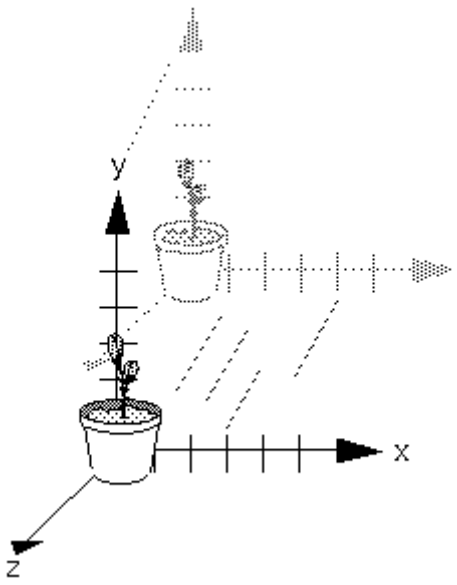


Slika 5: Transformacije od koordinata modela do prozora na ekranu

Transformacije pogleda i modela udružene kreiraju "modelview" matricu, koja je primjenjena za dobivanje koordinata očiju. Nakon toga, definiramo odrezujuće plohe, te uz pomoć projekcijske matrice dobivamo "clip koordinate", koje se još i normaliziraju. I konačno, primjenom "viewport" transformacije dobivamo ekranske ("window") koordinate.

3.10.1 Transformacija pogleda i modela

Ponovno napominjemo da transformacije pogleda i modela udružene kreiraju "modelview" matricu, koja je primjenjena za pretvaranje koordinata iz lokalnog koordinatnog sustava modela u koordinatni sustav kamere (eye coordinates), kao što je objašnjeno na predavanju o iscrtavanju. Ova udružena "modelview" transformacija može se shvatiti kao pomak modela koji se trenutno iscrtava ili kao pomak kamere. Jedno i drugo je ekvivalentno, kao što se vidi iz slike – npr. za pomak modela prema kameri ili pomak kamere prema modelu koristila bi se jedna te ista "modelview" matrica.



Dakle, "modelview" matricom mijenjamo koordinatni sustav u kojem trenutno crtamo, čime postižemo efekt pomicanja predmeta koji se trenutno crta, ili pomicanja kamere.

Modelview matricu možemo rotirati, translatirati i skalirati ili vršiti kombinaciju ove tri operacije. Naredbe vezane za rad sa matricama možete pogledati u poglavlju 3.10.4

3.10.2 Projekcija

Projekcija je poput biranja objektiva na kameri. Dakle, projekcija definira kako je objekt projiciran na ekran. Postoje dvije vrste projekcije:

Perspektivna projekcija – projekcija kao u realnom svijetu (npr. vidimo kako je željeznica sve uža u daljini). Koristi se za kreiranje realnih slika.

Ortogonalna projekcija – projecira objekte direktno na ekran, bez promjene veličine. Koristi se, npr., u arihitekturi za prikaz zgrada i unutrašnjosti iz različitih kutova gledanja

Naredbom **glFrustum()** definiraju se parametri projekcije.

Void **gluLookAt**(GLdouble *eyex*, GLdouble *eyey*, GLdouble *eyez*, GLdouble *centerx*, GLdouble *centery*, GLdouble *centerz*, GLdouble *upx*, GLdouble *upy*, GLdouble *upz*) – omogućava programeru pogled na scenu iz različitih proizvoljnih točaka gledišta. Željena točka gledišta definira se sa eyex, eyey i eyez koordinatama. Centerx, centery i centerz je bilo koja točka na liniji gledišta, dok nam upx, upy i upz koji je smjer prema gore.

3.10.3 Transformacija u prozor

Projekcija i transformacija u prozor definiraju kako će scena biti iscrtana na monitoru. Transformacija u prozor pokazuje oblik dostupnog dijela ekrana na koji će se scena iscrtati. Kako ova transformacija definira područje na ekranu, možemo ju zamisliti i kao definiranje veličine i lokacije krajnje fotografije (npr., treba li biti uvečana ili raširena).

Transformacija se postavlja naredbom **glViewport().** Kao argumente unosimo koordinate početnih točaka dostupnog ekrana, te širinu i visinu (npr., 0,0, 64, 80). Valja napomenuti da se širina i visina ekrana upisuju u pikselima!!!

3.10.4 Operacije s tekućom matricom

Sa **glMatrixMode()** naredbom biramo da li tekuća matrica definira projekciju ili modelview. Kao argument unosimo GL_PROJECTION ili GL_MODELVIEW, na primjer ako uzmemo naredbu glMatrixMode(GL_MODELVIEW) sve naknadne naredbe (npr. glLoadIdentity) za operacije na matrici izvršavat će se na modelview matrici.

Naredbom **glLoadIdentity()** vršimo inicijalizaciju, tj. tekuća matrica postaje jedinična matrica.

Nakon što je matrica inicijalizirana možemo je transformirati. Transformaciju radimo naredbom **glTranslatef()**. Naredbom **glRotatef()** vršimo rotaciju.

Promjenu veličine pozivamo naredbom **glScalef()**.

Dakle, ako smo naredbom glMatrixMode() zadali da je tekuća matrica modelview, ovim naredbama jednostavno vršimo transformacije trenutno crtanog objekta, odnosno kamere.

glLoadMatrix*() – eksplicitno definira s kojom se matricom popunjava tekuća matrica.

void **glMultMatrix**{fd}(const **TYPE** **m*) – množi tekuću matricu s matricom unsenom u argumentu.

3.10.5 Operacije inicijalizacije prozora i callBack funkcije

void glutInitDisplayMode(unsigned int mode) naredba deklarira prikazni model, tj. biramo da li ćemo korisiti jednostruki ili dvostruki spremnik te RGB boju.

void glutInitWindowSize(int width, int height) deklaracija inicijalne veličine prozora.

void glutInitWindowPosition(int x, int y) postavljanje početnog položaja prozora.

Naredbom **int glutCreateWindow(char *name)** otvara se prozor sa naslovom.

void glutDisplayFunc(void (*func)(void)) naredba postavlja callback funkciju za iscrtavanje slike na ekran. Kada god je potrebno iscrtati novu sliku, poziva se callback funkcija koja vrši iscrtavanje slike. Callback funkciju mora definirati sam programer.

void glutReshapeFunc(void (*func)(int width, int height)) postavlja callback funkciju za podešavanje novih parametara. Callback funkcija se poziva kod bilo kakve promjene veličine prozora ili premještanja prozora s jedne na drugu poziciju.

4. Opis zadatka

1. ZADATAK

Potrebno je proučiti priloženi OpenGL program, kompajlirati ga i pokrenuti njegovo izvršavanje.

2. ZADATAK

Potrebno je proučiti priloženi OpenGL program, pokrenuti njegovo izvršavanje i odgovoriti na postavljena pitanja:

a) Što se dobiva pokretanjem ovog programa?

b) Na koji se način iscrtava dobiveni objekt?

c) Na koji se način postiže dojam 3-dimenzionalnosti pri iscrtavanju objekta?

e) Objasniti čemu služi varijabla ***step*** unutar *drawsphere(float R, float **step**)* metode?

3. ZADATAK

Potrebno je napisati program koji će simulirati dio Sunčevog sustava, odnosno rotaciju Zemlje oko Sunca i oko svoje osi, te rotaciju Mjeseca oko Zemlje i oko svoje osi. Putanje rotirajućih tijela neka budu kružne.

5. Upute za rad

1. ZADATAKv

Na početku ovih uputa dan je (pseudokod) jednostavnog OpenGL program. Uz njega su objašnjene neke od funkcija koje se općenito koriste za izvođenje većine OpenGL programa. U dodatnim materijalima možete pronaći izvorni kod koji u cjelosti implementira taj pseudokod, odnosno jednostavan "Hello World!" program u OpenGL-u.

NAPOMENA: ukoliko se koristi MS Visual Studio obavezno kod pokretanja programa za projekt izabrati Win32 Console Application, te program spremiti kao ime.c, a ne ime.cpp!

2. ZADATAK

Program je priložen u dodatnim materijalima. Potrebno ga je dobro proučiti i odgovoriti na postavljena pitanja. Ovaj zadatak će biti koristan prilikom rješavanja 3. zadatka.

3. ZADATAK

U dodatnim materijalima dan je kôd koji djelomično implementira Sunčev sustav. Funkcija *reshape()* definira parametre (globalnog) koordinatnog sustava u kojem će se postavljati objekti. Funkcija *display()* služi za opis i crtanje samih objekata. Funkcija *mouse()* na lijevi klik miša pokreće animaciju, a na desni klik miša ju zaustavlja. Funkcija *spinDisplay()* mijenja parametre čijom promijenom se izvodi sama animacija.

Osnovne funkcije:

- Funkcija *void init (void)* – inicijalizacija osnovnih parametra
- Funkcija *int main(int argc, char** argv)* – deklarira veličinu i položaj prozora te prikazni mod. Obrađuje događaje u glavnoj petlji.
- Funkcija *void mouse()* – pokreće i završava animaciju.

Funkcije koje treba nadopuniti:

- Funkcija *drawSphere(float R, float step)* – iscrtavanje objekata. Koristi gotovu funkciju iz drugog zadatka.
- Funkcija *SpinDisplay()* – izračunava vrijednost pomaka pri gibanju Zemlje i Mjeseca iz vremena. Potrebno je izračunati parametre transformacijskih matrica koje se koriste za animaciju.
- Funkcija *display()* – implementacija animacije sunčevog sustava. Tu je potrebno dodati matrice s kojima će se ostvariti animacija.

Detaljan opis funkcija i potrebnih parametara nalazi se u samom kôdu.

6. Pomoćni materijali za izradu vježbe

Pomoćni materijali za izradu vježbe nalaze se u datoteci **OpenGL.zip** koja se može naći na web stranicama predmeta. Datoteka sadrži kod prvog, drugog i dijela trećeg zadatka.

Na web stranicama predmeta također možete naći **OpenGL Programming Guide (The Red Book)**.

7. Predavanje rezultata vježbe

Rezultati vježbe se predaju zapakirani u arhivu ***OVO-V4- Rezultati-<ImePrezime>.zip*** koja treba sadržavati:

- Izvještaj o izvođenju vježbe s odgovorima na pitanja iz drugog zadatka, te opisom postupka rješavanja 3. zadatka.
- Rješenje 3. zadatka kao MS visual C++ projekt spreman za kompiliranje, s dobro komentiranim izvornim kodom.
- Izvršnu datoteku 3. zadatka.

Navedena arhiva treba biti predana korištenjem *Web aplikacije za predaju vježbi* dostupne preko web stranica predmeta.

Napomena: Rezultati se šalju isključivo preko gore navedene aplikacije. U slučaju problema, javiti se email-om na adresu ovo@tel.fer.hr. Sačuvajte kopiju poslanih rezultata.

8. Napredni zadaci (ovaj dio nije obavezan)

- Dodati klizač za mijenjanje brzine simulacije u 3. zadatku.
- Dodati mogučnost biranja pogleda tako da se automatski gleda sa mjeseca prema zemlji, sa zemlje prema suncu, i sl.
- Dodati model svemirske letjelice kojom korisnik može upravljati preko miša i tako putovati simuliranim sunčevim sustavom, uz mogučnost pogleda iz cockpit-a ili iz fiksne točke iza letjelice.