UNIVERZITET U ZENICI
MAŠINSKI FAKULTET
KATEDRA ZA MATEMATIKU I RAČUNARSKE TEHNOLOGIJE
PREDMET:METODE KONSTRUIRANJA CAD / CAM

SEMINARSKI RAD

OpenGL

Fetić Admir

Mentor: Doc.dr. Senad Balić

Zenica, april 2005.

Sadržaj

1. Uvod

1.1. Što je to OpenGL?

OpenGL predstavlja primarno okruženje za razvoj interaktivnih 2D i 3D grafičkih aplikacija (OpenGL je skraćenica od *Open Graphics Language*). Predstavljen je prvi put 1992. godine, i od tada OpenGL predstavlja najčešće korišćeni grafički API *(Application Programming Interface)*, koji je donio na hiljade aplikacija za sve vrste računarskih platformi. Pored OpenGL-a, trebalo bi podesiti da se radi i sa GLUT-om (GLUT je skraćenica od *Graphics Language Utility Toolkit*) i u potpunosti je kompatibilan sa OpenGL-om.

Satoji se od nekoliko stotina različitih naredbi kojima se navode objekti i operacije potrebni za izradu interaktivnih 3D aplikacija. Nezavisan je od hardwarea ili operacionog sistema radi implementacije na različitim platformama. Zbog toga ne sadrži naredbe za rad sa prozorima ili prikupljanje podataka od korisnika *(miš, tipkovnica...).* Ne sadrži naredbe za opisivanje složenih 3D objekata, koji se grade iz jednostavnih geometrijskih primitiva: tački, linija i poligona. Moguća je izgradnja sofisticiranijih biblioteka nad OpenGL-om.

1.2. Kako je nastao ?

Istorijat :

- 1980 – IrisGL (Silicon Graphic, preteča OpenGL-a)
- 1992 – OpenGL verzija 1.0
- 1998 – OpenGL verzija 1.2
- 2001 – OpenGL verzija 1.3
- 2002 – OpenGL verzija 1.4
- 2003 – OpenGL verzija 1.5
- 2004 – OpenGL verzija 2.0

Silicon Graphics potaknuo je stvaranje OpenGL-a radi ujedinjavanja industrije oko jednog standarda. Tako je 1992. stvoren Architecture Review Board (ARB) koji upravlja razvojem OpenGL tehnologije.

Članovi ARB-a: Digital Equipment Corporation, Evans & Sutherland, HP, IBM Corp., Intel Corp., Intergraph Corp., Microsoft Corp., Silicon Graphics Inc., Sun Microsystems Inc.

1.3. OpenML , OpenGL ES

Kronosova grupa je nezavisni i neprofitni konzorcij kojeg su u januaru 2000. godine osnovale vodeće svjetske kompanije na području grafike i digitalnih medija s ciljem stvaranja i razvoja bogatih medijskih sadržaja i to kroz razvoj otvorenih standarda za API (*Application Program Interface*) na različitim platformama i uređajima. Takvi otvoreni standardi su danas neophodni za razvoj i kompatibilnost između hardvera, platformi i aplikacija zbog brzih promjena u medijskim tehnologijama i zahtjevima tržišta. Članovi konzorcija uviđaju i mogućnost velikih prihoda u dostavljanju takvih bogatih 3D aplikacija na nove uređaje, te su svjesni hitnosti definiranja grafičkih sučelja s velikim 2D i 3D mogućnostima. Kronosova grupa se sastoji od radnih grupa za OpenML, OpenGL te radne grupe za marketing. U njima mogu sudjelovati predstavnici kompanija sa takozvanim Contributing Membershipom.

Te kompanije su 3d4W, Bitboys Oy, Fathammer, Fuetrek, HI Corporation, Hybrid Graphics, MediaQ, Mitsubishi Electric, Seaweed Systems, Superscape, Symbian, Texas Instruments, Tungsten Graphics, Vicarious Visions & Yumetech.

Rezultat rada konzorcija su specifikacije OpenML i OpenGL ES.

OpenML je otvoreni standard za programsko okruženje koje omogućuje spremanje, transport, procesiranje, prikaz i sinhronizaciju digitalnih medijskih sadržaja. To uključuje 2D i 3D grafiku, audio i video procesiranje, umrežavanje i slično.

OpenGL ES *(OpenGL for Embedded Systems)* je otvoreni standard koji definiše programsko sučelje za korištenje naprednih grafičkih mogućnosti za sve veći broj različitih mobilnih i ručnih uređaja. Temelji se na najjednostavnijem podskupu OpenGL-a i omogućuje jednostavno sučelje između hardvera i softvera. Takvo standardno 3D grafičko sučelje za ugrađene sisteme neophodno je za brzo i jeftino stvaranje različitih 3D grafika i to za većinu mobilnih i ugrađenih platformi.

1.4. Šta omogućava OpenGL ?

OpenGL nam omogućava:

- kreiranje kompleksnih objekata pomoću elementarnih geometrijskih primitiva i njihovo raspoređivanje u sceni
- postavljanje položaja i orijentacije posmatrača u odnosu na scenu
- specificiranje karakteristika osvjetljenja u sceni
- automatsko generisanje slike na osnovu zadatih prethodnih stavki
- manipulacija slika (2D)
- sve ovo se obavlja od grafičkog hardwarea, operativnog sistema i prozorskog okruženja nezavisnog skupa funkcija *(komandi)*

OpenGL ne obezbjeđuje:

- podršku za opis objekata i scene
- NURBS (podrška za parametrizovani opis površina)
- rad sa prozorima
- sjenčenje

Postoje biblioteke koje ovo obezbjeđuju i predstavljene su u sljedećem poglavlju.

2. OpenGL biblioteke i njihove mogućnosti

OpenGL biblioteke može napisati bilo ko, ukoliko te biblioteke prođu testove mogu se nazvati OpenGL, u suprotnom ne, tada se nazivaju MESA.

OpenGL – gl.h*, osnovna biblioteka*

GLU *(OpenGL Utility Library)* ***– glu.h,*** *obavezni dio svake OpenGL imlementacije*

- Transformacija koordinata
- Osnovni objekti
- NURBS
- Sastavni dio OpenGL distribucije

GLX – biblioteke koje proširuju funkcionalnost prozorskog okruženja u smislu podrške OpenGL iscrtavanju, biblioteka za X Windowse

GLUT (*OpenGL Utility Toolkit*) ***– glut.h ,*** *biblioteka koja omogućuje interakciju sa različitim prozorskim okruženjima*

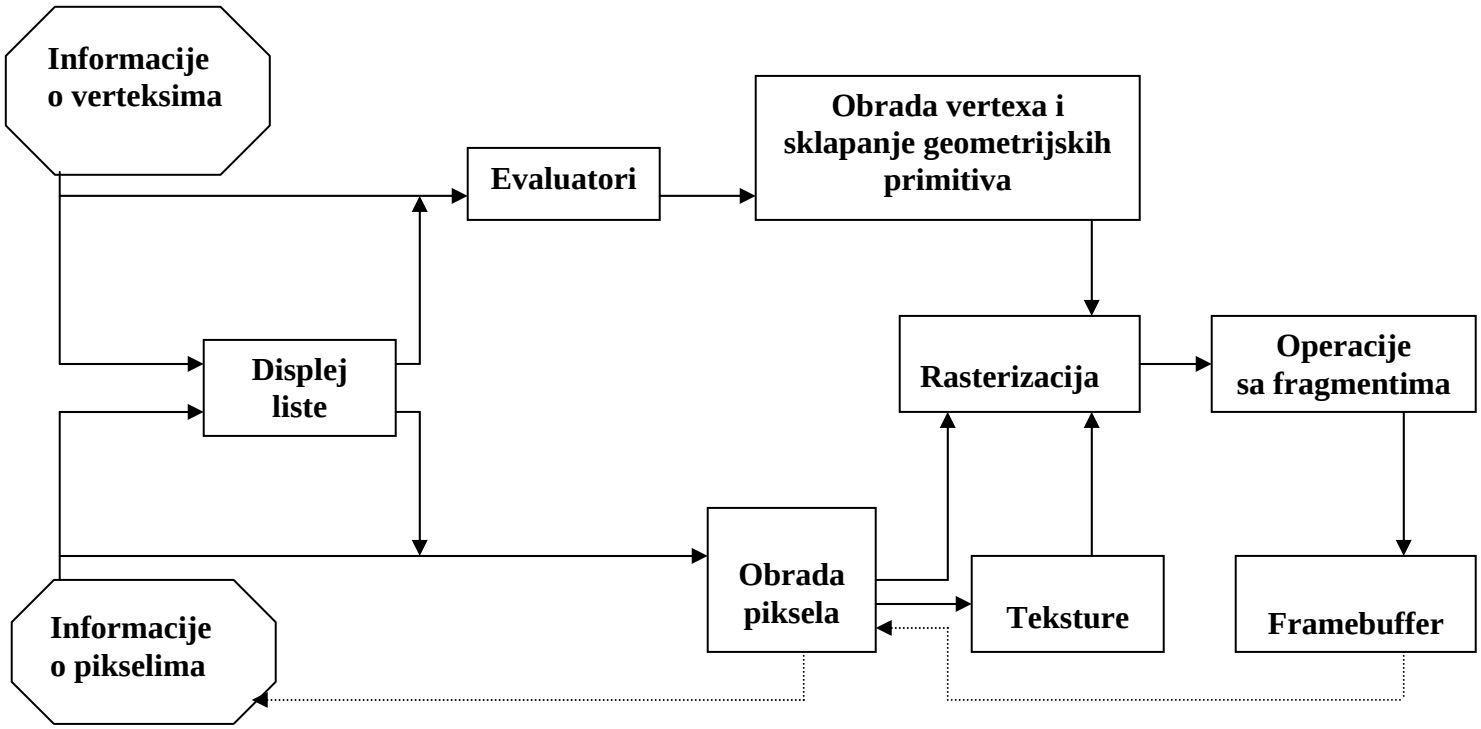
- Rutine za rad sa prozorima (platformski nezavisne)
- Nije dio OpenGL distribucije

Open Inventor, *objektno-orijentisana biblioteka visokog nivoa zasnovana na OpenGL-u*

- Nije dio OpenGL distribucije
- Objekti za rad sa modelima
- Objektno orijentisan

3. Tok podataka

3.1. Blok dijagram OpenGL sistema



3.1.1. Geometrijski podaci

- Svi objekti su opisani pomoću tačaka(vertex)
- Tačka se opisuje sa nekoliko parametara
    - Koordinate (x, y, z, w)
    - Podaci o vektoru normale u tački
    - Podaci o teksturi koja se koristi
    - Boja (RGBA ili indeks)
    - Podaci o ivici (edge-flag)
- Svi ovi podaci procesiraju se zajedno
- Detalji o geometrijskim podacima će biti kasnije objašnjeni

3.1.2. Tok geometrijskih podataka

- Faza "Evaluatori"
    - Objekti mogu biti predstavljeni pomoću evaluatora (Bezier i sl.)
    - U ovoj fazi se te informacije konvertuju u tačke

- Faza "Obrada verteksa"
    - Koordinate i normala se transformišu po potrebi
    - Računa se osvjetljenje i mijenja se boja tačaka
    - Generišu se nove koordinate za teksture (po potrebi)

- Faza "Sklapanje geometrijskih primitiva"
    - Vrši se odsijecanje primitive prema zadatom prozoru
    - Primjenjuje se perspektiva na scenu

- Faza "Rasterizacija"
    - Primitive se konvertuju u fragmente
    - Na objekte se primjenjuju njihove osobine (širina linije, stil ...)
    - Određuju se pikseli koje primitiva zauzima na ekranu
    - Vrijednosti boje i dubine se dodeljuju svakom pikselu

- Faza "Operacija sa fragmentima"
    - U slučaju da teksture postoje svakom fragmentu se pridružuje teksel
    - Nad fragmentima se primjenjuju testovi (scissor test, alpha test, stencil test, depth-buffer test)
    - Fragmentu se dodaje boja pa se upisuje u buffer

3.1.2. Tok podataka o slici (pikseli)

- Faza "Obrada piksela"
  - Vrši se konverzija slike u odgovarajući oblik
  - Po potrebi se vrši i transformacija slike (skaliranje i sl.)
  - Slika potom ide u framebuffer ili u memoriju tekstura

- Dohvatanje slike iz framebuffer-a
  - Pikseli se mogu prebacivati i iz framebuffer-a u operativnu memoriju
  - Vrše se potrebne transformacije (skaliranjei sl.)

- "Teksture"
  - U memoriji za teksture se čuvaju slike koje će se koristiti kao teksture
  - One mogu biti iz operativne memorije ili iz framebuffer-a

4. Sintaksa OpenGL komandi

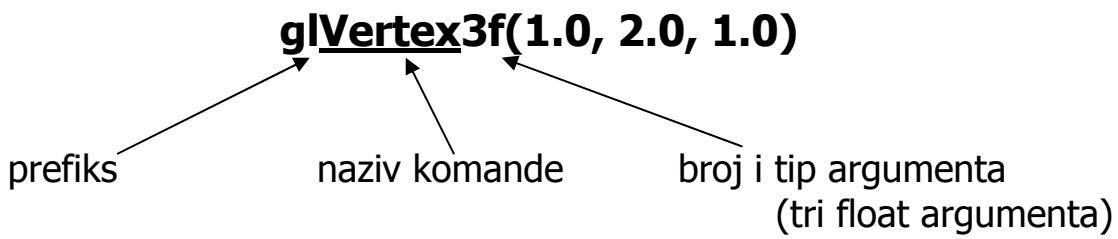
- Sve OpenGL komande imaju prefiks i sufiks
- U slučaju procedura početno slovo svake riječi je veliko
- sve komande imaju prefiks `gl`
- sve konstante imaju prefiks `GL_`
- definisan je određen broj tipova fiksne dužine
  (`GLbyte, GLint, GLfloat, GLdouble, GLubyte,GLuint` itd.)
- pojedine komande mogu imati do tri sufiksa
  - *prvi sufiks* (2, 3 ili 4) označava broj argumenata
  - *drugi sufiks* označava tip argumenata
  - *treći sufiks* (v) označava da se argumenti prenose kao vektor

tip	**prefiks**	**primjer**
poziv procedure	gl	`glEnd()`
konstanta	GL_	`GL_POINTS`
tip podataka	GL	`GLfloat`

4.1. Tabela tipova argumenata i sufiksa OpenGL komandi

Sufiks	**Tip podataka**	**OpenGL tip**	**C tip**
b	8-bitni cio broj	`GLbyte`	`char`
s	16-bitni cio broj	`GLshort`	`short`
i	32-bitni cio broj	`GLint, GLsizei`	`long`
f	32-bitni broj u pokretnom zarezu	`GLfloat, GLclampf`	`float`
d	64-bitni broj u pokretnom zarezu	`GLdouble, GLclampd`	`double`
ub	8-bitni pozitivan cio broj	`GLubyte, GLboolean`	`unsigned char`
us	16-bitni pozitivan cio broj	`GLushort`	`unsigned short`
ui	32-bitni pozitivan cio broj	`GLuint, GLenum, GLbitfield`	`unsigned long unsigned int`

4.2. ***Tipična komanda***



Primjer

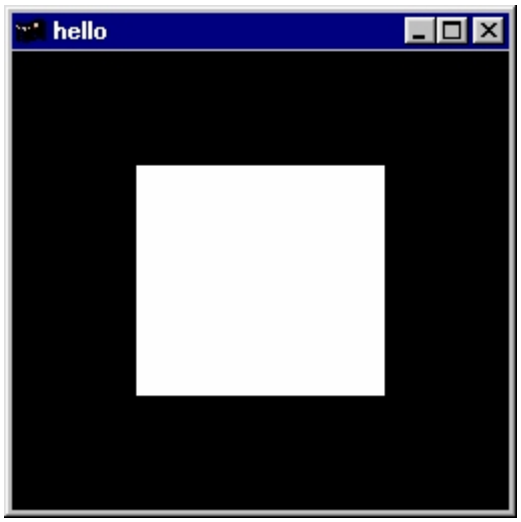
Ovde će biti prikazan jednostavan i uobičajen program koji omogućava iscrtavanje kvadrata u bijeloj boji na crnoj pozadini. Treba napomenuti da korisnik može da zadaje proizvoljnu boju, kao i ostale parametre.

Na početku svakog programa slijedi spisak fajlova koje se smještaju u heder (zaglavlje) programa i koji nose sa sobom potrebne informacije o funkcijama.

```
/* Jednostavan program za upoznavanje sa OpenGL-om */
#include <GL/glut.h>

void display(void)
{

/* "Čiste" se svi pikseli */
   glClear (GL_COLOR_BUFFER_BIT);

/* Iscrtava se bijeli pravougaonik sa koordinatama tjemena
 * (0.25, 0.25, 0.0) i (0.75, 0.75, 0.0) */
   glColor3f (1.0, 1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.25, 0.25, 0.0);
      glVertex3f (0.75, 0.25, 0.0);
      glVertex3f (0.75, 0.75, 0.0);
      glVertex3f (0.25, 0.75, 0.0);
   glEnd();

/* Počinje procesiranje OpenGL rutina */
   glFlush ();
}

void init (void)
{

/* Bira se boja za prebrisavanje   */
   glClearColor (0.0, 0.0, 0.0, 0.0);

/* Inicijalizuju se koordinate pogleda */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
```

```
/*
 * Definiše se inicijalna veličina prozora, pozicija i ekranski
mod.
 * Otvara se prozor sa hello naslovom.
 * Pozivaju se inicijalne rutine.
 */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (250, 250);
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello");
   init ();
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

Kao rezultat se pojavljuje prozor sa prikazanim kvadratom i sa pripadajućim parametrima. Korisnik može da mijenja sve dostupne parametre, kao što su veličina, pozicija, boja, itd.



Kompletan listing programa:

```
#include <GL/glut.h>

void display(void)
{

   glClear (GL_COLOR_BUFFER_BIT);

   glColor3f (1.0, 1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.25, 0.25, 0.0);
      glVertex3f (0.75, 0.25, 0.0);
      glVertex3f (0.75, 0.75, 0.0);
      glVertex3f (0.25, 0.75, 0.0);
   glEnd();

   glFlush ();
}

void init (void)
{

   glClearColor (0.0, 0.0, 0.0, 0.0);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (250, 250);
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello");
   init ();
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

5. OpenGL – State mašina

- OpenGL se može posmatrati i kao stejt mašina
- OpenGL možete postaviti u različita stanja (modove) u kojima on ostaje sve dok ih ne promjenite

- Stanja se kontrolišu pomoću atributa
  - Svi atributi stanja su modalni
  - Primjer : tekuća boja
    - Tekuća boja se može postaviti na bilo koju vrijednost
    - Svi objekti koji se crtaju imaju tekuću boju

- Dvije vrste atributa (neformalna podjela)
  - Promjenjive stanja (boja, debljina linije ...)
  - Kontrolni atributi (kontrolišu koje mogućnosti OpenGL-a su trenutno aktivne)

- Stanja se kontrolišu pomoću
  - `glEnable(GLenum cap)` – uključuje neku od mogućnosti OpenGL-a
  - `glDisable(GLenum cap)` – isključuje neku od mogućnosti OpenGL-a

```
// Uključuje z-buffer testiranje
glEnable(GL_DEPTH_TEST);
// Isključuje osvjetljenje
glEnable(GL_LIGHTING);
```

- Osobine koje se mijenjaju pomoću ovih komandi imaju samo dvije vrijednosti (uključeno - isključeno).Ovih osobina ima mnogo.

6. GLUT – OpenGL Utility Toolkit

- Platformski je nezavistan
- Glut obezbjeđuje sljedeće funkcionalnosti:
  - Više prozora za OpenGL rendering
  - Obrada događaja pomoću "callback" funkcija
  - Sofisticirani ulazni uređaji
  - Tajmere i "idle" rutinu
  - Kaskadni pop-up meniji
  - Rutine za generisanje raznih objekata
  - Podrška za fontove ("bitmap" i "stroke")
  - Razne funkcije za manipulisanje sa prozorima
- GLUT funkcije imaju prefiks glut
- Sve GLUT funkcije imaju malo parametara
- Funkcije ne vraćaju pointere
- Pointeri su prisutni samo u slučaju stringova

- Organizovan je kao skup API-ja zaduženih za :
  - inicijalizaciju
  - započinjanje obrade događaja
  - rad sa funkcijama za obradu događaja
  - rad sa prozorima
  - rad sa menijima
  - rad sa fontovima
  - rad sa geometrijskim objektima

Struktura tipičnog programa koji koristi GLUT :

- Inicijalizacija OpenGL
- Postavljanje osobina prozora
- Kreiranje menija
- Registrovanje callback funkcija
- Ulazak u glavnu GLUT petlju za obradu događaja

Tipična main() rutina

```
int main(int argc, char *argv[]) {
  glutInitDisplayMode(mode);          // određuje OpenGL osobine
  glutInit(argc, argv);               // inicijalizuje GLUT biblioteku
  postaviOsobineProzora();            // pozicija, dimenzije prozora
  glutCreateWindow("Ime aplikacije"); // kreira prozor
  napraviMenije();                    // kreira menije
  napraviCallbackRutine();            // callback
  glutMainLoop();                     // pokreće aplikaciju
  return 0;                           // nikad se neizvršava
}
```

- `void glutInit(int *argcp, char **argv);`
  - Služi za inicijalizaciju GLUT biblioteke
  - Parametri :
    - `argcp` – pokazivač na prvi paramatar main() funkcije
    - `argv` – pokazivač na drugi parametar main() funkcije
  - GLUT se može kontrolisati i pomoću parametara iz komandne linije (otuda i parametri funkcije)

- `void glutInitWindowPosition(int x, int y);`
  - Inicijalizuje poziciju prozora na ekranu
  - Parametri :
    - x – X koordinata gornjeg lijevog ugla prozora
    - y – Y koordinata gornjeg lijevog ugla prozora

- `void glutInitWindowSize(int w, int h);`
  - Inicijalizuje veličinu prozora na ekranu
  - Parametri :
    - w – širina prozora
    - h – visina prozora

- `int glutCreateWindow(char *name);`
  - Kreira osnovni prozor sa imenom *name*
  - Svaki prozor ima jedinstven OpenGL kontekst
  - Promjena OpenGL stanja u jednom prozoru ne utiče na ostale koji su kreirani
  - Stanje OpenGL konteksta može se mijenjati odmah po kreiranju prozora

- `void glutInitDisplayMode(unsigned int mode);`
  - Postavlja inicijalni mod prikazivanja
  - Poziva se (po potrebi) za svaki prozor koji se kreira
  - Parametar mask se dobija logičkom OR funkcijom sljedećih vrijednosti :

GLUT_RGBARGBA	mod za boju (podrazumjevano)
GLUT_RGBRGB	mod za boju
GLUT_INDEX	indeksirani mod za boju
GLUT_SINGLE	prozor sa jednim baferom (podrazumjevano)
GLUT_DOUBLE	dvostruko baferisani prozor
GLUT_ACCUM	prozor sa akumulacionim baferom
GLUT_ALPHA	prozor sa baferom za alfa komponentu boje
GLUT_DEPTH	prozor sa baferom za dubinu
GLUT_STENCIL	prozor sa stencil baferom
GLUT_MULTISAMPLE	prozor sa podrškom za multisampling
GLUT_STEREO	stereo prozor
GLUT_LUMINANCE	luminance mod za boju (slično indeks modu)

- `void glutMainLoop(void);`
  - Poziva se tek pošto se obavi kompletna inicijalizacija
  - Poziva se najviše jedanput iz jednog programa
  - Započinje GLUT-ovu petlju za obradu događaja
  - Iz nje se pozivaju korisničke rutine za obradu događaja
  - Ova rutina se nikada ne završava

6.1. Callback funkcije

- Koriste se za obradu događaja (miš, tastatura ...)
- Poziva ih GLUT kada se nešto dogodi
- Moraju se registrovati da bi GLUT znao da ih pozove
- Moraju imati unaprijed određeno zaglavlje

Primjer :

// Funkcija koja će obrađivati pritisnute tastere
// parametar key – taster koji je pritisnut

```
void obradiTastaturu(unsigned char key, intx, int y) {
     if (key==27) exit(0);          // ako je ESC ==> izlazak iz aplikacije
}
…
int main() {
     …
     glutKeyboardFunc(obradiTastaturu);   // registrujemo  callback
     …
}
```

6.1.2. Registracija Callback funkcija

Registrovanje funkcije za iscrtavanje sadržaja prozora

- `void glutDisplayFunc(void (*func) (void));`
  - Parametar `func` :
    - Funkcija bez argumenata koja ništa ne vraća
    - Pozivaće se svaki put kada je potrebno ponovo iscrtati sadržaj prozora

- `func()` se poziva kada :
  - GLUT procijeni da je potrebno ponovno iscrtavanje
  - Programer eksplicitno pozove `glutPostRedisplay();`
- Postoji posebna funkcija za svaki prozor
- Uvijek se poziva funkcija koja je povezana sa prozorom koji je trenutno aktivan

Registrovanje funkcije koja se poziva pri promjeni dimenzije prozora

- `void glutReshapeFunc(void (*func) (int w, int h));`

  - Parametar `func` :
    - Funkcija sa argumentima koja ništa ne vraća
    - argument w – nova širina prozora
    - argument h – nova visina prozora
  - `func()` se poziva kada :
    - se promjeni veličina prozora
    - se prozor prvi put iscrtava na ekran
  - Ukoliko je `func` NULL ili nije registrovana poziva se `glViewport(0, 0, w, h);`

Registrovanje funkcije koja se poziva kada se pritisne neki taster

- `void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));`

  - Argument `func` :
    - Funkcija sa argumentima koja ništa ne vraća
    - argument key – pritisnuti taster (ASCII)
    - argument x – x koordinata miša (relativna)
    - argument y – y koordinata miša (relativna)
  - Ukoliko je `func` NULL ili nije registrovana događaji tastature se ignorišu

Registrovanje funkcije koja se poziva kada se pritisne ili otpusti dugme na mišu

- `void glutMouseFunc(void (*func)(int button, int state, int x, int y));`

- Parametar `func` :
  - Funkcija sa argumentima koja ne vraća ništa

- argument button – dugme koje pritisnuto :
  - GLUT_LEFT_BUTTON – lijevo dugme
  - GLUT_MIDDLE_BUTTON – srednje dugme
  - GLUT_RIGHT_BUTTON – desno dugme
- argument state – stanje dugmeta :
  - GLUT_UP(otpušteno) GLUT_DOWN(pritisnuto)
- argumenti x i y – pozicija miša

- Ukoliko je `func` NULL ili nije registrovana pritisci dugmadi se ignorišu

Registrovanje funkcije koja se poziva kada se miš pomjeri, a pritisnut je neki od tastera

- `void glutMotionFunc(void (*func)(int x, int y));`

  - Parametar `func` :
    - Funkcija sa argumentima koja ništa ne vraća
    - argumenti x i y – nova pozicija miša

Registrovanje funkcije koja se poziva kada se miš pomjeri, a nije pritisnut nijedan taster

- `void glutPassiveMotionFunc(void (*func)(int x, int y));`

  - Parametar funkcije je isti kao u prethodnoj

7. GREŠKE

- U toku rada mogu nastati razne greške
- OpenGL detektuje jedan dio grešaka
- Kada se greška detektuje
  - OpenGL pamti kod greške koja je nastala
  - Komanda koja je generisala grešku se ignoriše
  - U slučaju GL_OUT_OF_MEMORY rezultat je nepredvidiv

- `GLenum glGetError(void);`
  - Vraća kod greške koja se dogodila
  - U slučaju da greške nema vraća GL_NO_ERROR
  - Resetuje kod greške na GL_NO_ERROR
  - Trebalo bi je pozivati barem jedanput poslije crtanja scene

Greške do kojih može doći u toku rada su :

vrijednost	***opis***
GL_NO_ERROR	greška se nije dogodila
GL_INVALID_ENUMGLenum	argument van opsega
GL_INVALID_VALUE	numerički argument van opsega
GL_INVALID_OPERATION	operacija nije dozvoljena u tekućem stanju
GL_STACK_OVERFLOW	operacija bi dovela do prepunjenja steka
GL_STAC_UNDERFLOW	operacija bi dovela do čitanja sa praznog steka
GL_NO_MEMORY	nema dovoljno memorije da se izvrši komanda

Pored ovih postoje i druge greške:

- 37 GLU NURBS grešaka (GLU_NURBS_ERROR1 ...)
- 14 GLU tesselator grešaka (GLU_TESS_ERROR*)

8. Baferi

- OpenGL koristi nekoliko bafera u toku rada
- Ovi baferi su implementirani u hardveru

naziv	**opis**
frame buffer	čuva piksele za ispis na ekran
z – buffer (depth buffer)	sadrži udaljenost svake tačke na ekranu po z – osi
alpha buffer	sadrži providnost svakog piksela
accumulation buffer	služi za antialiasing
color buffer	čuva podatke o boji (RGB mod)
index buffer	čuva podatke o boji (u indeksnom modu)
stencil buffer	ima više primjena (na primjer da spriječi crtanje na određenom dijelu ekrana)

8.1. Čišćenje bafera

Izbor vrijednosti za čišćenje

- boja (color buffer)
  `glClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf alpha)`
- dubina (z- buffer)
  `glClearDepth(GLclampd depth)`
- stencil
  `glClearStencil(GLint s)`
- akumulacija
  `glClearAccum(GLclampf r, GLclampf g, GLclampf b, GLclampf alpha)`
- indeks
  `glClearIndex(GLfloat index)`

Postavljanje izabrane vrijednosti

- baferi se čiste pomoću komande `glClear(GLint mask)`
- mask je maska koja određuje koji baferi se čiste
- mask se dobija kombinacijom sljedećih vrijednosti :

buffer	***maska***
color	GL_COLOR_BUFFER_BIT
depth	GL_DEPTH_BUFFER_BIT
stencil	GL_STENCIL_BUFFER_BIT
accumulation	GL_ACCUM_BUFFER_BIT

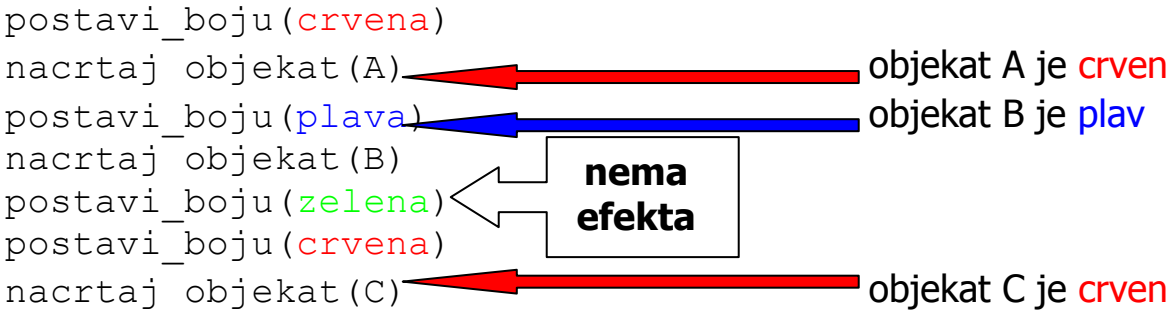
Primjer :

```
glClearColor(0.0, 0.0, 0.0, 0.0);
glClearDepth(0.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

9. BOJA

Tekuća boja se postavlja komandom `glcolor()`. Tako zadata boja primjenjuje se na svaki objekat koji se generiše nakon toga sve dok se novim pozivom iste komande opet ne promijeni tekuća boja. OpenGL računa boju samo u tjemenima primitive, a zatim interpolira boje po unutrašnjosti primitive (osim ako komandom `glShadeModel()` ovo interpoliranje nije isključeno). Dakle:

- Opis objekta je nezavisan od boje kojom se crta
- Svi elementi koji se iscrtavaju imaju tekuću boju
- Tekuća boja se može promijeniti u bilo kom trenutku
- Boje se opisuju pomoću RGB ili RGA modela
- Često se primjenjuje i indeksni mod za boju :
    - Sve vrijednosti boje su upisane u jedan niz
    - Boju piksela određuje indeks boje u nizu



Boja se zadaje pomoću jedne od sljedećih komandi :

```
void glColor3{b i f d ub us ui}(TYPE r, TYPE g, Type b)
void glColor3{b i f d ub us ui}v(TYPE *color)
void glColor4{b i f d ub us ui}(TYPE r, TYPE g, Type b, TYPE a)
void glColor4{b i f d ub us ui}v(TYPE *color)
```

Primjer (ekvivalentne naredbe, sve postavljaju crnu boju) :

```
glColor3f(0.0, 0.0, 0.0);
glColor3b(-128, -128, -128);
glColor3ub(0, 0, 0);
unsigned int boja = {0, 0, 0};
glColor3uiv(boja);
```

10. Shading

- Linije ili popunjeni poligoni se mogu crtati
  - Uvijek jednobojno – flat shading (constant shading)
  - U više boja – smooth shading (Goraud shading)

- Flat shading
  - Svi pikseli koji pripadaju poligonu imaju istu boju
  - Boja poligona određena je bojom prve tačke koja je zadata (boja ostalih tačaka ne utiče)

- Smooth shading
  - Boja svakog verteksa se uzima u obzir
  - Boje piksela u poligonu se interpoliraju između boja svih verteksa kojima je poligon zadat
  - Dva susjedna piksela imaju neznatno različitu boju
  - OpenGL pokušava da napravi gradijent unutar poligona (Gradient fill)
  - U slučaju RGBA boje ovo je u redu
  - U slučaju indeksiranog moda gradijent se izvodi nad indeksima
    - Dva susjedna indeksa mogu pokazivati na različite boje
    - Rješenje je da se napravi gradijent u nizu boja

- Shading se mora zadati pomoću naredbe
  - `void glShadeModel(GLenum mode);`
    - Mijenja način bojenja
    - Parametar mode
      - `GL_FLAT`
      - `GL_SMOOTH`

11. Nasilno iscrtavanje

- Ponekad je potrebno obezbijediti da se scena iscrta prije nego što krenemo sa drugim radom
- Postoje dvije komande za nasilno iscrtavanje :
  - `void glFlush(void)`
  - `void glFinish(void)`
- **`glFlush()`**
  - Forsira početak obrade prethodno zadatih komandi
  - Obezbjeđuje da se one izvrše u konačnom vremenu
  - Ne čeka da se komande izvrše, odmah vraća kontrolu

- **`glFinish()`**
  - Kao `glFlush()` samo što čeka da se komande izvrše

12. Geometrijske primitive

Primitive se zadaju nizom tjemena specificiranih glVertex() komandama između `glBegin()` i `glEnd()` komandi. Svi složeniji geometrijski objekti se kreiraju na osnovu skupa osnovnih primitiva koje su ili tačke ili linije ili poligoni.

- Sve primitive su opisane pomoću tačaka :
  - Tačka opisuje sama sebe
  - Linija se opisuje pomoću krajnjih tačaka, linijom se naziva ono što u matematičkom smislu predstavlja duž.
  - Poligon se opisuje pomoću uglova
- Vertex (nema adekvatan prevod)
  - Interna predstava tačke u hardveru
  - npr. dva ili tri broja u pokretnom zarezu koji opisuju koordinate
  - Sva računanja se obavljaju nad trodimenzionalnim verteksima

12.1. Tačke

- Tačka u OpenGL-u uvijek ima konačne dimenzije
- OpenGL radi sa homogenim koordinatama [x, y, z, w]
  - Homogene koordinate su oblika [x, y, z, w]
  - Ukoliko w nije navedeno podrazumjeva se w=1.0
  - Ukoliko je w različito od 0, homogene koordinate se mapiraju u [x/w, y/w, z/w] 3D koordinate
  - w=0.0 označava zamišljenu tačku u beskonačnosti
  - OpenGL se ne snalazi najbolje kada je w<0.0
- Sve tačke se tretiraju kao 3D
  - Ukoliko z koordinata nije specificirana podrazumjeva se da je njena vrijednost 0.0
- Atributi svih primitiva se postavljaju modalno
  - Oni su globalne promjenjive stanja i mijenjaju se eksplicitno
  - Primitive se crtaju sa aktuelnim vrijednostima atributa

Specifikacija tačaka

- `void glVertex{2 3 4}{s i f d}[v](TYPEcoords);`
  - Tačke se mogu specificirati sa 2, 3 ili 4 koordinate
  - TYPEcoords su koordinate tačaka predstavljene u formatu koji odgovara sufiksima komande
- Primjer :

```
glVertex2s(1, 0);                       [1.0, 0.0, 0.0, 1.0]
glVertex3d(4.0, 2.5, 0.3);              [4.0, 2.5, 0.3, 1.0]
```

```
glVertex4f(1.0, 0.8, 0.0, 3.0);          [1.0, 0.8, 0.0, 3.0]
GLfloat koord[3]{1.0, 5.0, 18.4};
glVertex3fv(koord);                      [1.0, 5.0, 18.4, 1.0]
```

- `void glPointSize(GLfloat size);`
  - Određuje veličinu tačke na ekranu
  - `size` je veličina tačke u pikselima
- Postoji maksimalna veličina tačke
  - Dobija se sa parametrom `GL_POINT_SIZE_RANGE` i funkcijom `glGetFloatv`
- Bez antialiasinga
  - Veličina tačke se zaokružuje
  - Tačka se crta kao `size x size` kvadrat na ekranu
- Sa antialiasingom
  - Veličina tačke se ne zaokružuje
  - Tačka se crta kao krug (približno)
  - Pikseli koji su bliži ivici tačke imaju svjetliju boju radi postizanja glatkosti primitiva

12.2. glBegin()/glEnd()

- `glBegin()/glEnd()` – specificira kod za opis primitive
  - Između njih se pomoću `glVertex*()` specificira lista verteksa
  - Verteksi koji se specificiraju između ovih komandi čine jednu primitivu
  - Postoji ograničenje u vidu komandi koje moguće upotrebiti između `glBegin` i `glEnd`

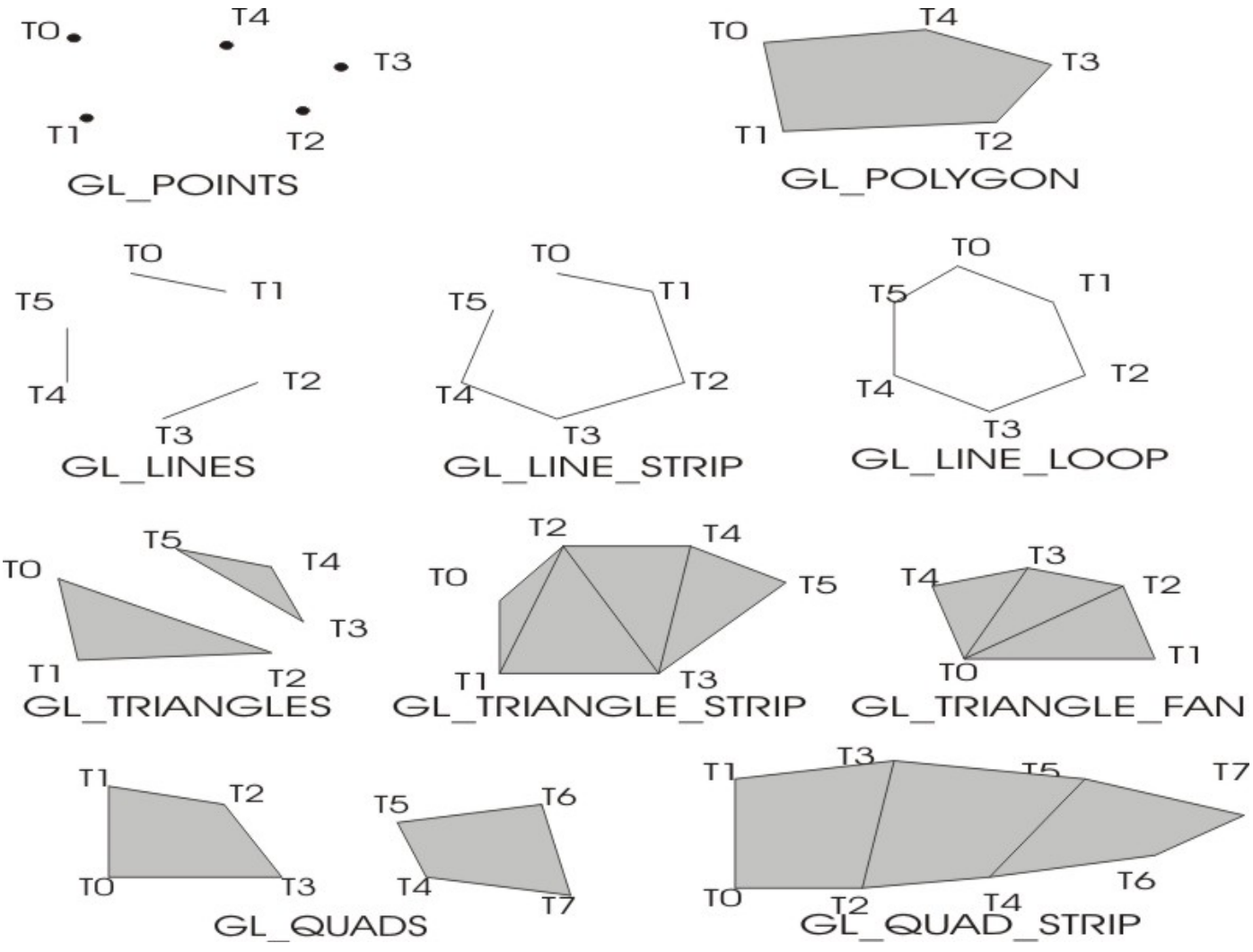
Primjer :

```
glBegin(GL_POLYGON);
          glVertex2f(0.0, 0.0);
          glVertex2f(1.0, 0.0);
     glVertex2f(0.5, 0.866);
glEnd();
```

- `void glEnd();`
  - Označava kraj dijela koda za specificiranje primitive

- `void glBegin(GLenum mode);`
  - Označava početak sekvence komandi za opis primitive
  - mode određuje tip primitive

vrijednost	***značenje***
GL_POINTS	crta individualne tačke
GL_LINES	parovi tačaka se interpretiraju kao individualne linije
GL_LINE_STRIP	svi segmenti su povezani u liniju
GL_LINE_LOOP	kao i prethodno, samo što je dodat segment između posljednje i prve tačke
GL_TRIANGLES	po tri tačke obrazuju jedan trougao
GL_TRIANGLE_STRIP	kao i prethodno, samo što jedna tačka može biti tjeme više trouglova
GL_TRIANGLE_FAN	po tri tačke obrazuju trouglove pri čemu nulta tačka pripada svakom trouglu
GL_QUADS	po četiri tačke obrazuju četvorouglove koji se crtaju
GL_QUAD_STRIP	kao i prethodno, samo što jedna tačka može biti u više četvorouglova
GL_POLYGON	sve tačke obrazuju jedan poligon



- Komande koje se mogu koristiti unutar `glBegin/glEnd`

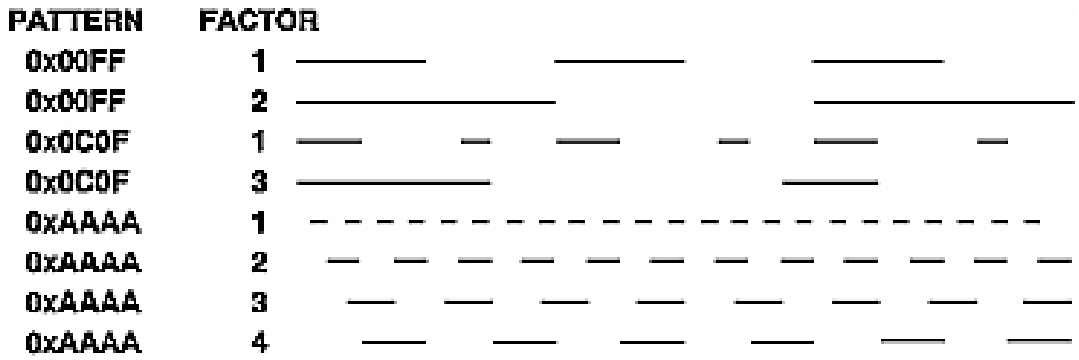
naziv	namjena
`glVertex*()`	postavlja koordinate tačke
`glColor*()`	postavlja tekuću boju
`glIndex*()`	postavlja tekući indeks
`glNormal*()`	postavlja koordinate vektora normale
`glTexCoord*()`	postavlja koordinate teksture
`glEdgeFlag*()`	kontroliše crtanje ivica
`glMaterial*()`	postavlja osobine materijala
`glArrayElement()`	vraća podatke o tačkama
`glEvalCoord*(), glEvalPoint*()`	generiše koordinate
`glCallList(), glCallLists()`	izvršava displej listu (liste)

12.3. Linije - širina

- OpenGL omogućava kontrolu stila i širine linije
- `void glLineWidth(GLfloat width);`
    - Postavlja širinu linije koja se crta
    - Parametar `width` – širina linije u pikselima
    - Širina se tretira kao broj tačaka po x-osi koje linija zauzima (a ne po normali na liniju)
    - Širina se interno zaokružuje na cio broj prije crtanja
    - Antialiasing
        - Linija može imati proizvoljnu širinu (ne zaokružuje se)
        - Ivični pikseli se crtaju sa manjim intenzitetom da bi se postigao efekat glatkosti
    - Postoji ograničenje u pogledu maksimalne širine
- dobija se sa parametrom GL_LINE_WIDTH_RANGE i funkcijom `glGetFloatv`

12.4. Linije - stil

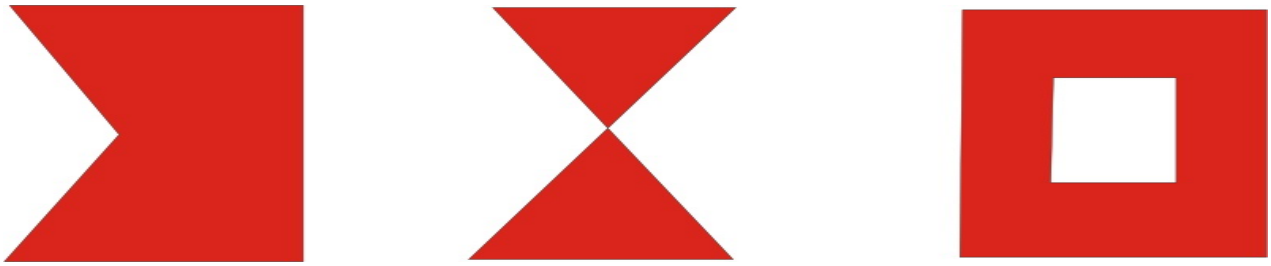
- Linije se iscrtavaju sa stilom koji je unaprijed određen
- `void glLineStipple(GLint factor, GLushort pattern);`
    - Određuje stil kojim se linija crta
    - Parametri
        - pattern – obrazac za crtanje – ako je bit postavljen tačka se crta, u suprotnom ne
        - factor – multiplicira obrazac za crtanje

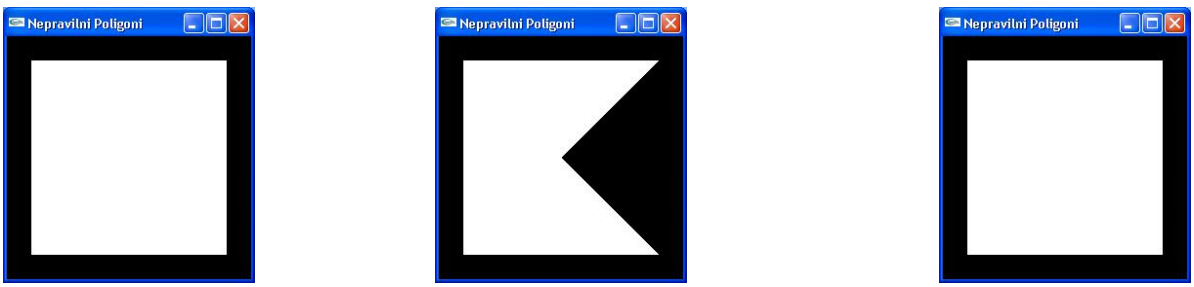


- Mora se omogućiti upotreba stilova
- `glEnable(GL_LINE_STIPPLE)` – omogućuje stilove
- `glDisable(GL_LINE_STIPPLE)` – isključuje upotrebu stilova – podrazumjevana vrijednost
- `glEnd()` – poništava tekući obrazac za crtanje

12.5. Poligoni

- Poligon – oblast ograničena zatvorenom linijom
- Mora biti konveksan, ne smije presijecati samog sebe, i ne smije imati rupu
- Primjer nepravilnih poligona (i kako ih OpenGL crta):





12.6. Poligoni – pravougaonici

- `void glRect{s i f d}(x1, y1, x2, y2);`
- `void glRect{s i f d}[v](tacka1, tacka2);`
  - Crta pravougaonik sa jednim tjemenom u (x1, y1) i drugim tjemenom u (x2, y2)
  - Podrazumijevano je z = 0
  - Ova naredba je ekvivalentna sljedećoj sekvenci

```
glBegin(GL_QUADS);
glVertex(x1, y1);
glVertex(x2, y2);
glVertex(x3, y3);
glVertex(y4, y4);
glEnd();
```

  (x3, y3) i (x4, y4) su određeni na osnovu prve dvije tačke

12.7. Poligoni - popunjavanje

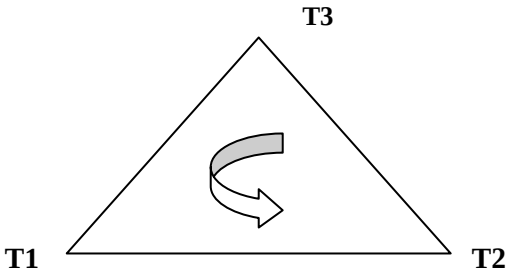
- Poligoni mogu biti popunjeni zadatom maskom
- Maska se zadaje kao 32bit x 32bit matrica
  - Elementi su neoznačeni bajtovi
  - Ukoliko je bit 1 onda se crta, u suprotnom ne
- Popunjavanje poligona mora biti omogućeno
  - `glEnable(GL_POLYGON_STIPPLE)` – omogućuje popunjavanje
  - `glDisable(GL_POLYGON_STIPPLE)` – isključuje popunjavanje – podrazumijevana vrednost
- Pikseli u matrici se interpretiraju onako kako je specificirano pomoću `glPixelStore*()`
  - Podrazumijevano se iz svakog bajta čita prvo MSB

12.8. Poligoni – face culling

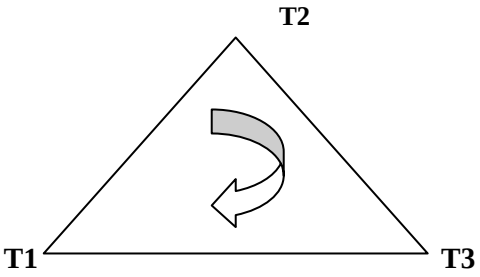
- Svaki poligon ima dva lica – prednje i zadnje
- Podrazumjevano je da se oba lica crtaju
- `void glPolygonMode(GLenum face, GLenum mode);`
  - Specificira kako se tretiraju lica poligona
  - face – lice
    - GL_FRONT – prednja strana (lice)
    - GL_BACK – zadnja strana (lice)
    - GL_FRONT_AND_BACK – obje strane

- mode – mod iscrtavanja strane
  - GL_POINT – crtaju se samo tačke
  - GL_LINE – crtaju se ivice poligona
  - GL_FILL – crta se popunjen poligon

- Redoslijed tačaka određuje orijentaciju poligona
- Ako su tačke zadavane u smjeru kazaljke na satu(CW), poligon je licem naprijed
- U suprotnom slučaju poligon je licem nazad (CCW)



CCW (POZADI)



CW (NAPRIJED)

- `void glFrontFace(GLenum face);`
  - Određuje orijentaciju poligona
  - Parametar `face`
- `GL_CW` – ako su tačke zadavane u CW smjeru poligon je licem naprijed
- `GL_CCW` – suprotno od prethodnog

13. Transformacije

Transformisanje 3D modela u 2D sliku:

1. Primjenjuju se transformacije za modelovanje scene, za pogled i za projekciju
2. Primjenjuje se neki od algoritama za odsjecanje poligona, tj. u sceni ostaje samo ono što će se vidjeti na ekranu
3. Uspostavlja se korespodencija između tačaka koje su ostale poslije koraka 2 i piksela na ekranu (viewport transformacija)

Transformacije se obavljaju preko množenja matrica.
Moguće je putem naredbi promijeniti parametre za svaku transformaciju.

Prevođenje iz 2D u 3D je analogno fotografisanju.

Fotografija se dobija u sljedećim koracima:

1. Postavimo kameru i uperimo je ka sceni (transformacija pogleda)
2. Uredimo objekte koji čine scenu po želi (transformacija modela)
3. Izaberemo sočiva i podesimo zum kamere (transformacija projekcije)
4. Odredimo veličinu slike (viewport)

Pošto se sve obavi, pristupa se fotografisanju (tj. crtanju scene)

13.1. Matrice

- Transformacija je definisana pomoću 4x4 matrice
- Primjer
  - v – verteks koji transformišemo
  - v` – verteks poslije transformacije
  - M – matrica transformacije

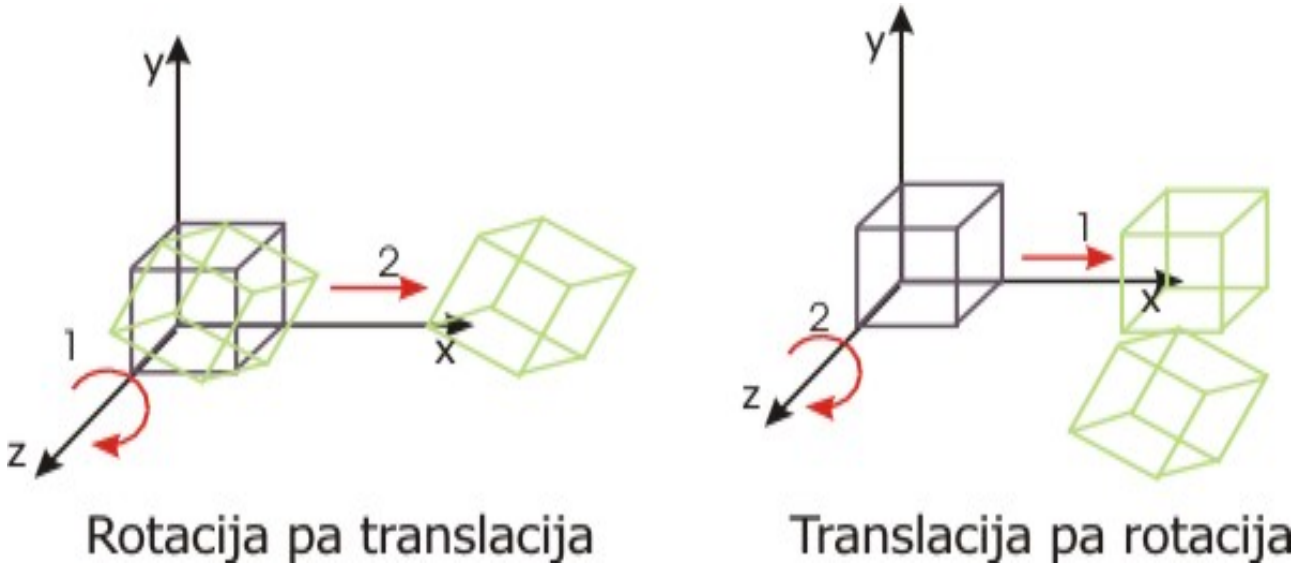
$$v` = \begin{bmatrix} x` \\ y` \\ z` \\ w` \end{bmatrix} = M \cdot v = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- C i C++ smještaju matrice u memoriji po vrstama
- OpenGL ih smješta po kolonama
  `GLfloat m[4][4];` `m[i][j]` – i-ta kolona i j-ta vrsta
  Ovo bi OpenGL protumačio kao i-tu vrstu i j-tu kolonu
- Da bi se izbjegla zabuna radite ovako :

```
GLfloat m[16] = {m0, m1, m2, m3, m4, m5, m6, m7, m8, m9,
m10, m11, m12, m13, m14, m15};
```

$$M = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

- Redoslijed transformacija koje se primjenjuju je bitan
- Ukoliko se transformacije primjenjuju različitim redom krajnji efekat nije isti



13.2. Transformacije modela i pogleda

- Logičan redoslijed
  - Prvo sredimo i postavimo sve objekte na njihova mjesta
  - Potom postavimo kameru na željeno mjesto
- Iz ovoga proizilazi da u programu
  - Prvo zadajemo transformacije pogleda
  - Potom zadajemo transformacije modela
  - Ovo će dovesti do ispravnog redoslijeda obavljanja transformacija
- Dakle, prvo se scena sastavlja, pa se potom postavlja kamera

13.3. Transformacije modela

- Transformacije modela moguće je zadati
  - Preko unaprijed izračunate matrice i `glMultMatrix()`
    - Ovo je vrlo kompikovan proces u slučaju više uzastopnih osnovnih transformacija
  - Preko predefinisanih transformacija
    - Translacija – pomjeramo objekte na željeno mjesto
    - Rotacija – rotiramo objekte za željeni ugao
    - Skaliranje – mjenjamo veličinu objekta
    - Ovo su tzv. afine transformacije
  - Ova varijanta je mnogo brža jer hardver često posjeduje mogućnosti da brzo odradi ove afine transformacije.

13.4. Transformacije pogleda

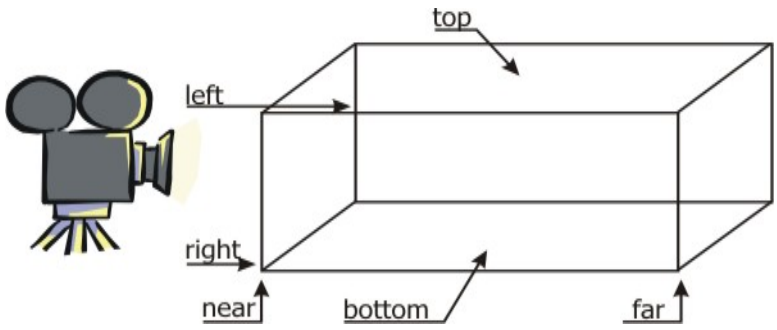
- Kameru možemo pozicionirati na dva načina
  - Putem pomjeranja svih objekata u sceni, pri čemu je lokacija kamere fiksna (transformacija modela)
  - Putem pomjeranja kamere (transformacija pogleda)
- Pomjeranje kamere
  - Ekvivalentno je pomjeranju svih objekata u suprotnom pravcu
  - npr. rotacija kamere u smjeru kazaljke na satu ekvivalentna je rotiranju svih objekata u suprotnom smjeru
- Transformacije pogleda treba da dođu prije transformacija modela

13.5. Transformacije projekcije

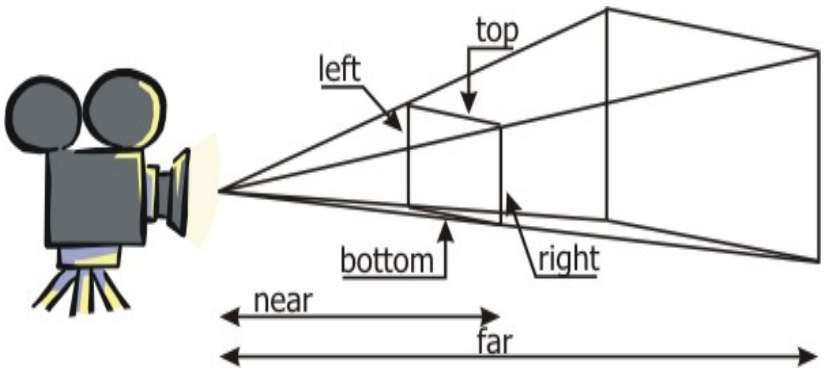
- Definišu prostor u kome se scena nalazi
  - Objekti unutar prostora se crtaju
  - Objekti koji su van prostora se odsijecaju
- Određuju kako se objekti projektuju na ekran
- Transformacije projekcije mogu biti
  - Ortografska (paralelna) projekcija
  - Perspektiva
  - Korisnički definisana projekcija
- Mora se izabrati matrica za rad sa projekcijama pomoću `glMatrixMode(GL_PROJECTION);`
- Pomoću naknadnih rotacija i translacija moguće je promijeniti orijentaciju prostora za odsjecanje

13.6. Paralelna projekcija

- Prostor za odsijecanje je paralolepiped
- Bez dodatnih transformacija prostor je paralelan sa z-osom
- Veličina svih objekata je očuvana (tj. ne zavisi od toga koliko su daleko od kamere)
- Uglovi između pojedinih ivica su očuvani
- Često se koristi u inžinjerskim aplikacijama (CAD i sl.)
- `glOrtho2D(GLdouble left,GLdouble right,GLdouble bottom, GLdouble top);`
  - Isti efekat kao GLortho() samo što podrazumijeva da su sve z-koordinate između –1 i 1



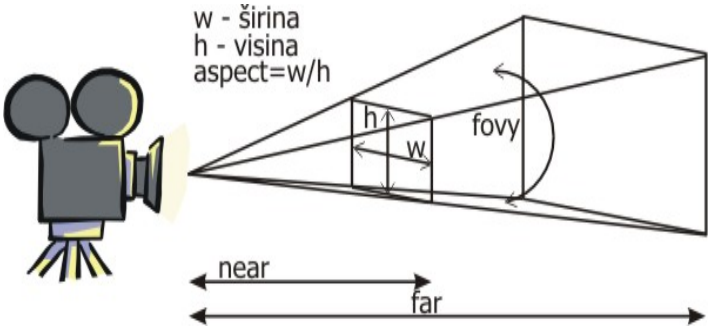
- `void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);`
  - Postavlja perspektivnu projekciju
  - Parametri određuju prostor za odsijecanje (slika)



13.7. Perspektiva

- Veličina objekata se mijenja
- Objekti koji su dalje od kamere izgledaju manji
- Slika se projektuje na vrh piramide koji se nalazi kod kamere
- Ova projekcija bolje odslikava realnost od paralelne
- Često se koristi u animacijama i video igrama
- Ova projekcija može biti asimetrična
- Postoji funkcija `gluPerspective` koja je intuitivnija za korišćenje

- `void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);`
  - Postavlja perspektivnu projekciju
  - Projekcija je uvek simetrična
  - Parametri određuju prostor za odsijecanje (slika)



- `gluPerspective`
  - Parametri
    - `aspect` – odnos širine i dužine bliže ravni (pozitivan broj)
    - `fovy` – ugao koji zauzima piramida po y-osi (između 0 i 180 stepeni)
    - `near` i `far` – određuju ravni odsijecanja po z-osi
  - Na osnovu ova tri parametra piramida je jedinstveno određena
  - Piramida je paralelna pogledu posmatrača
  - Piramida koja se kreira je simetrična u odnosu na ose (x i y osu)

13.8. Transformacije – viewport

- Potrebno je definisati koliki prostor slika zauzima na ekranu (viewport)
- Cijela scena se crta samo unutar tog prostora
- Taj prostor je uvijek pravougaonog oblika
- Njegova veličina je izražena u ekranskim koordinatama (unutar prozora)
- Viewport se zadaje relativno u odnosu na donji lijevi ugao prozora
- Podrazumijevano je da viewport zauzima cio prozor
- `void glViewport(GLint x, GLint y,GLsizei width, GLsizei height);`
  - Određuje prostor unutar prozora u kojem se crta
  - Parametri
    - (`x, y`) – koordinate donjeg lijevog ugla
    - (`width, height`) – dimenzije prostora
- Prostor je pravougaonik određen zadatim parametrima
- Treba biti pažljiv sa veličinom viewporta

13.9. Transformacije – dubina

- Dubina tačaka na ekranu se transformiše prilikom viewport transformacije
- Parametri ove transformacije se mogu promijeniti
- `void glDepthRange(GLclampd near, GLclampd far);`
  - Određuje način preslikavanja z-koordinate u depth bafer
  - Parametri near i far
    - Predstavljaju minimalnu i maksimalnu vrijednost koja se može smjestiti u depth bafer
    - Podrazumijevano je near = 0.0, far = 1.0
    - z-koordinate se transformišu u vrijednosti između near i far

13.10. Transformacije – stekovi

Obzirom da se jednom zadate transformacije primjenjuju na sve naredne geometrijske primitive potreban je mehanizam za izolovanje efekata transformacije. Ovaj mehanizam je implementiran u vidu stekova matrica kojima se manipuliše komandama `glpushMatrix() i glPopMatrix.`

- OpenGL čuva sve matrice na stekovima
- Za svaku vrstu transformacije postoji poseban stek
- Kada primjenimo neku transformaciju ona modifikuje samo matricu koja je na vrhu odgovarajućeg steka
- `glLoadMatrix`, `glMultMatrix` i `glIdentity` utiču samo na vrh steka
- Postoje dvije naredbe za manipulaciju stekom

- `glPopMatrix()` – pomjera sve matrice na steku jedan nivo naniže; matrica na vrhu steka se duplira
- `glPushMatrix()` – skida matricu sa vrha steka i uništava je; ukoliko je stek imao samo jednu matricu
- Obe naredbe rade sa tekućim stekom

- Dubina steka se može saznati pomoću

```
glGetIntegerv(GL_MODELVIEW_STACK_DEPTH, &depth);
glGetIntegerv(GL_PROJECTION_STACK_DEPTH, &depth);
```

- Maksimalna dubina steka se može saznati pomoću

```
glGetIntegerv(GL_MAX_MODELVIEW_STACK_DEPTH, &depth);
glGetIntegerv(GL_MAX_PROJECTION_STACK_DEPTH, &depth);
```

- Pogodni su za konstrukciju hijerarhijskih modela
  - Komplikovaniji objekti su sastavljeni od jednostavnijih

14. Zaključak

OpenGl je prilagodljivo, proceduralno sučelje nezavisno o računarskom sustavu prihvaćeno od mnogih vodećih firmi koje se bave računarskom grafikom, potpuno je dorađen standard i lak je za upotrebu.

Prihvaćen je od mnogih proizvođača grafičkog hardware-a i koristi se u mnogim programskim paketima kao i u mnogim 3d igrama kao što su: Open Inventor, SolidWorks, ProEngineer, LightWave, MultiGen, 3D Studio Max, CosmoPlayer, SoftImage, Quake, Descent 3, Half Life, Duke Nukem Forever, Streets of Sim City...

15. Literatura

Woo, Neider, Davis. *Redbook, OpenGL Programming Guide*, Second Edition, Addison-Wesley Publishing Company

Segal, Akeley. *The OpenGL Graphics Interface*

16. Linkovi:

www.sgi.com/Technology/OpenGL
www.opengl.org
www.workstation.digital.com./products/opengl
www.sun.com/solaris/opengl
www.intergraph.com/ics/graphics/opengl.html
www.austin.ibm.com/software/OpenGL
www.sgi.com/products/software/opengl/examples/redbook
www.eecs.tulane.edu/www/Terry/OpenGL