Računalniška grafika

Leon Kos

10. februar 2009

Povzetek

Predstavljeni so bistveni prijemi pri programiranju računalniške grafike z grafično knjižnico OpenGL. Ta dokument naj bi bil osnova za vaje pri predmetih RPK, OPK, PK in izdelavo seminarjev s tega področja. Ker študenti FS pridobijo znanje programiranja v jeziku Fortran, so primeri podani za ta jezik. To pa ne pomeni, da je teorija omejena le na ta jezik, saj brez bistvenih popravkov kode lahko pišemo tudi v jeziku C in C++.

1 Uvod

Za modeliranje posebnih modelov običajno ni možno uporabiti splošno namenskih grafičnih orodij. To se pokaže predvsem pri vizualizaciji inženirskih preračunov. Rezultati modeliranja običajno niso le funkcije ampak kompleksni objekti kot so grafi, vodi, hierarhične strukture, animacije gibanja, mehanizmi, kontrola poti, volumski modeli posebnih oblik, ...

Skozi razvoj računalnikov so se uvajali različni standardi za grafiko. Od začetnikov kot je GKS in njegovega naslednika PHIGS je ostal le še spomin. To pa predvsem zaradi zahtevnosti implementacije in zaprtosti kode. Kot edini „odprti“ standard obstaja le OpenGL, ki ga je najprej uvedel SGI na svojih grafično podprtih delovnih postajah. Poleg OpenGL obstaja tudi Microsoftov Direct3D, ki pa je omejen na PC računalnike z Windows in ni tako enostaven za uporabo kot OpenGL, ki se je zaradi svoje odprtosti in zmogljivosti uveljavil na vseh operacijskih sistemih in strojnih platformah.

Jezik OpenGL je konstruiran kot strojno-neodvisen vmesnik med programsko kodo in grafičnim pospeševalnikom. Strojna neodvisnost jezika OpenGL pomeni tudi to, da v specifikaciji jezika ni podpore za nadzor okenskega sistema in dogodkov (*events*) pri interaktivnem programiranju. Za tak nadzor so za vsak operacijski sistem izdelani vmesniki, ki povezujejo OpenGL stroj z okenskim sistemom.

Zaradi specifičnosti različnih okenskih sistemov (Windows, Xwindow, MacOS, BeOS) je potrebno za vsak sistem uporabiti posebne prijeme pri klicanje OpenGL ukazov. Da bi vseeno lahko posali programe, s sicer omejeno funkcionalnostjo uporabniškega vmesnika, se je izdelala knjižnica GLUT (GL UTility), ki vse razlike med operacijskimi sistemi kompenzira in vpeljuje skupen način manipuliranja z dogodki (*events*). S knjižnico GLUT je tako mogoče pisati prenosljive programe, ki so enostavni za programiranje in dovolj zmogljivi za nezahtevne uporabniške vmesnike.

2 Enostavni OpenGL program

Osnovni jezik OpenGL je podan v knjižnici GL. Bolj zahtevni primitivi se gradijo z knjižnico GLU (GL Utility) v kateri so podprogrami, ki uporabljajo rutine GL. Rutine GLU vsebujejo več GL ukazov, ki pa so splošno uporabni in so bili zato standardizirani.

2.1 Dogodki

Vsi okenski vmesniki delujejo na principu dogodkov (*events*). To so signali okenskega sistema, ki se pošiljajo programu. Naš program je tako v celoti odgovoren za vsebino okna. Okenski sistem mu le dodeli področje (okno) katero vsebino mora popolnoma nadzorovati. Poleg dodeljenega področja pa okenski sistem pošilja še sporočila našemu programu. Najbolj pogosta sporočila so:

display Prosim obnovi (nariši) vsebino okna. Več možnih primerov je, da se to zgodi. Lahko je drugo okno odkrilo del našega okna, okno se je premaknilo na omizju ali pa se je ponovno prikazalo po tem ko je bilo ikonizirano. Prestrezanje tega dogodek je obvezno saj mora prav vsak program poskrbeti, da se vsebina okna obnovi.

reshape Velikost/oblika okna se je spremenila. Poračunaj vsebino okna za novo velikost. Ta dogodek se zgodi, kadar uporabnik z miško spremeni velikost okna.

keyboard Pritisnjena je bila tipka na tipkovnici.

mouse Stanje gumbov na miški se je spremenilo. Uporabnik je pritisnil ali sprostil enega od gumbov.

motion Uporabnik premika miško ob pritisnjenem gumbu.

timer Program zahteva sporočilo po preteku po določenega časa, da bo popravil vsebino okna. Primerno je za časovne simulacije.

Seveda poleg naštetih dogodkov obstajajo še drugi dogodki, za katere lahko skrbi naš program. Ni pa potrebno da naš program skrbi za vse naštete dogodke. Običajno mora program povedati okenskem sistemu, za katere dogodke bo skrbel in za te dogodke mu bo sistem tudi pošiljal sporočila.

2.2 GLUT

Za abstrakcijo dogodkov okenskega sistema v našem primeru skrbi knjižnica GLUT. Primer minimalnega programa, ki nariše črto, je naslednji:

```
subroutine display
include 'GL/fgl.h'
implicit none
call fglClear(GL_COLOR_BUFFER_BIT)
call fglColor3f(1.0, 0.4, 1.0)
call fglBegin(GL_LINES)
call fglVertex2f(0.1,0.1)
call fglVertex3f(0.8,0.8,1.0)
call fglEnd
call fglFlush
end

program crta
external display
include 'GL/fglut.h'
call fglutInit
call fglutInitDisplayMode(GLUT_SINGLE + GLUT_RGB)
call fglutCreateWindow('Fortran GLUT program')
call fglutDisplayFunc(display)
call fglutMainLoop
end
```

Fortranski program je sestavljen iz dveh delov: glavnega programa `crta` in podprograma display. Z ukazom `fglutInit` inicializiramo GLUT knjižnico podprogramov. Sledi zahteva po vrsti okna. S konstantama GLUT_SINGLE in GLUT_RGB povemo, da želimo okno z eno ravnino tribarvnega RGB prostora. spremenljivka `window` hrani številko okna, ki jo naredi `fglutCreateWindow` in hkrati pove OS, kakšen naj bo napis na oknu. Okenskemu sistemu moramo še dopovedati, katere dogodke bo program prestrezal. Za podani primer je to le prikaz vsebine okna. S klicem podprograma `fglutDisplayFunc` prijavimo OS, da naj pošilja sporočila za izris, knjižnici GLUT pa s tem dopovemo, da ob zahtevi za ponovni izris pokliče podprogram `display`.

Zadnji klic v glavnem programu je vedno `fglutMainLoop`. Ta podprogram se konča le takrat se konča celoten program. Kot že ime podprograma govori, je to glavna zanka programa, v kateri GLUT sistematično sprejema sporočila in kliče naše podprograme za dogodke. V podanem primeru je to le en tip dogodkov, ki je tudi najbolj uporaben - `display`. Ostale tipe dogodkov pa `fglutMainLoop` obdeluje na privzeti način, ki je lahko preprosto ignoriranje sporočila ali pa klicanje vgrajenega podprograma, ki obdela sporočilo. Izgled glavnega programa je tako običajno zelo podoben za vse tipe GLUT programov v katerem si sledijo ukazi v naslednjem zaporedju:

1. Vključi definicije konstant `GLUT` z ukazom `include 'GL/glut.h'`
2. Dopovej Fortranu, da so imena, kot je npr. `display`, podprogrami in ne spremenljivke. To se naredi z ukazom `external`
3. Inicializiraj GLUT
4. Nastavi parametre okna (položaj, velikost, tip, bitne ravnine, pomnilnik)
5. Naredi okno in ga poimenuj
6. Prijavi podprograme, ki jih bo program izvajal ob dogodkih. Obvezno prestrezanje je le za *display*. Ostali so poljubni.
7. Nastavi lastnosti OpenGL stroja. To so običajno ukazi `fglEnable` ali pa kakšna nastavitev luči, materialov in obnašanja GL stroja. V tem področju se običajno nastavi tudi ostale spremenljivke, ki niso neposredno vezane na OpenGL, ampak na samo delovanje programa, ki poleg prikaza dela še kaj drugega.
8. Zadnji je klic `fglutMainLoop`, iz katerega se program vrne, ko zapremo okno. Ob tem glavni program konča.

2.3 Izris v jeziku OpenGL

V podprogramu `display` se morajo torej nahajati ukazi, ki nekaj narišejo v okno. To so ukazi v jeziku OpenGL ali krajše v jeziku GL. Vsi podprogrami ali funkcije v GL imajo predpono pri imenu *gl* oziroma za Fortran *fgl*. Te predpone so potrebne zaradi možnega prekrivanja z imeni v drugih knjižnicah in jezikih. Za razumevanje jezika se lahko funkcije razlaga brez teh predpon, saj je OpenGL koncipiran tako, da so tipi argumentov za vse jezike podobni. Za posamezen jezik se predpostavi prefiks (za Fortran je to `fgl`, za C pa `gl`)

Podprogram `display` torej skrbi za izris vsebine okna. Z ukazom *Clear* se briše celotno področje okna. Kaj konkretno se briše povemo z argumentom. V našem primeru je to `GL_COLOR_BUFFER_BIT`, kar pomeni brisanje vseh točk v barvnem pomnilniku.

Z ukazom *Color* se nastavi trenutna barva grafičnih gradnikov, ki se bodo izrisovali v nadaljnih ukazih. Kot argument se podaja barva v RGB komponentah. Običajno imajo GL ukazi na koncu imena tudi oznako tipa argumentov, ki jih je potrebno podati pri klicu podprograma. To je običaj za skoraj vse jezike. Tako imamo za Fortran in za C pri podprogramih, kot končnico imena še oznako o številu argumentov in tip argumentov. Podprogram *fglColor3f* torej pomeni, da zahteva podprogram tri argumente tipa `float`, kar je ekvivalentno v fortranu tipu *real* ali `real*4`. Najbolj uporabljane GL ukaze lahko torej podamo z različnimi tipi argumentov za isti ukaz. Izbor tipa argumentov je odvisen od programerja in njegovih zahtev. Tako so argumenti za isto funkcijo izbrani glede na priročnost. Za podani primer imamo ukaz *Vertex* v dveh oblikah in isti tip argumentov. `Vertex2f` pomeni, da podajamo kordinate vozlišča z dvema argumentoma. Tipi argumentov (končnic) so naslednji:

f V jeziku C float in real ali real*4 za Fortran

d double za C in real*8 za Fortran

i integer

s short integer v C-ju ali integer*2 za F

Namesto fiksiranega števila argumentov obstajajo tudi funkcije, ki imajo podan argument v obliki vektorja. Za to se uporabi končnica `v`. Nekaj primerov končnic:

3f Sledijo trije argumenti realnih števil

3i Sledijo trije argumenti celih števil

3fv Sledi vektor treh realnih števil

Ker je GL v osnovi 3D pomeni, da so argumenti če sta podani le dve koordinati x in y v ravnini $z = 0$. Torej je to ekvivalentno klicu podprograma *Vertex(x, y, 0)*. V našem primeru imamo tudi nastavitev vozlišča z ukazom `Vertex3f(0.8,0.8,1.0)`, ki podaja vse tri koordinate v prostoru. Koordinata $z = 1$ je torej podana, vendar je zaradi privzetega začetnega ortografskega pogleda v prostor ravnine (x,y) koordinata z po globini neopazna. če pa bi bila projekcija perspekvivna in ne ortogonalna, bi opazili tudi vpliv koordinate z. Raznolikost tipov argumentov se pokaže ravno pri podajanju vozlišč, saj obstajajo naslednjji podprogrami: *glVertex2d, glVertex2f, glVertex2i, glVertex2s, glVertex3d, glVertex3f, glVertex3i, glVertex3s, glVertex4d, glVertex4f, glVertex4i, glVertex4s, glVertex2dv, glVertex2fv, glVertex2iv, glVertex2sv, glVertex3dv, glVertex3fv, glVertex3iv, glVertex3sv, glVertex4dv, glVertex4fv, glVertex4iv, glVertex4sv*. In vse to za en sam ukaz.

Namen velikega števila istih podprogramov za isto funkcijo je opustitev pretvarjanja tipov in s tem pisanje bolj razumljive in hitrejše kode. V jeziku C++ ali Java, ki pa sam dodaja ustrezne argumente k imenom funkcij, pa bi lahko obstajal le en podprogram (npr. *glVertex*), jezik pa bi sam dodal ustrezne končnice in s tem klical ustrezno funkcijo.

Izris grafičnih elementov risbe se v GL podaja med ukazoma *glBegin* in *glEnd*. Predmet risanja podamo kot argument v ukazu `Begin`. Na splošno velja, da funkcije brez končnic zahtevajo en sam argument s konstanto, ki je podana v *header* datoteki s končnico `.h` in se vključuje za začetek programske enote s stavkom `include 'GL/fgl.h'`. Za fortran je programska enota vsak podprogram, zato moramo pri vsakem podprogramu, ki uporablja te konstante, na začetku dopisati še vključevanje teh konstant. Za C je modul *.c* datoteka, in ni potrebno vključevanje definicij konstant za vsak podprogram, kot je to nujno za Fortran.

Vse te GL konstante, ki so napisane v *fgl.h* in *fglu.h* imajo standardno predpono `GL_` in je za vse jezike enoznačen. Ker pa Fortran ne zahteva deklaracje spremenljivk in ima implicitno definirane tipe je prav možno, da se zatipkamo pri imenu konstante, kar za Fortran pomeni realno številko z vrednostjo 0.0. Da se izognemo takim težavam, se priporoča ukaz *implicit none*, s katerim izključimo predpostavljene tipe in moramo za vsako spremenljivko povedati, kakšnega tipa je. žal pa F77 ne omogoča prototipov tako, da je še vedno po-

trebna pazljivost, kakšne tipe podajamo kot argumente podprogramom. Posebno to velja za podprograme *GLU*, ki običajno nimajo tako razvejanih možnosti argumentov, kot knjižnica *GL*.

Zadnji ukaz `glFlush` dopove GL stroju naj vse te ukaze, ki jih je sprejel do sedaj, spravi iz svojih internih pomnilnikov v okno okenskega sistema. Ker imamo v našem primeru le enostaven izris, smo se odločili le za en slikovni pomnilnik (`GLUT_SINGLE`), ki je primeren le za statične slike. Za aplikacije pri katerih se vsebina zaslona pogosto spreminja, je primerneje uporabiti okno z dvema grafičnima pomnilnikoma `GLUT_DOUBLE`. Prednost slednjega je v tem, da v en pomnilnik rišemo, drugega pa prikazujemo. Rišemo v ravnino, ki je v ozadju. Ob koncu risanja pa le zamenjamo ravnini. Ker pa je to odvisno od sistema se ukaz za zamenjavo risalnih ravnin imenuje `glutSwapBuffers`. Prednost takega načina se pokaže pri animacijah.

3 Geometrijski primitivi

Uporabiti je mogoče le enostavne primitive. To pa predvsem zaradi zahtevane hitrosti in enostavosti izdelave strojnega pospeševanja. Ločimo tri vrste teh enostavnih primitivov:

- točke ali pike
- črte
- ploskvice konveksnega tipa

Bolj zahtevne predstavitve izvedemo z kombiniranjem teh primitivov. Tako krivulje različnih tipov aproksimiramo z lomljenkami, površine pa s ploskvicami. Za najbolj razširjene kompleksne tipe se že nahajajo podprogrami v knjižnici *GLU*. Obstajajo tudi možnosti sestavljenih enostavnih gradnikov za črte in ploskvice. Za črte poznamo tako naslednje možnosti:

GL_LINES Pari vozlišč podajajo posamezne segmente

GL_LINE_STRIP Zaporedje povezanih vozlišč podaja lomljenko

GL_LINE_LOOP Lomljenka se zaključi tako, da poveže prvo in zadnje vozlišče.

Konstante so podane kot argument za podprogram *Begin*. Za ploskve se podaja več točk. Najenostavnejše ploskve so trikotniki. Možni so še ravninski štirikotniki in konveksni ravninski mnogokotniki. Enostavne elemente lahko podajamo tudi v pasovih in trikotnike v pahljačah:

GL_TRIANGLES Tri vozlišča za en trikotnik

GL_TRIANGLE_STRIP Pas trikotnikov. Tri vozlišča za prvi in nato vsako nadaljnje vozlišče k prejšnjemo trikotniku doda nov trikotnik.

GL_TRIANGLE_FAN Pahljača: vsako dodatno vozlišče naredi dodaten trikotnik v smislu dežnika.

GL_QUADS Ravninski štirikotnik se podaja s štirimi vozlišči.

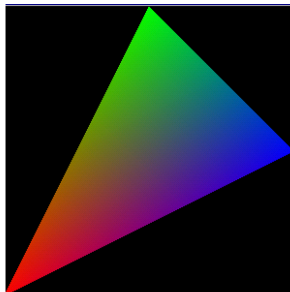
GL_QUAD_STRIP Dodatni štirikotniki gradijo pas z dodajanjem parov vozlišč.

GL_POLYGON En sam konveksni mogokotnik poljubnega števila vozlišč.

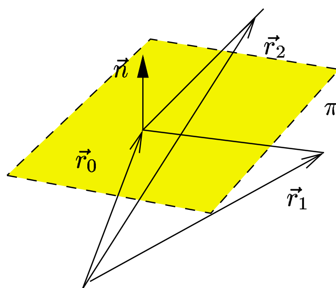
Za nesestavljeni tipe gradnikov lahko med *Begin* in *End* podamo tudi več vozlišč. Tip gradnika se pri pri tem avtomatsko ponovi. Med *Begin* in *End* se lahko uporabljajo še ukazi za barvo *glColor* in normale *glNormal*. Ti ukazi nastavljajo trenutno stanje, ki velja za vsa naslednja vozlišča. Primer podprograma za prikaz enega trikotnika v ravnini je naslednji:

```
subroutine display
implicit none
include 'GL/fgl.h'
call fglClear(GL_COLOR_BUFFER_BIT)
call fglBegin(GL_TRIANGLES)
call fglColor3f(1.0, 0.0, 0.0)
call fglVertex2f(-1.0, -1.0)
call fglColor3f(0.0, 1.0, 0.0)
call fglVertex2f(0.0, 1.0)
call fglColor3f(0.0, 0.0, 1.0)
call fglVertex2f(1.0, 0.0)
call fglEnd
call fglFlush
end
```

Pred vsako točko je podana še trenutna barva. Izrisani trikotnik tako ni enotne barve ampak se njegova notranjost preliva iz ene skrajne barve v drugo. Rezultat prikazuje slika 1. Uporaba različnih barv v vozliščih mogoče ni posebno uporabna. Za podajanje normal pa je običajno potrebno, da so normale v vozliščih različne. Različne normale v vozliščih nastajajo povsod tam kjer imajo ploskvice skupen rob, za katerega želimo, da ima gladek prehod. To pa je povsod tam, kjer aproksimiramo „gladko“ površino z osnovnimi gradniki. Slika 2 kaže splošno postavitev treh točk v prostoru. Normalo za trikotnik z vozlišči $\vec{r}_0, \vec{r}_1, \vec{r}_2$



Slika 1: Trikotnik s podanimi barvami v vozliščih



Slika 2: Normala

izračunamo z vektorskim produktom

$$\vec{n} = \frac{(\vec{r}_1 - \vec{r}_0) \times (\vec{r}_2 - \vec{r}_0)}{|(\vec{r}_1 - \vec{r}_0) \times (\vec{r}_2 - \vec{r}_0)|}$$

Imenovalec zgornje enačbe je dolžina vektorja $\vec{n}$. Normala je pravokotna na razliko vektorjev, ki podajajo vozlišče gradnika.

4 Geometrijske transformacije

Osnova vseh grafičnih knjižnic so tudi osnovne geometrijske transformacije, kot so:

Translate(x, y, z) Premik v smeri vektorja

Rotate(fi, x, y, z) Rotacija za *fi* stopinj okoli osi podane z (x, y, z)

Scale(x, y, z) Skaliranje po posameznih oseh

Ukazi za transformacije se ne smejo pojavljati med *Begin/End*, saj bi to pomenilo, da se transformacija spreminja med izrisom. Geometrijske transformacije nam pomagajo pri modeliranju, saj lahko podajamo vozlišča gradnikov v nekem poljubnem koordinatnem sistemu. To je lahko svetovni koordinatni sistem ali lokalni koordinatni sistem. Za primer izberimo izris krivulje $y(x) = \sin(x)$ v jeziku GL. Kot smo že opazili, je prednastavljeno okno v GLUT obliki kvadrata, velikosti (-1,-1) do (1,1). Vsega skupaj torej dve enoti. V zaslonskih koordinatah je prednastavljena velikost okna 300×300 pikslov. Za nas je pomembno, da sinus narišemo v mejah od -1 do 1. Vzemimo primer, ko predvidimo število točk. Podprogram za izris je naslednji:

```
subroutine display
include 'GL/fgl.h'
call fglClear(GL_COLOR_BUFFER_BIT)
call fglBegin(GL_LINE_STRIP)
do i=0,10
  y = sin((i-5)/5.0*3.14)
  call fglVertex2f((i-5)/5.0, y/3.14)
end do
call fglEnd
call fglFlush
end
```

Da smo spravili naših 11 točk lomljenke v okvir -1, 1 je bilo potrebno premakniti koordinatni sistem osi x za 5, ga nato še skalirati tako, da smo iz območja [0,10] dobili območje [-3.14, 3.14]. čeprav smo za izračun koordinate y potrebovali na osi x območje [-3.14, 3.14] pa je potrebna os x za izris v območju [-1,1]. Zato pri izrisu podajamo os x tako, da ponovno poračunavamo območje [0,10] v območje [-1,1], tako da `i`-ju odštejemo 5 in delimo z 5. Lahko bi tudi delili s 5 in odšteli 1. Nekoliko bi poenostavili stvari, če bi imeli vsaj en kordinatni sistem že takoj uporaben. Recimo os x. Zanka se nekoliko poenostavi, še vedno pa je potrebno vse koordinate pomanjšati za 3.14 oziroma poskalirati.

```
do x=-3.14, 3.14, 0.6
y = sin(x)
call fglVertex2f(x/3.14, y/3.14)
end do
```

Bolj razumljivo bi bilo risati kar v lokalnem koordinatnem sistemu in prednastaviti pomanjšavo modela. Za pomanjšavo uporabimo ukaz za skaliranje, ki posamezne koordinate množi s konstanto 1/3.14, preden se izriše. Podprogram za izris je naslednji:

```
subroutine display
include 'GL/fgl.h'
call fglClear(GL_COLOR_BUFFER_BIT)
call fglScalef(1/3.14, 1/3.14, 1.0)
call fglBegin(GL_LINE_STRIP)
do x=-3.14, 3.14, 0.6
y = sin(x)
call fglVertex2f(x, y)
end do
call fglEnd
call fglFlush
end
```

Prednost takega načina razmišljanja se pokaže, že ko želimo pod sinusom narisati še krivuljo kosinusa. Seveda ni možno obeh krivulj risati z *GL_LINE_STRIP* v isti zanki. Zato se odločimo za ponovno risanje v lokalnem koordinatnem sistemu in prednastavimo pomik navzdol za 1.5 enote.

```
subroutine display
include 'GL/fgl.h'
call fglClear(GL_COLOR_BUFFER_BIT)
call fglScalef(1/3.14, 1/3.14, 1.0)
call fglBegin(GL_LINE_STRIP)
do x=-3.14, 3.14, 0.6
y = sin(x)
call fglVertex2f(x, y)
end do
call fglEnd
call fglTranslatef(0.0, -1.5, 0.0)
call fglBegin(GL_LINE_STRIP)
do x=-3.14, 3.14, 0.6
y = cos(x)
call fglVertex2f(x, y)
end do
call fglEnd
call fglFlush
end
```

Podani program za kosinus ne nastavlja ponovno skaliranja, saj je ukaz že pred tem nastavil pomanjšavo. Translacija za -1.5 se izvede v koordinatnem sistemu kosinusa. Splošen napotek za razumevanje transformacije vsakega vozlišča je, da se za podano koordinato upoštevajo transformacije, kot so napisane od spodaj na vzgor. Transformacija, ki se izvede zadnja je torej napisana na prvem mestu v programu. Tak način transformiranja točk nam omogoča enostavnejše modeliranje. Koordinata y kosinusa se izračuna tako, da se pred izrisom najprej vsaki točki y prišteje translacija -1.5 in potem se še izvede skaliranje tako, da se ta vmesna točka pomnoži še z 1/3.14.

4.1 Nadzor transformacijske matrike

OpenGL pa za izračun koordinat ne hrani vse zgodovine posameznih transformacij za nazaj, saj bi bilo to računsko potratno. Vse te transformacije, ki jih v poljubnem zaporedju navajamo v programu, popravljajo transformacijsko matriko. OpenGL ima le dve aktivni transformacijski matriki, ki jih uporablja za poračun koordinat. Prva matrika je modelna, druga pa je projekcijska. Mi bomo uporabljali le modelno transformacijo in upoštevali, da projekcijska matrika omogoča prikaz ravnine (x, y) v področju [-1,1]. Modelna matrika je tudi stalno aktivna, če se ne izbere projekcijsko.

Modelna matrika se ob vsakem klicu transformacijskega podprograma popravi. Začetna oblika modelne matrike je enotska. Vsak klic podprograma *Translate*, *Scale* in *Rotate* pa matriko popravi tako, da so upoštevane vse prejšnje transformacije in nad njimi še novo podana transformacija. Matrika je torej stalna, kar se kaže tudi v napaki prejšnjega programa za izris sinusa in kosinusa, ki je pri vsakem ponovnem izrisu trikrat manjši. To lahko preverimo tako, da okno prekrijemo s kakim drugim oknom in ga potem ponovno odkrijemo. Kot že omenjeno, je na začetku programa matrika enotska. S podprogramom *fglLoadIdentity* na začetku bi lahko to tudi zagotovili ob vsakem izrisu.

Za bolj zahtevne transformacije je potrebno matriko začasno shraniti in obnoviti. OpenGL ima v ta namen poseben pomnilnik v obliki sklada, v katerega lahko shranjujemo trenutno transformacijsko matriko. V pomnilniku oblike LIFO (*Last In, First Out*) je prostora je za najmanj 32 matrik. Pomnilnik si lahko predstavljamo kot hladilnik, v katerega shranjujemo matrike. Matriko, ki jo želimo shraniti, potisnemo na začetek in to tako, da vse ostale matrike potisnemo malo naprej na polici. Ko nekaj želimo iz hladilnika, je to lahko le zadnja matrika. če želimo predzadnjo, moramo poprej vzeti zadnjo. Za shranitev trenutne matrike se uporabi **glPushMatrix**, za ponastavitev iz sklada pa uporabimo **glPopMatrix**. Izkaže se, da je za modeliranje taka oblika pomnilnika povsem primerna.

Za primer vzemimo primer kocke sestavljene iz šestih ploskev. Za izris kvadrata obstaja že krajša funkcija `glRectf(x1, y1, x2, y2)` za ravnino $z = 0$. če želimo imeti kvadrat v poljubni ravnini, pa uporabimo transformacije.

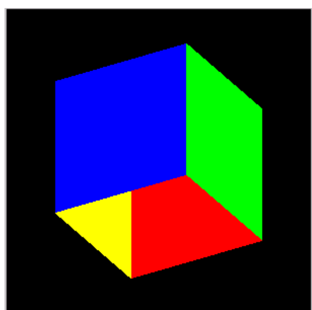
```
subroutine kvadrat(i)
real r(6), g(6), b(6)
data r /1,0,0,1,1,1/, g /0,1,0,1,0,0/
data b /0,0,1,0,1,1/
call fglPushMatrix
call fglColor3f(r(i), g(i), b(i))
call fglTranslatef(0.0, 0.0, 1.0)
call fglRectf(-1.0, -1.0, 1.0, 1.0)
call fglPopMatrix
end

subroutine display
implicit none
include 'GL/fgl.h'
call fglClear(GL_COLOR_BUFFER_BIT+GL_DEPTH_BUFFER_BIT)
call fglPushMatrix
call fglRotatef(30.0, 1.0, 0.0, 0.0)
call fglRotatef(30.0, 0.0, 1.0, 0.0)
call fglScalef(0.5, 0.5, 0.5)
call kvadrat(1)
call fglRotatef(90.0, 0.0, 1.0, 0.0)
call kvadrat(2)
call fglRotatef(90.0, 0.0, 1.0, 0.0)
call kvadrat(3)
call fglRotatef(90.0, 0.0, 1.0, 0.0)
call kvadrat(4)
call fglRotatef(90.0, 1.0, 0.0, 0.0)
call kvadrat(5)
call fglRotatef(180.0, 1.0, 0.0, 0.0)
call kvadrat(6)
call fglPopMatrix
call fglFlush
end

program kocka
external display
include 'GL/fglut.h'
include 'GL/fgl.h'
call fglutinit
```

```
call fglutInitDisplayMode(GLUT_SINGLE+GLUT_DEPTH)
call fglutCreateWindow('Fortran GLUT program')
call fglutDisplayFunc(display)
call fglEnable(GL_DEPTH_TEST)
call fglutmainloop
end
```

Podprogram *kvadrat* je narejen tako, da riše transliran kvadrat v ravnini $z = 1$. To lahko razumemo kot nov primitiv, saj par ukazov Push/Pop ne popravlja transformacije ob klicu podprograma. šest stranic se riše z rotacijo osnovne stranice okoli osi y in x. Modelna matrika se shani z začetnim ukazom *Push* in potem ponovno obnovi z ukazom *Pop*.

4.2 Globinski pomilnik

Da se ploskve v prostoru pravilno izrisujejo tudi takrat, ko rišemo ploskvice za drugimi, je potrebno uporabiti globinski pomnilnik ali *z-buffer*. To pa mora omogočati že sam okenski sistem, zato je potrebno tak način prikaza zahtevati že pri `fglutInitDisplayMode` in kasneje še dopovedati GL stroju, da poleg barve točk na zaslonu shranjuje še koordinato z v svoj pomnilnik. S tem pomnilnikom GL ob rasterizaciji lika za vsako točko ugotovi, če je že kakšna točka po globini pred njim in jo zato ne riše. Z ukazom `fglEnable(GL_DEPTH_TEST)` se zahteva izračunavanje globine, ki jo je potrebno tako kot barvo pred vsakim začetkom risanja pobrisati z ukazom `fglClear`.



Slika 3: Kocka brez spodnjega in zgornjega pokrova (kvadrat(5) in kvadrat(6)

če rišemo zaprte modele, potem notranjosti ni možno videti. Primer odprtega modela kaže slika 3. V takih primerih se ob uporabi prostorskega pomnilnika običajno kar polovica ploskvic modela prekrije v celoti in kasneje na zaslonu ni vidna. Skupna značilnost vseh teh ploskvic, ki se prekrijejo je, da imajo normalo površine negativno ($n_z < 0$). Da se izogemo nepotrebni rasterizaciji teh ploskvic, vključimo `GL_CULL_FACE`. Da pa bo izločanje delovalo, mora imeti GL podatek za normalo površine, ki jo je potrebno podati pred podatki v vozliščih. Za pravilno delovanje globinskega pomnilnika je potrebna tudi nastavitev projekcijske matrike kot je to opisano v §4.4.

4.3 Animacija

Imejmo primer animacije vozil na avtocesti. Predstavljeno bo cestišče v eno smer z dvema pasovoma, voznim in prehitevalnim. Vozila imajo začetni položaj in hitrost. Opazujemo vozišče dolžine 500 metrov. Hitrost vozila med vožnjo se ne spreminja. Spreminja se le položaj vozil (x, y) na cestišču, ki jih izriše podprogram `vozilo`.

```
subroutine display
implicit none
include 'GL/fgl.h'
common /vozila/ y(5), v(5)
real y, v, pas
integer i
data y /0,50,120,170,200/
data v /50,30,45,31,33/
call fglClear(GL_COLOR_BUFFER_BIT)
call fglPushMatrix
call fglRotatef(-45.0, 0.0, 0.0, 1.0)
call fglTranslatef(0.0, -1.0, 0.0)
call fglScalef(0.004, 0.004, 0.004)
call fglColor3f(0.0, 0.0, 0.0)
call fglRectf(-4.0, 0.0, 4.0, 500.0)
call fglTranslatef(0.0, -50.0, 0.0)
do i=1,5
   if (i.ne.5 .and. y(i+1)-y(i).lt.10.0) then
     pas=-2.0
   else
      pas = 2.0
   end if
   call vozilo(y(i), pas)
end do
call fglPopMatrix
call fglutSwapBuffers
end

subroutine vozilo(y, pas)
call fglPushMatrix
call fglColor3f(1.0, 1.0, 1.0)
call fglTranslatef(pas, y, 0.5)
call fglRectf(-2.0, 0.0, 2.0, 6.0)
call fglPopMatrix
end

subroutine ura(n)
common /vozila/ y(5), v(5)
real y, v, dt
dt = 0.1
do i=1,5
  y(i)=y(i)+v(i)*dt
end do
call fglutPostRedisplay
call fglutTimerfunc(100, ura, 0)
end

program Mad Max
```

```
external display
external ura
include 'GL/fglut.h'
integer window
call fglutInit
call fglutInitDisplayMode(GLUT_RGB+GLUT_DOUBLE)
call fglutCreateWindow('Avtocesta')
call fglClearColor(0.0, 0.5, 0.0, 0.0)
call fglutDisplayFunc(display)
call fglutTimerFunc(100, ura, 0)
call fglutMainLoop
end
```

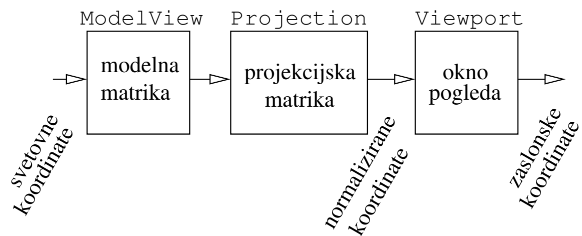
Pas predstavlja odmik v smeri x od sredine cestišča. Vse enote so v metrih. Vozilo je zaradi sorazmerja narisano nekoliko večje. Za animacije je primernejša uporaba dvojnega pomnilnika GLUT_DOUBLE. S tem se izognemo težavam izrisa, saj v trenutku, ko se zgornja plast izrisuje, nemoteno rišemo v spodnjo plast. Ko je spodnja plast izdelana z ukazom *fglutSwapBuffers*, zamenjamo trenutni prikaz.

Za animacijo, pri kateri je zahtevano točno časovno zaporedje je primerno uporabiti uro (*timer*), ki program opozori, da je pretekel predpisani čas in da je potrebno izračunati nov položaj vozil. V našem primeru je podana sprememba vsakih 100 ms in zato nov položaj v smeri y linearno narašča za $v(i)dt$, kjer je hitrost podana v metrih na sekundo. Izbor 0.1s za premik pomeni 1/0.1=10 posnetkov na sekundo, kar je spodnja meja pri animacijah. Po poračunu novih položajev pošljemo sporočilo *fglutPostRedisplay*, da se na novo izriše scena. Lahko bi tudi neposredno klicali display, vendar bi bilo potem potrebno zagotoviti še kompenzacijo hitrosti, saj že v podprogramu ura izgubimo nekaj časa pri izrečunu novih položajev. Prostorski pomnilnik v tem primeru ni potreben, saj je zagotovljeno, da se izrisi prekrijejo v pravilnem vrstnem redu.

4.4 Transformacije pogleda

Za zahtevnejše načine gledanja na model je potrebno nastaviti projekcijo modela iz svetovnih koordinat v normalizirane oz. zaslonske koordinate. V praksi obstajata dva načina projekcije: ortografska in perspektivna. V tehniški predstavitvah se uporablja predvsem paralelna oz. ortografska projekcija. Le v primeru animacije, kjer želimo poudariti bližino in oddaljenost določenih objektov, se uporablja tudi perspektivna projekcija.

OpenGL ločuje projekcijsko matriko in modelno matriko zato, da ni potrebno nastavljati projekcije pri vsakem izrisu. Slika 4 kaže zaporedje transformacij iz svetovnih koordinat v zaslonske. Pri risanju modela običajno začnemo z enotsko *ModelView* matriko.



Slika 4: Zaporedje pretvorbe koordinat vozlišč

Najpreprostejši način prikaza, kot je bil prikazan tudi v dosedanjih primerih je, da stlačimo naš model s transformacijami v privzete normalizirane koordinate [-1, 1]. Pri tem načinu sta tako modelna kot projekcijska matrika enotski. Modelna matrika je enotska le na začetku vsakega risanja, projekcijska pa je konstantna ves čas. Pri takem načinu ni potrebno preklapljati med trenutno aktivnima projekcijama. In če se zadovoljimo s takim načinom, potem zadostuje tudi privzeta zaslonska transformacija pri spremembi velikosti okna, ki je pri sistemu GLUT le enovrstičen ukaz:

```
call fglViewport (0, 0, width, height)
```

Nekoliko zahtevnejša je sorazmerna sprememba, ki ne bo anamorfično popravljala velikosti okna:

```
subroutine  reshape (w, h)
integer w, h
implicit none
include 'GL/fgl.h'
common /viewport/ width, height
integer width, height
real*8 left, right, bottom, top, znear, zfar
width = w
height = h
if (w .ge. h) then
        left = -width/(1.0*height)
        right = width/(1.0*height)
        bottom = -1.0
        top = 1.0
else
        left = -1.0
        right = 1.0
        bottom = -height/(1.0*width)
        top = height/(1.0*width)
end if
znear = -1.0
zfar = 1.0
call fglViewport (0, 0, width, height)
call fglMatrixMode (GL_PROJECTION)
call fglLoadIdentity
call fglOrtho(left, right, bottom, top, znear, zfar)
call fglMatrixMode(GL_MODELVIEW)
end
```

Predstavljeni podprogram se priporoča v uporabo za vse programe, ki pripravljajo model v velikosti [-1, 1] za *ModelView*. če bi želeli dodati modelno transformacijo v *reshape*, potem za zadnjo vrstico dopišemo še modelno transformacijo in nato v programu za izris pred začetkom le obnovimo stanje modelne matrike. Primer animacije 4.3 bi tako imel namesto na-

stavitve modelne transformacije slednje v podprogramu *reshape*. Začetna nastavitev modelne matrike pred začetkom izrisa v podprogramu *display* pa bi bila:

```
call fglClear(GL_COLOR_BUFFER_BIT)
call fglPushMatrix
... izris
call fglPopMatrix
```

Takoj za brisanjem zaslona z ukazom *Push* shranimo modelno matriko in jo ob koncu ponovno nastavimo na začetno vrednost. V podprogramu *reshape*, pa modelno matriko popravljamo:

```
call fglMatrixMode(GL_MODELVIEW)
call fglLoadIdentity
call fglRotatef(-45.0, 0.0, 0.0, 1.0)
call fglTranslatef(0.0, -1.0, 0.0)
call fglScalef(0.004, 0.004, 0.004)
```

Tak pristop nekoliko jasneje predstavi program, saj so vse enote, s katerimi manipuliramo, v programu za izris v svetovnih oz. modelnih koordinatah. V spošnem se priporoča nastavitev projekcije za vse modele, ki uporabljajo izris ploskev. To pa zaradi tega, ker je privzeta projekcijska matrika enotska. Poglejmo to na primeru paralelne projekcije *glOrtho(l,r,b,,n,f)*:

$$PM = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{l-r} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad .$$

Za primer normalizacijskega prostora v obsegu [-1,1] je tako matrika paralelne projekcije

$$PM(-1,1,-1,1,-1,1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ,$$

kar se razlikuje od enotske prav v koordinati z. če bi projekcijsko matriko ohranili enotsko, potem bi to pomenilo, da objekt gledamo kot zrcalno sliko zadnje strani. Nastavitev projekcijske matrike je torej obvezna za vse izrise ploskvic po globini, kot tudi za modele z osenčenjem. Zaradi tega je tudi program za izris kocke nelogično postavil v ospredje modro stranico in ne rdečo.

5 Osvetlitev

Do sedaj predstavljeni primeri so uporabljali le sintetične barve. To pomeni, da se barva vsake ploskvice ne spreminja v odvisnosti od položaja v prostoru. Tak način prikaza je uporaben le za omejen nabor prostorskih modelov. Neprimeren je že za vse modele, ki imajo površine sestavljene iz primitivov in te površine niso ravninske. Za primer kocke (slika 3) je bilo potrebno za vsako stranico nastaviti svojo barvo, da smo lahko dobili vtis prostora. če bi kocko risali le z eno barvo, potem bi dobili na zaslon le obris.

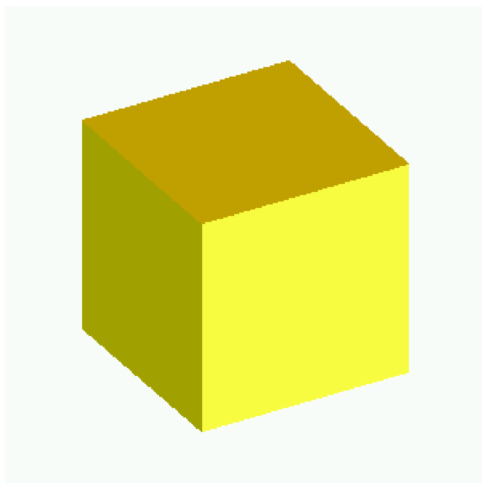
Za bolj realističen izris je potrebno vključiti računanje osvetlitve. žal osvetlitev zajema veliko parametrov, ki jih je potrebno nastaviti preden lahko karkoli dobimo na zaslonu. Tako je potrebno nastavljati položaj in lastnosti luči, osvetlitveni model in lastnosti površin modelov. Za vsako luč se lahko tako nastavi 10 lastnosti in vsaka površina ima 5 lastnosti materiala.

Kot predpogoj za pravilno osvetljen model pa je podana normala v vsakem vozlišču vsake ploskvice. Najpreprostejši način pri uporabi osvetlitve je, da parametre luči ne nastavljamo in da uporabimo nastavljanje lastnosti materiala površine le z ukazom za barvo. S tem vpeljemo veliko predpostavk, ki pa so za šolsko rabo povsem uporabne. Predpostavljena je le ena luč bele svetlobe s položajem $(0,0,1)$ in difuzni odboj svetlobe na površini. Barvo površine podajamo kar z običajnim ukazom za barvo. Program za izris osenčenega modela kocke je tako v minimalni obliki naslednji:

```
subroutine kvadrat()
call fglPushMatrix
call fglTranslatef(0.0, 0.0, 1.0)
call fglNormal3f(0.0, 0.0, 1.0)
call fglRectf(-1.0, -1.0, 1.0, 1.0)
call fglPopMatrix
end

subroutine display
implicit none
include 'GL/fgl.h'
call fglClear(GL_COLOR_BUFFER_BIT+GL_DEPTH_BUFFER_BIT)
call fglColor3f(0.7, 0.6, 0.2)
call fglPushMatrix
call fglScalef(0.5, 0.5, 0.5)
call kvadrat
call fglRotatef(90.0, 0.0, 1.0, 0.0)
call kvadrat
call fglRotatef(90.0, 0.0, 1.0, 0.0)
call kvadrat
call fglRotatef(90.0, 0.0, 1.0, 0.0)
call kvadrat
call fglRotatef(90.0, 1.0, 0.0, 0.0)
call kvadrat
call fglRotatef(180.0, 1.0, 0.0, 0.0)
call kvadrat
call fglPopMatrix
call fglFlush
end

program kocka
external display
external reshape
include 'GL/fglut.h'
include 'GL/fgl.h'
call fglutinit
```

```
call fglutinitdisplaymode(GLUT_SINGLE+GLUT_DEPTH)
call fglutcreatewindow('Osencena kocka')
call fglutDisplayFunc(display)
call fglutReshapeFunc(reshape)
call fglClearColor(1.0, 1.0, 1.0, 1.0)
call fglEnable(GL_LIGHTING)
call fglEnable(GL_LIGHT0)
call fglEnable(GL_DEPTH_TEST)
call fglEnable(GL_COLOR_MATERIAL)
call fglutMainLoop
end

subroutine  reshape (w, h)
integer w, h
implicit none
include 'GL/fgl.h'
real*8 l
l = 1.0
call fglViewport (0, 0, w, h)
call fglMatrixMode (GL_PROJECTION)
call fglLoadIdentity
call fglOrtho(-l, l, -l, l, -l, l)
call fglMatrixMode(GL_MODELVIEW)
call fglLoadIdentity
call fglRotatef(30.0, 1.0, 0.0, 0.0)
call fglRotatef(30.0, 0.0, 1.0, 0.0)
end
```

Razširjeni primitiv smo poenostavili tako, da ne vsebuje več definicije barve, ampak le geometrijo. Obvezno je bilo potrebno podati izračun normale. Za naš primitiv kvadrata je to $(0, 0, 1)$. Program za izris v bistvu ni spremenjen, le da je sedaj transformacija modela preseljena v podprogram za nastavitev velikosti okna `reshape`. V glavnem programu pa je potrebno najprej vključiti računanje osvetlitve *GL_LIGHTING*, prižgati je potrebno luč št 0, ki ima začetni položaj $(0, 0, 1)$.



Slika 5: Osenčen model kocke

Z vključitvijo *GL_COLOR_MATERIAL* pa poenostavimo podajanje barve za material površine tako, da vsi klici podprogramov *Color* nastavljajo privzeto difuzno in ambientno barvo površine. Slika 5 prikazuje rezultat upodabljanja z osvetlitvijo.

6 Tekst

OpenGL sam ne podpira teksta in je zato potrebno uporabiti razne prijeme za izris teksta v prostoru. Možnih je več načinov za risanje besedila:

stroke črke so izrisane s črtami v prostoru modela

bitmap črke so izrisane na zaslon

teksture črke so izrisane rastrsko v prostoru modela

V šolskih primerih so najbolj uporabni že izdelani fonti v knjižnici GLUT. Možne so naslednje številke fontov:

1. GLUT_STROKE_ROMAN
2. GLUT_STROKE_MONO_ROMAN
3. GLUT_BITMAP_9_BY_15
4. GLUT_BITMAP_8_BY_13
5. GLUT_BITMAP_TIMES_ROMAN_10
6. GLUT_BITMAP_TIMES_ROMAN_24
7. GLUT_BITMAP_HELVETICA_10
8. GLUT_BITMAP_HELVETICA_12
9. GLUT_BITMAP_HELVETICA_18

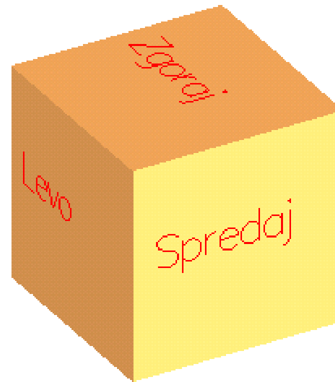
Za primer razširimo program za izris osenčene kocke z besedilom na vsaki stranici. Podprogram *kvadrat* kot argument vzame besedilo. Začetek izpisa premakne za malenkost višje in začne v koordinati $x = -0.8$. Ker pa ne želimo, da se besedilo senči je tu potrebno izklapljanje senčenja takrat, ko izrisujemo posamezne črke. Ker so črke v vnaprej določeni velikost, jih je potrebno ustrezno pomanjšati s skaliranjem. Podprogram `fglutStrokeCharacter` po vsaki izrisani črti sam nastavi pomik v smeri x za širino izrisane črke.

```
subroutine kvadrat(s)
include 'GL/fgl.h'
character s*(*), c
call fglPushMatrix
call fglTranslatef(0.0, 0.0, 1.0)
call fglNormal3f(0.0, 0.0, 1.0)
call fglRectf(-1.0, -1.0, 1.0, 1.0)
call fglTranslatef(-0.8, 0.0, 0.01)
call fglDisable(GL_LIGHTING)
call fglScalef(0.003, 0.003, 0.003)
call fglColor3f(1.0, 0.0, 0.0)
lenc = len(s)
do i=1,lenc
  c = s(i:i)
  call fglutStrokeCharacter(1, ichar(c))
end do
call fglEnable(GL_LIGHTING)
```

```
call fglPopMatrix
end

subroutine display
implicit none
include 'GL/fgl.h'
real mat(4)
data mat /0.9, 0.6, 0.3, 1.0/
call fglClear(GL_COLOR_BUFFER_BIT)
call fglClear(GL_DEPTH_BUFFER_BIT)
call fglPushMatrix
call fglRotatef(30.0, 1.0, 0.0, 0.0)
call fglRotatef(30.0, 0.0, 1.0, 0.0)
call fglScalef(0.5, 0.5, 0.5)
call fglMaterialfv(GL_FRONT, GL_DIFFUSE, mat)
call kvadrat('Spredaj')
call fglRotatef(90.0, 0.0, 1.0, 0.0)
call kvadrat('Desno')
call fglRotatef(90.0, 0.0, 1.0, 0.0)
call kvadrat('Zadaj')
call fglRotatef(90.0, 0.0, 1.0, 0.0)
call kvadrat('Levo')
call fglRotatef(90.0, 1.0, 0.0, 0.0)
call kvadrat('Spodaj')
call fglRotatef(180.0, 1.0, 0.0, 0.0)
call kvadrat('Zgoraj')
call fglPopMatrix
call fglFlush
end

subroutine reshape(w, h)
include 'GL/fgl.h'
integer w, h
real*8 l
l = 1
call fglViewPort(0, 0, w, h)
call fglMatrixMode(GL_PROJECTION)
call fglLoadIdentity
call fglOrtho(-l,l,-l,l,-l,l)
call fglMatrixMode(GL_MODELVIEW)
call fglLoadIdentity
end

program crta
external display
external reshape
include 'GL/fglut.h'
include 'GL/fgl.h'
call fglutinit
call fglutinitdisplaymode(GLUT_SINGLE+GLUT_DEPTH)
call fglutcreatewindow('Fortran GLUT program')
call fglutDisplayFunc(display)
call fglutReshapeFunc(reshape)
call fglEnable(GL_DEPTH_TEST)
call fglEnable(GL_LIGHTING)
call fglEnable(GL_LIGHT0)
call fglClearColor(1.0, 1.0, 1.0, 1.0)
call fglutmainloop
end
```

Podajanje barve za površino je spremenjeno tako, da se ne uporabi funkcije *Color* ampak normalno funkcijo za podajanje lastnosti materiala `fglMaterialfv`. Rezultat kaže slika 6. če bi napisali komentar pred izrisom štirikotnika, potem bi bilo vidno besedilo tudi za ostale (skrite) strani.

Včasih pa raje želimo, da se besedilo na zaslonu ne izrisuje rotirano in senčeno, temveč da se le pojavi na določenem položaju v prostoru in potem izriše v zaslonskih koordinatah. V ta namen uporabimo *bitmap* fonte

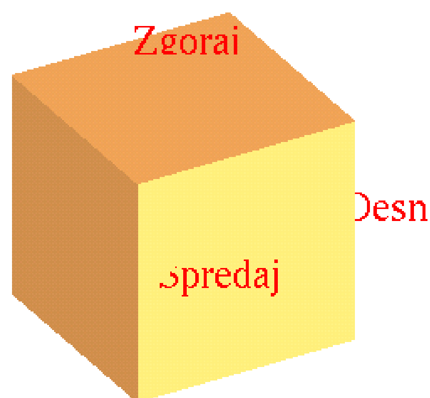


Slika 6: Osenčen model kocke z napisi

in naslednji podprogram za izpis besedila:

```
subroutine output(x,y,z,s)
character s*(*)
call fglRasterPos3f(x,y,z)
lenc = len(s)
do i=1,lenc
  call fglutBitmapCharacter(6, ichar(s(i:i)))
end do
end
```

Primer izrisa z bitmap fonti kaže slika 7



Slika 7: Osenčen model kocke z *bitmap* napisi

7 Uporabniški vmesnik

V nadaljevanju so prikazani primeri programov, ki izkoriščajo dodatne funkcionalnost knjižnice GLUT za vnos dodatnih podatkov v program. To je predvsem uporaba tipk in miške.

7.1 Rotacija s tipkami

Rotiramo že vgrajeni geometrijski model čajnika s tipkami **x, y, z**. Vsak pritisk na tipko poveča kot rotacije za pet stopinj. Podatke o trenutni rotaciji prenašamo s poljem `common`. Ker izrisujemo žični model, podprogram za *reshape* ni potreben. Podprogram *keyboard* ob pritisku na tipko dobi tudi informacijo o zaslonskem položaju miške.

```
subroutine display
implicit none
include 'GL/fgl.h'
common /rotation/ rx, ry, rz
real rx, ry, rz
call fglClear(GL_COLOR_BUFFER_BIT)
call fglColor3f(0.5, 0.4, 1.0)
call fglPushMatrix
call fglRotatef(rx, 1.0, 0.0, 0.0)
call fglRotatef(ry, 0.0, 1.0, 0.0)
call fglRotatef(rz, 0.0, 0.0, 1.0)
call fglutWireTeapot(dble(r))
call fglPopMatrix
call fglutSwapBuffers
end

subroutine keyboard(key,x,y)
common /rotation/ rx, ry, rz
integer key,x,y
print *, 'Key ', char(key),  key, ' at', x, y
if (key .eq. ichar('x')) rx = rx + 5.0
if (key .eq. ichar('y')) ry = ry + 5.0
if (key .eq. ichar('z')) rz = rz + 5.0
call fglutpostredisplay
end

program teapot
external display
external keyboard
include 'GL/fglut.h'
integer window
call fglutInit
call fglutInitDisplayMode(ior(GLUT_DOUBLE,GLUT_RGB))
window = fglutCreateWindow('Use keys x, y, and z')
call fglutDisplayFunc(display)
call fglutKeyboardFunc(keyboard)
call fglutMainLoop
end
```

7.2 Miška in inverzna projekcija

Za vsak pritisk gumba miške lahko dobimo poleg koordinate tudi še stanje gumbov. Naslednji primer prikazuje risanje črte v ravnini (x,y) s tem, da je potrebno zaslonske koordinate pretvoriti nazaj v modelne.

```
subroutine redraw
implicit none
include 'GL/fgl.h'
common /vertices/ n, vertex(2, 100)
integer n, i
real vertex
call fglClear(GL_COLOR_BUFFER_BIT)
call fglbegin(GL_LINE_STRIP)
do i = 1,n
        call fglVertex2f(vertex(1, i), vertex(2, i))
end do
call fglend
call fglFlush
end

subroutine mouse (button, state, x, y)
implicit none
include 'GL/fglut.h'
include 'GL/fgl.h'
include 'GL/fglu.h'
common /vertices/ n, vertex(2, 100)
integer n, i
real vertex
integer button, state, x, y
integer viewport(4)
real*8 mvmatrix(16), projmatrix(16)
real*8 wx, wy, wz               ! returned world x, y, z coords
real*8 px, py, pz               ! picked window coortinates
integer status

if (button .eq. GLUT_LEFT_BUTTON) then
 if (state .eq. GLUT_DOWN) then
        call fglGetIntegerv (GL_VIEWPORT, viewport)
        call fglGetDoublev (GL_MODELVIEW_MATRIX, mvmatrix)
        call fglGetDoublev (GL_PROJECTION_MATRIX, projmatrix)
        note viewport(4) is height of window in pixels
        px = x
        py = viewport(4) -  y - 1
        pz = 0.0
        print *, ' Coordinates at cursor are ', px, py
        status = fgluUnProject (px, py, pz,  mvmatrix,
                  projmatrix,  viewport, wx, wy, wz)
        print *, 'World coords at z=0.0 are ', wx, wy, wz
        n = n + 1
        vertex(1, n) = wx
        vertex(2, n) = wy
        call fglutPostRedisplay
 end if
end if
end


program main
external redraw
external mouse
include 'GL/fglut.h'
call fglutinit
call fglutinitdisplaymode(ior(GLUT_SINGLE,GLUT_RGB))
call fglutInitWindowSize (500, 500)
call fglutInitWindowPosition (100, 100)
window = fglutcreatewindow('Click in window')
call fglutdisplayfunc(redraw)
call fglutMouseFunc(mouse)
call fglutmainloop
end
```

7.3 Kvaternionska rotacija

Naslednji program prikazuje vrtenje osenčenega čajnika z miško. V ta namem se uporabi že izdelam podprogram v jeziku C, ki ga kličemo iz fortrana in nam omogoča kvaternionsko rotacijo. Za vrtenje enot-

ske krogle je potrebno zaznati tako začetni pritisk na gumb (podprogram *mouse*) kot vse naslednje pomike miške (podprogram *motion*).

```
subroutine display
implicit none
include 'GL/fgl.h'
common /quaternion/ last(4), cur(4)
real last, cur, m(4,4)
call build_rotmatrix(m, cur)
call fglLoadIdentity
call fglMultMatrixf(m)
call fglclear(GL_COLOR_BUFFER_BIT+GL_DEPTH_BUFFER_BIT)
call fglutSolidTeapot(dble(0.5))
call fglutSwapBuffers
end

subroutine motion (x, y)
include 'GL/fglut.h'
include 'GL/fgl.h'
implicit none
integer x, y
common /quaternion/ last(4), cur(4)
common /mousestart/ beginx, beginy
common /viewport/ width, height
integer width, height
integer beginx, beginy
real last, cur
real p1x, p1y, p2x, p2y
p1x = (2.0*beginx - width)/width
p1y = (height - 2.0*beginy)/height
p2x = (2.0 * x - width) / width
p2y = (height - 2.0 * y) / height
call trackball(last,p1x, p1y, p2x, p2y)
call add_quats(last, cur, cur)
beginx = x
beginy = y
call fglutPostRedisplay
end

subroutine mouse (button, state, x, y)
implicit none
integer button, state, x, y
include 'GL/fglut.h'
include 'GL/fgl.h'
include 'GL/fglu.h'
common /mousestart/ beginx, beginy
integer beginx, beginy
beginx = x
beginy = y
end

subroutine reshape(w, h)
include 'GL/fgl.h'
integer w, h
real*8 l
common /viewport/ width, height
integer width, height
width=w
height=h
l = 1
call fglViewPort(0, 0, w, h)
call fglMatrixMode(GL_PROJECTION)
call fglLoadIdentity
call fglOrtho(-l,l,-l,l,-l,l)
call fglMatrixMode(GL_MODELVIEW)
call fglLoadIdentity
end

program trackballdemo
implicit none
include 'GL/fglut.h'
include 'GL/fgl.h'
include 'GL/fglu.h'
external display
external motion
external mouse
external reshape
integer window
common /quaternion/ last(4), cur(4)
real last, cur
call trackball(cur, 0.0, 0.0, 0.0, 0.0)
call fglutinit
call fglutinitdisplaymode(GLUT_DOUBLE+GLUT_RGB+GLUT_DEPTH)
window = fglutcreatewindow('Use mouse to rotate')
call fglutdisplayfunc(display)
call fglutmousefunc(mouse)
call fglutmotionfunc(motion)
call fglutreshapefunc(reshape)
call fglEnable(GL_LIGHTING)
call fglEnable(GL_LIGHT0)
call fglEnable(GL_DEPTH_TEST)
call fglutmainloop
end
```

Predstavljeni program je sestavljen iz dveh delov. Kodo za rotacijo v jeziku C `trackball.c` uporabimo kod zunanje podprograme. Rezultat osenčenega model, ki je bil obrnjen z miško, prikazuje slika 8.

Slika 8: Osenčen model čajnika

8 Razvojno okolje

Za šolske probleme smo pripravili razvojno okolje, ki omogoča prevajanje kode v jezikih F77, C++ in C za okenski sistem Windows. Razvojno okolje deluje v načunu ukazne vrstice in nima priloženega integriranega vmesnika. Vsi ukazi za popravljanje programov in prevajanje se tako podajajo v ukazni vrstici DOS okna (*Start-Programs-Command Prompt*).

Osnova okolja je Borlandov C++ prevajalnik, ki ga lahko dobimo zastonj. Besedilo dogovora uporabe se nahaja v datoteki *linence.txt*. Prevajanje v jeziku Fortran pa dosežemo z pretvorbo fortranske kode v C, nato sledi prevajanje v C-ju in povezovanje v končni program (*.exe*). Končni program lahko zaženemo z DOS

okna ali z dvoklikom na izvršni program. Poleg Novega grafičnega okna vsak GLUT program uporablja še konzolo za morebiten vnos ali izpis z ukazoma `print *,` ali `read *,`

8.1 Namestitev

Namestitev je možna z CD-ROMA ali datoteke `bcc-fgl-full.zip`. V slednjem primeru je potrebno paketno datoteko odpakirati v začasen imenik, nakar sledi namestitev tako kot iz CD-ROM-a.

1. Za namestitev je na disku C potrebnih 60 MB prostora!
2. Dvolkikni na `install.bat`

8.2 Dokumentacija

Po namestitvi se navodila z nahajajo v imeniku `c:\bcc55\doc`. Priližena so naslednja navodila v obliki PDF:

redbook-*.pdf OpenGL Programming Guide

opengl-intro.pdf Ta dokument

fgl.pdf OpenGL reference

fglu.pdf OpenGL Utility reference

fglut.pdf GLUT

f2c.pdf Prevajalnik za Fortran

V datoteki se nahaja ta dokument. Iskanje po dokumentaciji za OpenGL (`fgl.pdf, fglu.pdf`), izvedemo tako, da natipkamo npr. `fglVertex3f(`

8.3 Prevajanje

Primeri so v imeniku `c:\bcc55\examples`.

Pred prevajanjem je potrebno odpreti okno DOS *Start-Run-command-OK* in nastaviti pot do prevajalnikov z ukazom

```
PATH=\BCC55\bin;%PATH%
```

Pomik v imenik naredimo z ukazom

```
c:
cd \bcc55\examples
```

Pot lahko nastavimo tudi za celoten sistem: *Start-Settings-Control Panel-System-evironment-¿PATH* in dopišemo `c:\bcc55\bin;` na začetku ali koncu obstoječe poti.

Za prevajanje fortranskih datotek ne smemo uporabiti končnice `.f` Primer prevajanja začetnega primera `line.f` za izris črte:

```
f77 line
```

Za najzahtevnejši primer čajnika uporabimo hkratno prevajanje fortranskega in C programa, ki oba med seboj tudi poveže v program `tblight.exe`

```
f77 tblight trackball.c
```

če imamo več modulov, potem lahko že prevedene podprograme `.obj` le povežemo v izvršno kodo. Primer:

```
f77 tblight trackball.obj
```

Prevajanje in povezovanje v jeziku C se izvede z klicem prevajalnika *bcc32*. Primer:

```
bcc32 teapot.c
```

8.4 Urejanje fortranskih in C programov:

Na urejanje kode lahko uporabimo DOS-ov urejevalnik EDIT ali Windows notepad. Oba imata svoje slabosti; EDIT ima težave z daljšimi imeni datotek, NOTEPAD pa nima prikaza trenutne vrstice in ob prvem shranjevanju datoteke lepi končnico `.txt`, tako da moramo datoteko kasneje preimenovati v `.f`. Kljub slabostim sta oba urejevalnika primerna za šolske probleme.

V trenutnem imeniku DOS okna odtipkamo izbrani ukaz:

```
notepad teapot.f
edit teapot.f
```

Besedila nalog so objavljena pod novicami in navodili za vaje na `http://www.lecad.uni-lj.si`