RAČUNARSKA GRAFIKA

Predavanje broj: 11

Nastavna jedinica: OpenGL

Nastavne teme: Nasilno iscrtavanje. Osnovni primitivi. Linija. Stilovi. Poligoni. Poligoni – popunjavanje. Poligoni – face culling. Opšte transformacije. Transformacija verteksa. Matrice. Transformacije modela: translacija, skaliranje, rotacija. Transformacija pogleda. Transformacija projekcije. View port, dubina, stekovi.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes: "Computer Graphics: Principles and Practice", 2nd ed. in C, Addison-Wesley, 1996.

Nasilno iscrtavanje

- Ponekad je potrebno obezbediti da se scena iscrta pre nego što krenemo sa drugim radom
- Postoje dve komande za nasilno iscrtavanje :
  - `void glFlush(void)`
  - `void glFinish(void)`

`glFlush()`

- Forsira početak obrade prethodno zadatih komandi
- Obezbeđuje da se one izvrše u konačnom vremenu
- Ne čeka da se komande izvrše, odmah vraća kontrolu (asinhrona)

`glFinish()`

- Kao glFlush() samo što čeka da se komande izvrše (sinhrona)

Geometrijske primitive

- Osnove primitive su:
  - tačke,
  - linije
  - poligoni

- Sve primitive su opisane pomoću tačaka :
  - Tačka opisuje samu sebe.
  - Linija se opisuje pomoću krajnjih tačaka.
  - Poligon se opisuje pomoću ćoškova (temena).

- Vertex (teme)
  - Interna predstava tačke u hardveru
    - Npr. dva ili tri broja u pokretnom zarezu koji opisuju koordinate temena
  - Sva računanja se obavljaju nad trodimenzionalnim verteksima

Geometrijske primitive - tačke

- OpenGL radi sa homogenim koordinatama [x, y, z, w]
  - Homogene koordinate su oblika [x, y, z, w]
  - Ukoliko w nije navedeno podrazumeva 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 ne ume da se snađe kada je w<0.0

- Sve tačke se tretiraju kao 3D
  - Ukoliko z koordinata nije specificirana podrazumeva se da je njena vrednost 0.0

- Atributi svih primitiva predstavljaju globalne promenjive stanja i menjaju se eksplicitno.
- Primitive se crtaju sa tekućim vrednostima atributa i sve će biti crtane na isti način dok se ne promeni vrednost atributa.

Geometrijske primitive – tačke

- Specifikacija tačaka :

```
void glVertex{2 3 4}{s i f d}[v](TYPEcoords);
```

  - Tačke se mogu specificirati sa 2, 3 ili 4 koordinate
  - TYPEcoords je lista koordinata tačke predstavljene u formatu koji odgovara sufiksima komande

Primer :

```
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}; <-- [1.0, 5.0, 18.4, 1.0]
glVertex3fv(koord);
```

Geometrijske primitive - tačke

```
void glPointSize(GLfloat size);
```

- Određuje veličinu tačke na ekranu
- Parametar *size* je veličina tačke u pikselima
  - Postoji maksimalna veličina tačke

    Dobija se parametrom GL_POINT_SIZE_RANGE i funkcijom glGetFloatv

```
GLfloat fSizes[2];
glGetFloatv(GL_POINT_SIZE_RANGE,fSizes);//min=0.5,max=10 win.
                                        //sa granulacijom od 0.125
```

- Bez antialiasinga
  - Veličina tačke se zaokružuje
  - Tačka se crta kao *size*$\times$*size* kvadrat na ekranu
- Sa antialiasingom
  - Veličina tačke se ne zaokružuje
  - Tačka se crta kao krug (približno)
  - Pikseli koji su bliži ivici tačke imaju svetliju boju radi postizanja glatkosti primitiva

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 je moguće upotrebiti između glBegin i glEnd

Primer :

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

`void glEnd();`

- Označava kraj dela koda za specificiranje primitive

glBegin()/glEnd()

- Komande koje se mogu koristiti unutar glBegin/glEnd

naziv	namena
`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)

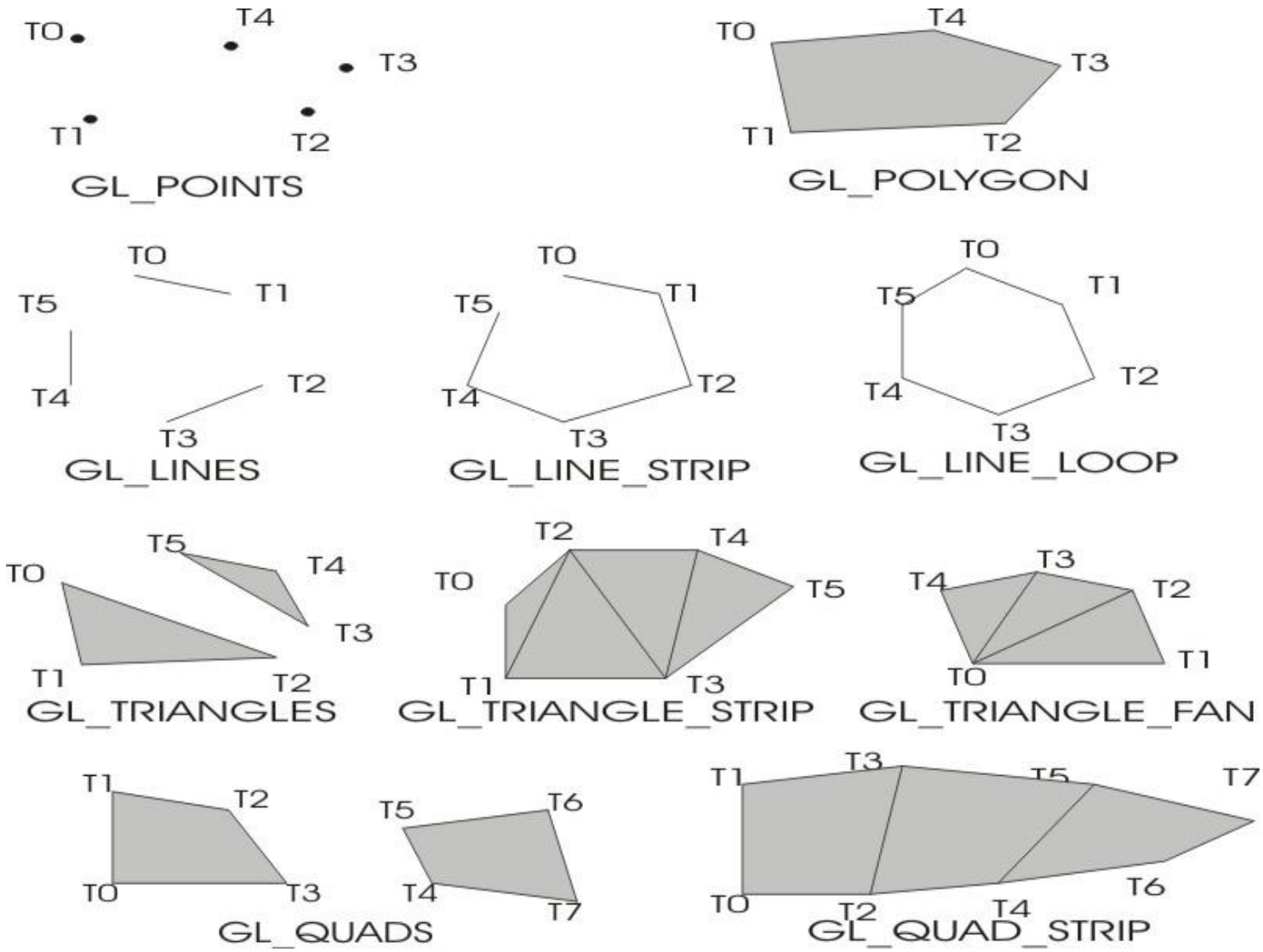
glBegin()/glEnd()

```
void glBegin(GLenum mode);
```

- Označava početak sekvence komandi za opis primitive
- mode određuje tip primitive

Vrednost	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 poslednje i prve tačke
GL_TRIANGLES	po tri tačke obrazuju jedan trougao
GL_TRIANGLE_STRIP	kao i prethodno, samo što jedna tačka može biti teme 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

glBegin()/glEnd()



Linije – širina

- OpenGL omogućuje 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 ceo broj pre 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 npr.

```
GLfloat sizes[2]; GLfloat step;
glGetFloatv(GL_LINE_WIDTH_RANGE,sizes);
glGetFloatv(GL_LINE_WIDTH_GRANULARITY,&step);
```

Linije – stil

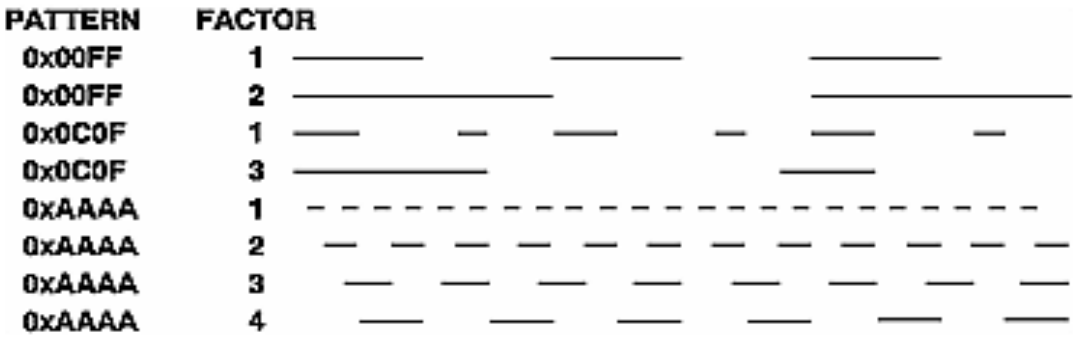
- Linije se iscrtavaju sa stilom koji je unapred određen:

```
void glLineStipple(GLint factor, GLushort pattern);
```

  - Određuje stil kojim se linija crta.
  - Parametri:
    - factor – multiplicira obrazac za crtanje.
    - pattern – obrazac za crtanje – ako je bit postavljen tačka se crta, u suprotnom ne.

- Omogućavanje upotrebe stilova:
  - `glEnable(GL_LINE_STIPPLE)` – omogućuje stilove
  - `glDisable(GL_LINE_STIPPLE)` – isključuje upotrebu stilova – podrazumevana vrednost

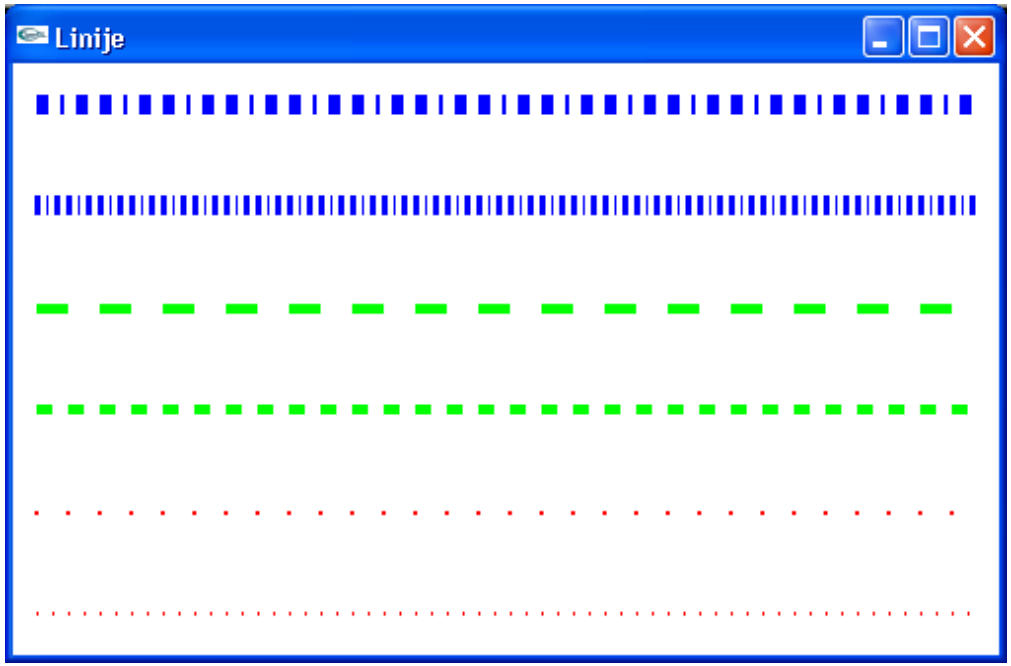
`glEnd()` – poništava tekući obrazac za crtanje.



Linije – primer

```
glEnable(GL_LINE_STIPPLE);
   glColor3f(1.0, 0.0, 0.0);
   glLineWidth(2.0);
   glLineStipple(1, 0x0101);
   nacrtaj_liniju(10, 10, 90, 10);
 glLineStipple(2, 0x0101);
 nacrtaj_liniju(10, 15, 90, 15);
   glColor3f(0.0, 1.0, 0.0);
   glLineWidth(5.0);
   glLineStipple(1, 0x00FF);
   nacrtaj_liniju(10, 20, 90, 20);
 glLineStipple(2, 0x00FF);
 nacrtaj_liniju(10, 25, 90, 25);
   glColor3f(0.0, 0.0, 1.0);
   glLineWidth(10.0);
   glLineStipple(1, 0x1C47);
   nacrtaj_liniju(10, 30, 90, 30);
 glLineStipple(2, 0x1C47);
 nacrtaj_liniju(10, 35, 90, 35);
glDisable(GL_LINE_STIPPLE);
glFlush();
```

```
void nacrtaj_liniju(float x1, float y1,
                    float x2, float y2)
{
    glBegin(GL_LINES);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
    glEnd();
}
```



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 crta se

- Popunjavanje poligona mora biti omogućeno

  `glEnable(GL_POLYGON_STIPPLE)` – omogućuje popunjavanje

  `glDisable(GL_POLYGON_STIPPLE)` – isključuje popunjavanje – podrazumevana vrednost

- Pikseli u matrici se interpretiraju onako kako je specificirano pomoću `glPixelStorei(), npr. glPixelStorei(GL_PACK_LSB_FIRST,true);`
  - Podrazumevano se iz svakog bajta čita prvo MSB, a u prethodnoj naredbi postavlja se čitanje LSB kao prvog

Poligoni – popunjavanje

- Primer

```
GLubyte np[] = { 0x00, … };
glClear(GL_COLOR_BUFFER_BIT);

glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(np);

glBegin(GL_POLYGON);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2f(-2.0, -2.0);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2f( 2.0, -2.0);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f( 2.0,  2.0);
    glColor3f(0.0, 1.0, 1.0);
    glVertex2f(-2.0,  2.0);
glEnd();

glDisable(GL_POLYGON_STIPPLE);
glFlush();
```

Poligoni – face culling

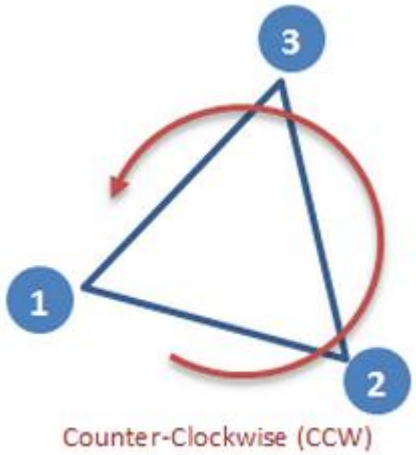
- Svaki poligon ima dva lica:
  - prednje
  - zadnje
- Podrazumevano je da se oba lica crtaju

```
void glPolygonMode(GLenum face, GLenum mode);
```

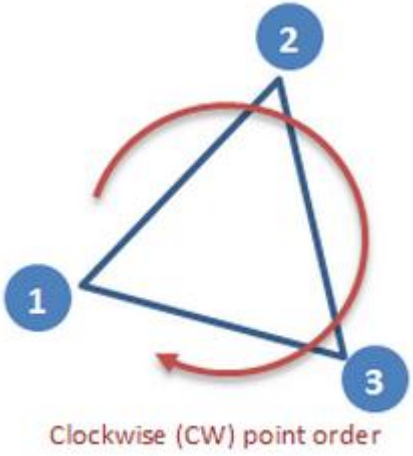
- Specificira kako se tretiraju lica poligona
- face – lice
  - GL_FRONT – prednja strana (lice)
  - GL_BACK – zadnja strana (lice)
  - GL_FRONT_AND_BACK – obe strane
- mode – mod iscrtavanja strane
  - GL_POINT – crtaju se samo tačke
  - GL_LINE – crtaju se ivice poligona
  - GL_FILL – crta se popunjen poligon

Poligoni

- Redosled tačaka određuje orijentaciju poligona
  - Ako su tačke zadavane suprotno smeru kazaljke na satu(CCW), poligon je licem napred
  - U suprotnom slučaju poligon je licem nazad (CW)



Counter-Clockwise (CCW)



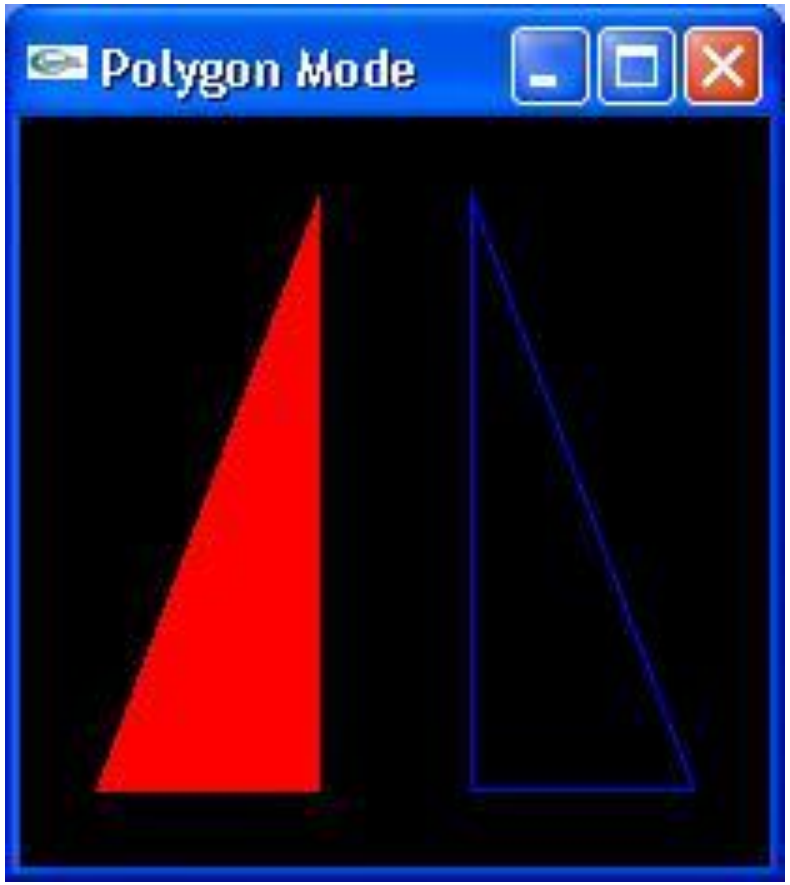
Clockwise (CW) point order

```
void glFrontFace(GLenum face);
```

- Određuje orijentaciju poligona
- Parametar face
  - GL_CCW – lice je određeno CCW orijentacijom (podrazumevano)
  - GL_CW – lice je određeno CW orijentacijom

Poligoni

```
glPolygonMode(GL_FRONT, GL_FILL);
glPolygonMode(GL_BACK, GL_LINE);

     glBegin(GL_TRIANGLES);

        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(-2.0, -2.0);
        glVertex2f(-0.5, -2.0);
        glVertex2f(-0.5,  2.0);

        glColor3f(0.0, 0.0, 1.0);
        glVertex2f( 2.0, -2.0);
        glVertex2f( 0.5, -2.0);
        glVertex2f( 0.5,  2.0);

    glEnd();

glFlush();
```



Transformacije

- Transformisanje 3D modela u 2D sliku:
  - 3D modeli se transformišu u 2D prikaz na ekranu monitora kao što sledi:
    1. Primenjuju se transformacije za:
       - modelovanje scene
       - pogled
       - projekciju
    2. Primenjuje se neki od algoritama za odsecanje poligona, tj. u sceni ostaje samo ono što će se videti na ekranu
    3. Uspostavlja se korenspondencija između tačaka koje su ostale posle koraka 2 i piksela na ekranu (viewport transformacija)
- Transformacije se obavljaju preko množenja matrica
  - sve su 4x4
- Moguće je putem naredbi promeniti parametre za svaku transformaciju

Transformacije – opšte

```
void glMatrixMode(GLenum mode);
```

- Selektuje aktivnu matricu
- Parametar mode
  - `GL_MODELVIEW` – selektuje matricu za transformaciju modela i pogleda
  - `GL_PROJECTION` – selektuje matricu za transformaciju projekcije
  - `GL_TEXTURE` – selektuje matricu za teksture
- Sve naknadne transformacije modifikuju matricu koja je selektovana pomoću ove komande
- Koja je matrica trenutno izabrana može se proveriti preko `glGet(GL_MATRIXMODE)`
- Na početku rada automatski je selektovana matrica za transformaciju modela i pogleda

Transformacije – opšte

```
void glLoadIdentity(void);
```

- Postavlja tekuću matricu na jediničnu 4x4 matricu

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
void glLoadMatrix{f d}(const TYPE *M);
```

- Postavlja tekuću matricu na matricu koja je zadata (M)

- Primer (definisanje sopstvene projekcije)

```
glMatrixMode(GL_PROJECTION);
glLoadMatrix(mojaProjekcija);
```

Transformacije – opšte

```
void glMultMatrix{f d}(const TYPE *M);
```

- Množi tekuću matricu sa zadatom matricom
- Parametar M
  - 4x4 matrica
  - Zadaje se kao niz od 16 vrednosti (double ili float)
- Primer
  - C – tekuća matrica transformacije
  - M – matrica koja je zadata
  - Posle ove naredbe biće C ← CM
- Množenje se u OpenGLu obavlja sa desne strane.
- Redosled primene glMultMatrix (u slučaju više njih) je bitan (množenje matrica nije komutativno)
  - množenje matrica je asocijativno

Transformacije – opšte

- C i C++ smeštaju matrice u memoriji po redovima.
- OpenGL ih smešta po kolonama.

```
GLfloat m[4][4];
```

`m[i][j]` – i-ta kolona i j-ti red

- Da bi se izbegla zabuna trebalo bi raditi ovako :

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

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

Transformacije – opšte

- Ponekad je korisno dohvatiti tekuću matricu

```
void glGetFloatv(GLenum what, GLfloat *m);
```

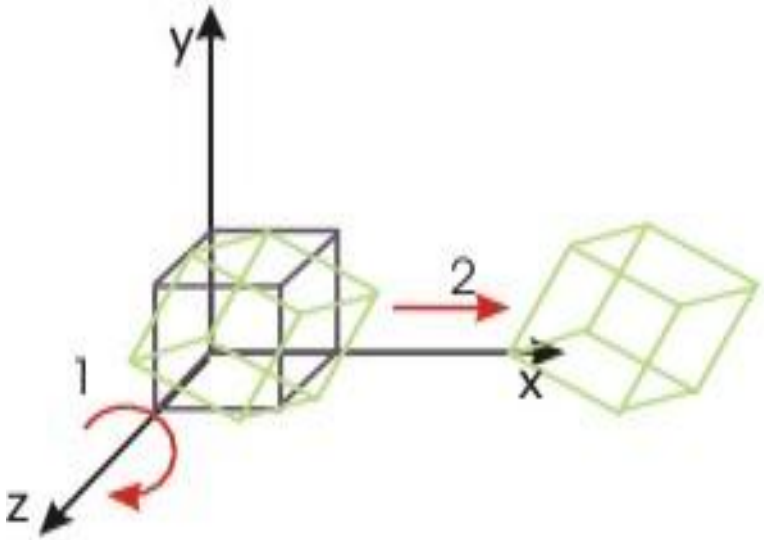
  - Dohvata matricu proizvoljne transformacije
  - Parametar what
    - Određuje koja se matrica traži
    - Kao parametar mode kod glMatrixMode
  - Parametar m
    - Pointer na 4x4 matricu
    - U njega će biti upisana matica

  - Primer (dohvata matricu projekcije)

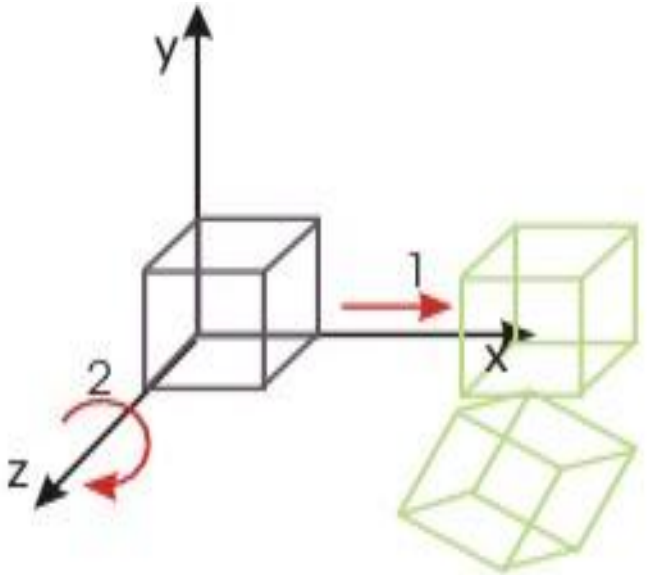
```
GLfloat m[16];
glGetFloatv(GL_PROJECTION, m);
```

Transformacije – osnove

- Redosled transformacija koje se primenjuju je bitan
- Ukoliko se transformacije primenjuju različitim redom krajnji efekat nije isti



Rotacija pa translacija



Translacija pa rotacija

Transformacije – složena transformacija

- Primer

```
glMatrixMode(GL_MODELVIEW);
  glMultMatrix(P);
  glMultMatrix(Q);
  glMultMatrix(R);
  glBegin(GL_POINTS);
    glVertex3fv(v);
  glEnd();
```

- U trenutku crtanja matica transformacije je rezultat množenja matrica PQR.
- Crta se transformisani verteks v`=PQRv=P(Q(Rv))
- Dakle, prvo se primenjuje transformacija koja je poslednja zadata.

Transformacije modela

- Transformacije modela moguće je zadati:
  1. Preko unapred izračunate matrice i `glMultMatrix()`
     - Ovo je vrlo kompikovan proces u slučaju više uzastopnih osnovnih transformacija
  2. Preko predefinisanih transformacija
     - Translacija
       - pomeraju se objekti na željeno mesto (po parametar za svaku koordinatu)
     - Rotacija
       - rotiraju se objekti za željeni ugao oko željene prave u prostoru
     - Skaliranje
       - menja se veličina objekta (po parametar za svaku koordinatu)

  – Druga varijanta je mnogo brža jer hardver često poseduje mogućnosti da brzo odradi Afine transformacije

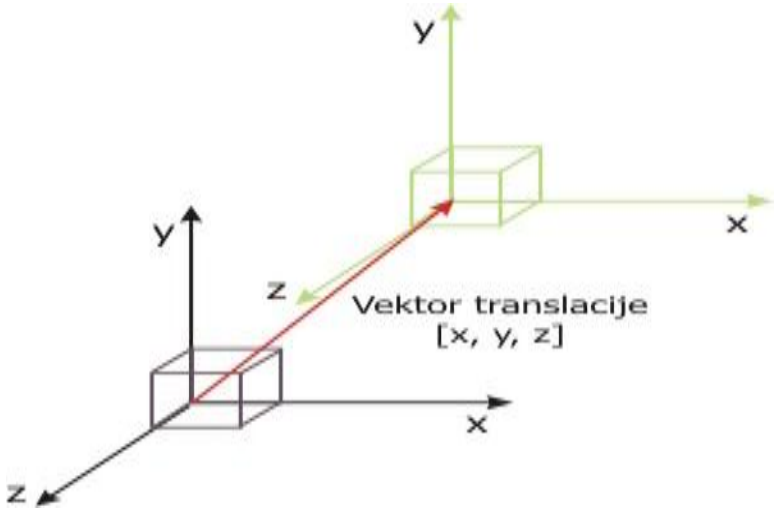
Transformacije modela - translacija

```
void glTranslate{f d}(TYPE x, TYPE y, TYPE z);
```

- Vrši pomeranje objekta za vektor [x, y, z]
- Parametri x, y i z su koordinate vektora translacije

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrica translacije T
kojom se množi tekuća
Matrica (OpenGL postavka)

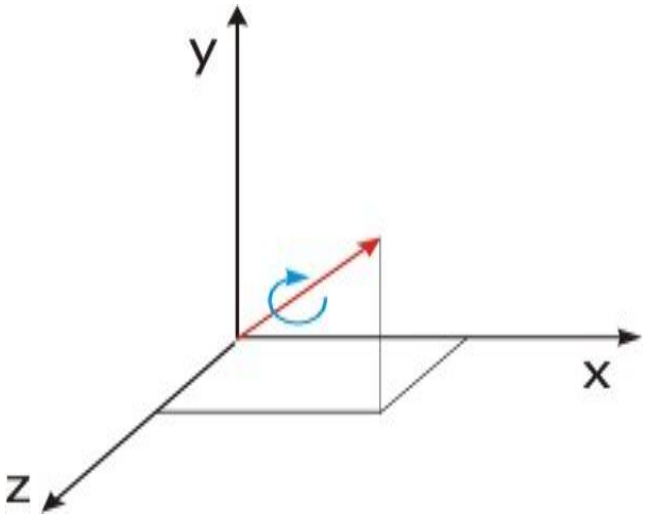


Primer : Translacija objekta i njegovog lokalnog koordinatnog sistema za vektor [x, y, z]

Transformacije modela - rotacija

```
void glRotate{f d}(TYPE angle, TYPE x, TYPE y, TYPE z);
```

- Rotira objekat za ugao *angle* oko vektora [x, y, z]
- Parametar *angle*
  - Ugao za koji se vrši rotacija
  - Izražen je u stepenima od 0 do 360
- Parametri x, y i z – koordinate vektora oko kog se vrši rotacija



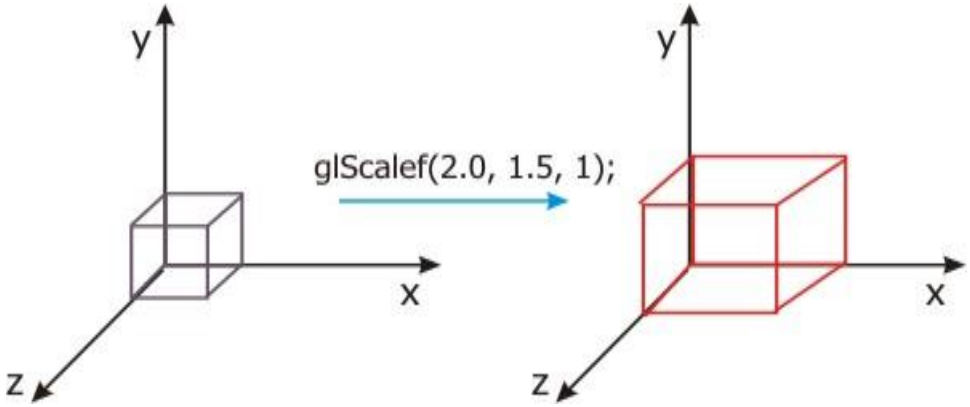
Transformacije modela – skaliranje

```
void glScale{f d}(TYPE sx, TYPE sy, TYPE sz);
```

- Menja veličinu objekta putem skaliranja
- Koordinata svake tačke objekta se množi sa odgovarajućim parametrom (sx, sy ili sz)
- sx, sy i sz određuju promenu u pravcu svake od osa

$$S = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

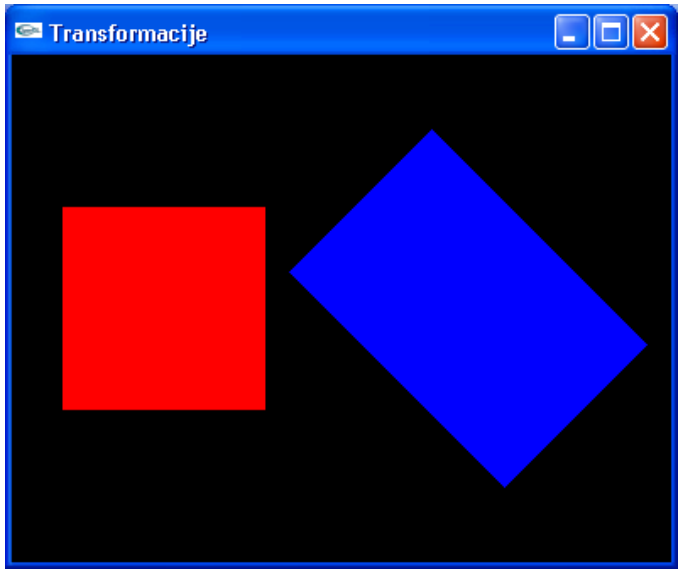
Matrica skaliranja S kojom se množi tekuća matrica(OpenGL postavka)



Primer

Transformacije modela – primer

```
void nacrtaj_kvadrat(void) {
    glBegin(GL_POLYGON);
        glVertex2f(-1.0, -1.0);
        glVertex2f( 1.0, -1.0);
        glVertex2f( 1.0,  1.0);
        glVertex2f(-1.0,  1.0);
    glEnd();
}

glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
  glColor3f(1.0, 0.0, 0.0);
  nacrtaj_kvadrat();
    glColor3f(0.0, 0.0, 1.0);
    glTranslatef(3.0, 0.0, 0.0);
    glRotatef(45.0, 0.0, 0.0, 1.0);
    glScalef(1.0, 1.5, 1.0);
    nacrtaj_kvadrat();
glFlush();
```

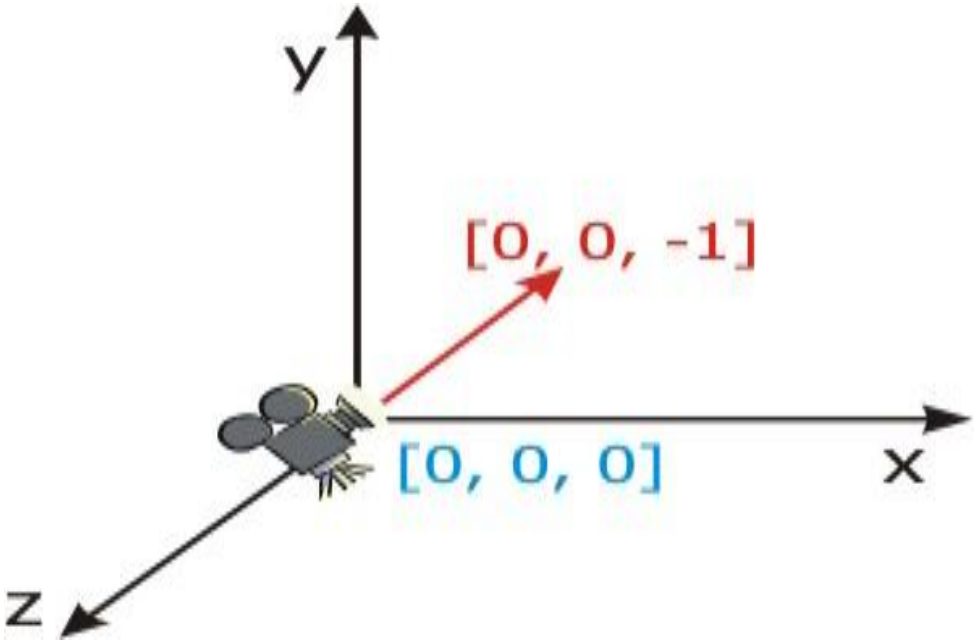


Transformacije pogleda

- Kameru možemo pozicionirati na dva načina
  - Putem pomeranja svih objekata u sceni
    - pri čemu je lokacija kamere fiksna
      - transformacija modela
  - Putem pomeranja kamere
      - transformacija pogleda
- Pomeranje kamere
  - Ekvivalentno je pomeranju svih objekata u suprotnom pravcu
  - npr. rotacija kamere u smeru kazaljke na satu ekvivalentna je rotiranju svih objekata u suprotnom smeru
- Redosled transformacija je:
  - transformacije pogleda
  - transformacije modela

Transformacije pogleda

- Kamera “gleda” u neku tačku
  - To podrazumeva da kamera gleda u tom pravcu
- Podrazumevani položaj kamere
  - Pozicija kamere je [0, 0, 0]
  - Kamera gleda u tačku [0, 0, -1]

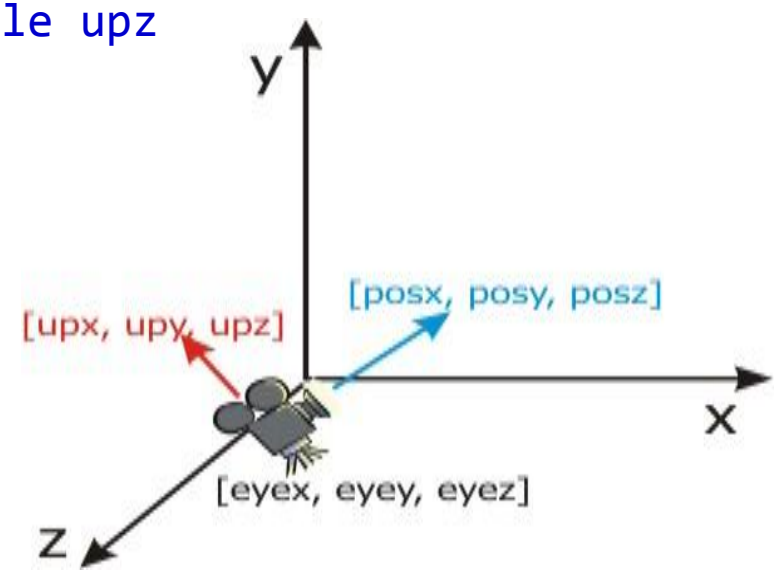


Transformacija pogleda

- Pozicioniranje i orijentacija kamere u 3D prostoru je "komplikovano" tako da je implementirana funkcija koja postavlja položaj i orijentaciju kamere.
- Može se uprostiti pomoću funkcije

```
void gluLookAt(
  GLdouble eyex, GLdouble eyey, GLdouble eyez,
  GLdouble posx, GLdouble posy, GLdouble posz,
  GLdouble upx, GLdouble upy, GLdouble upz
);
```

  - [eyex, eyey, eyez]
    - položaj kamere
  - [posx, posy, posz]
    - tačka u koji kamera "gleda"
  - [upx, upy, upz]
    - orijentacija kamere
    - šta je "gore" a šta je "dole"
    - Ukoliko bi sve ostalo isto, a ovaj vektor obrnemo, slika bi se okrenula za 180 stepeni po vertikali



Transformacije projekcije

- Transformacije projekcije definišu prostor u kome se scena nalazi:
  - Objekti unutar prostora se crtaju
  - Objekti koji su van prostora se odsecaju tako da su vidljivi samo oni delovi koji su u prostoru (ako ih ima)
- Transformacije projekcije određuju kako će se objekti projektovati na ekran.
- Transformacije projekcije mogu biti:
  - Ortografska (paralelna) projekcija
  - Perspektiva
  - Korisnički definisana projekcija

- Mora se izabrati matrica za rad sa projekcijama pomoću

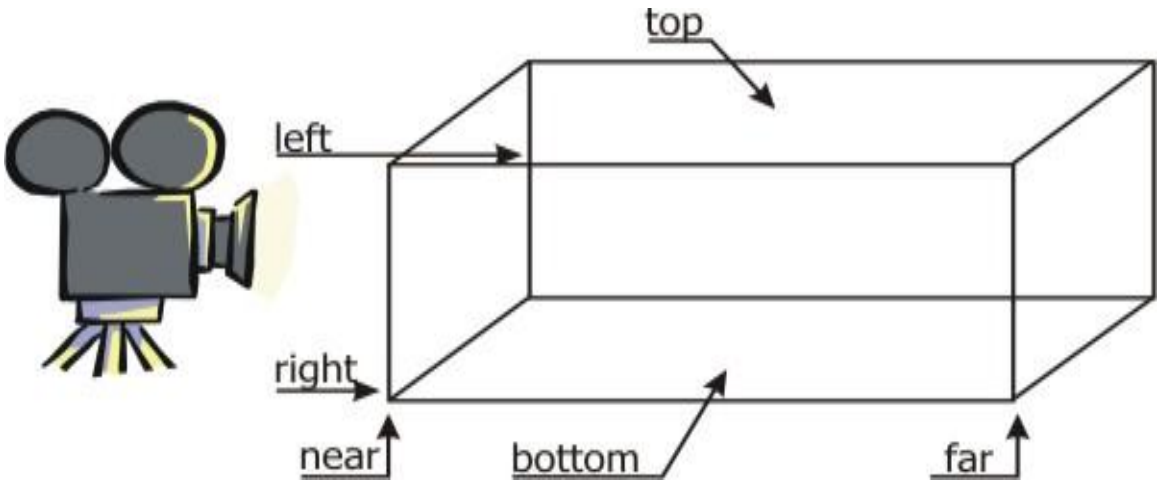
```
glMatrixMode(GL_PROJECTION);
```

- Objekti koji se transformacijama pomere van prostora projekcije biće ignorisani u smislu crtanja, odnosno, biće prikazani samo oni delovi koji se 'vide'.

Transformacije projekcije

```
void glOrtho(GLdouble left,   GLdouble right,
             GLdouble bottom, GLdouble top,
             GLdouble near, GLdouble far);
```

- Postavlja paralelnu projekciju
- Parametri određuju prostor za odsecanje (slika)



Transformacije projekcije

- Paralelna projekcija
  - Prostor za odsecanje je paralelopiped
  - 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 inženjerskim aplikacijama (CAD i sl.)

```
gluOrtho2D( GLdouble left,   GLdouble right,
            GLdouble bottom, GLdouble top);
```

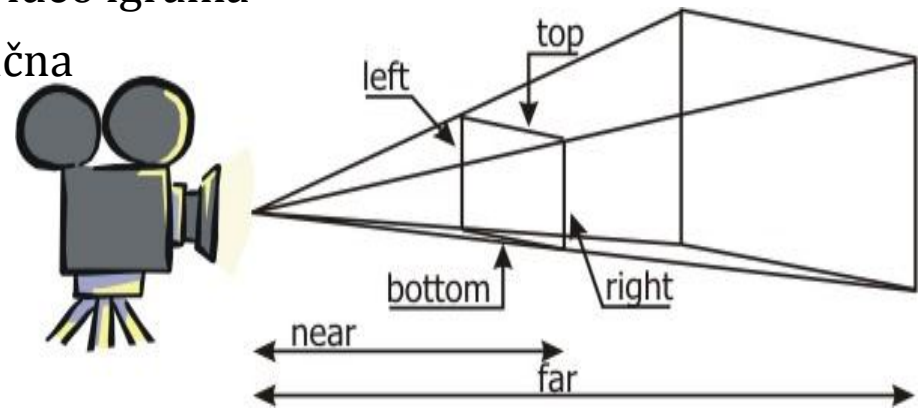
  - Isti efekat kao glOrtho() samo što podrazumeva da su sve z-koordinate između –1 i 1

Transformacije projekcije

```
void glFrustum( GLdouble left,   GLdouble right,
                GLdouble bottom, GLdouble top,
                GLdouble near,   GLdouble far   );
```

- Postavlja perspektivu
- Parametri određuju prostor za odsecanje(slika, krnja piramida)

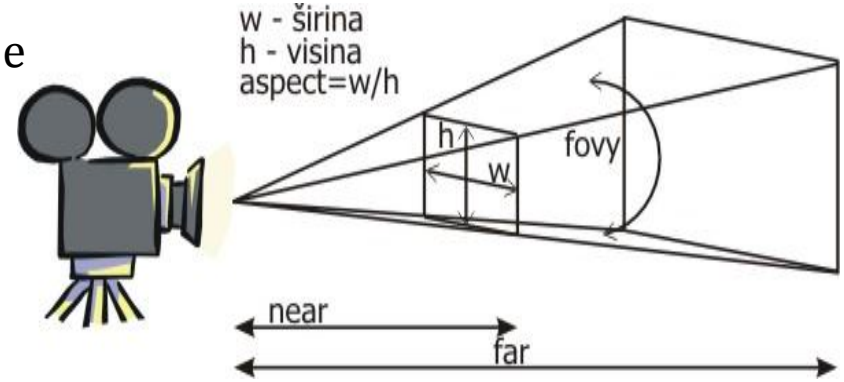
- Perspektiva
  - Veličina objekata se menja
  - Objekti koji su dalje od kamere izgledaju manji
  - Slika se projektuje na vrh piramide koji se nalazi kod kamere
  - Ova projekcija bolje odslikava realnost od paralelne
  - Često se koristi u animacijama i video igrama
  - Ova projekcija može biti asimetrična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



Transformacije projekcije

```
void gluPerspective(GLdouble fovy,
                    GLdouble aspect,
                    GLdouble near, GLdouble far);
```

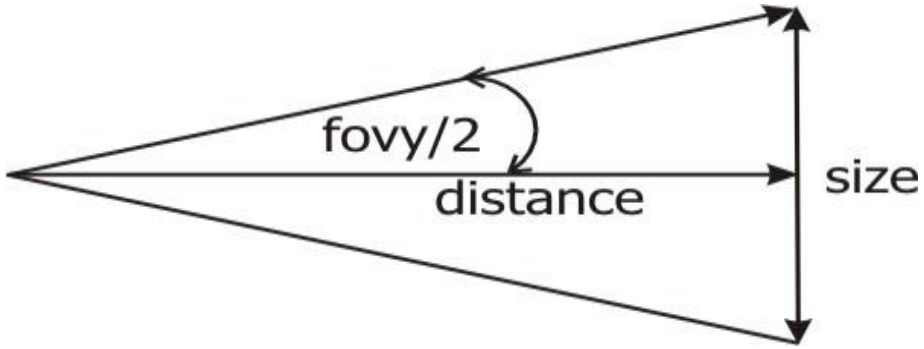
- Postavlja perspektivu
- Projekcija je uvek simetrična
- Parametri određuju prostor za odsecanje



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

Transformacije projekcije

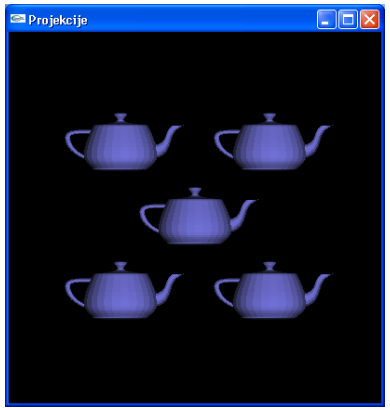
- Ponekad se objekat ne vidi ceo, ili se ne vidi u pravoj veličini (iako je kamera dobro postavljena)
- Potrebno je na osnovu veličine objekta odrediti projekciju (fovy)
- Za to se može napisati funkcija u C-u



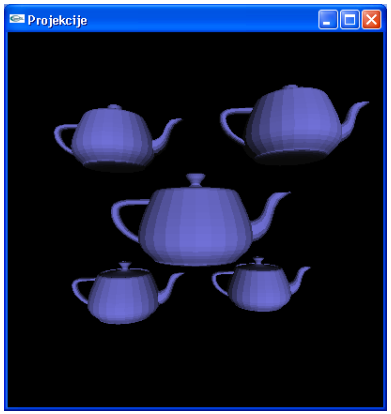

```
double fovy(double distance, double size) {
    double angle;
    angle = 2.0*atan2(size/2.0, distance);
    return (180*angle/PI);
}
```

Transformacije projekcije – primer

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-5, 5, -5, 5, -10, 10);
```




```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-2,2, -2,2, 3.0,10);
```




```
glTranslatef(0.0, 0.0, 3.0);
// Centralni cajnik, z=3.0
glutSolidTeapot(1.0);
glTranslatef(2.0, 2.0, -1.0);
// Gornji desni cajnik, z=2.0
glutSolidTeapot(1.0);
glTranslatef(-4.0, 0.0, -1.0);
// Gornji levi cajnik, z = 1.0
glutSolidTeapot(1.0);
```

```
glTranslatef(0.0, -4.0, -1.0);
// Donji levi cajnik, z = 0.0
glutSolidTeapot(1.0);
// Donji desni cajnik, z = -1.0
glTranslatef(4.0, 0.0, -1.0);
glutSolidTeapot(1.0);
```

Transformacije – viewport

- Potrebno je definisati koliki prostor slika zauzima na ekranu (viewport)
- Cela scena se crta samo unutar tog prostora
- Taj prostor je uvek pravougaonog oblika
- Njegova veličina je izražena u ekranskim koordinatama (unutar prozora)
- Viewport se zadaje relativno u odnosu na prozor opengl-a
- Podrazumevano je da viewport zauzima ceo prozor

Transformacije – viewport

- `void glViewport(GLint x, GLint y,`

  `GLsizei width, GLsizei height);`
  - Određuje površnu unutar prozora u kome se crta
  - Parametri
    - (x, y) – koordinate donjeg levog ugla
    - (width, height) – dimezije površine viewport-a
  - Površina je određena pravougaonikom koji je definisan parametrima

```
gluPerspective(fovy,   1.0,   near,   far);
glViewPort(0, 0, 400, 200);
```

Viewport je dva puta širi nego viši

Projekciona ravan je iste širine i dužine

Slika će biti izobličena

Transformacije – viewport

- Primer iscrtavanja sa dva viewporta

```
int x = 200;
int y = 400;
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// postavljamo donji viewport
glViewport(0, 0, x, y/2);
glutSolidTeapot(1.0);

// postavljamo gornji viewport
glViewport(0, y/2, x, y/2);
glRotatef(45, 1.0, 1.0, 1.0);
glutSolidTeapot(1.0);
```

Transformacije – dubina

- Dubina tačaka na ekranu se transformiše prilikom viewport transformacije
- Parametri ove transformacije se mogu promeniti

```
void glDepthRange(GLclampd near, GLclampd far);
```

  - Određuje način preslikavanja z-koordinate u depth bafer
  - Parametri near i far
    - Predstavljaju minimalnu i maksimalnu vrednost koja se može smestiti u depth bafer
    - Podrazumevano je near = 0.0, far = 1.0
    - z-koordinate se transformišu u vrednosti između near i far

Transformacije – stekovi

- OpenGL čuva sve matrice na stekovima.
- Za svaku vrstu transformacije postoji poseban stek.
- Kada primenimo neku transformaciju ona modifikuje samo matricu koja je na vrhu odgovarajućeg steka.
- Funkcije:
  - glLoadMatrix
  - glMultMatrix
  - glLoadIdentity

  utiču samo na vrh steka.
- Postoje dve naredbe za manipulaciju stekom

  `glPushMatrix()`
    - postavljanje tekuće matrice na stek

  `glPopMatrix()`
    - uzimanje matrice sa steka

Transformacije – stekovi

- 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);//3382
glGetIntegerv(GL_MAX_PROJECTION_STACK_DEPTH, &depth);//3384
```

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

Primer: "pomeranje" pčelice

```
#pragma comment( lib, "opengl32.lib")
#pragma comment( lib, "glu32.lib")
#pragma comment( lib, "glut32.lib")
#include <GL/glut.h>
GLubyte fly[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x03, 0x80, 0x01, 0xC0, 0x06, 0xC0, 0x03, 0x60,
  0x04, 0x60, 0x06, 0x20, 0x04, 0x30, 0x0C, 0x20,
  0x04, 0x18, 0x18, 0x20, 0x04, 0x0C, 0x30, 0x20,
  0x04, 0x06, 0x60, 0x20, 0x44, 0x03, 0xC0, 0x22,
  0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
  0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
  0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
  0x66, 0x01, 0x80, 0x66, 0x33, 0x01, 0x80, 0xCC,
  0x19, 0x81, 0x81, 0x98, 0x0C, 0xC1, 0x83, 0x30,
  0x07, 0xe1, 0x87, 0xe0, 0x03, 0x3f, 0xfc, 0xc0,
  0x03, 0x31, 0x8c, 0xc0, 0x03, 0x33, 0xcc, 0xc0,
  0x06, 0x64, 0x26, 0x60, 0x0c, 0xcc, 0x33, 0x30,
  0x18, 0xcc, 0x33, 0x18, 0x10, 0xc4, 0x23, 0x08,
  0x10, 0x63, 0xC6, 0x08, 0x10, 0x30, 0x0c, 0x08,
  0x10, 0x18, 0x18, 0x08, 0x10, 0x00, 0x00, 0x08};
```

Primer: "pomeranje" pčelice

```
GLubyte halftone[] = {
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55};

float k=0;
```

Primer: "pomeranje" pčelice

```
void display(void){
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(1.0, 1.0, 1.0);
 glRectf(25.0, 25.0, 125.0, 125.0);
 glEnable(GL_POLYGON_STIPPLE);
   glPolygonStipple(fly);
   glPushMatrix();
     glTranslatef(1+k,1+k,0);
     glRectf (128.0, 32.0, 32.0+128.0, 32.0+32.0);
   glPopMatrix();
   glPolygonStipple(halftone);
   glRectf(225.0, 25.0, 325.0, 125.0);
   glColor3f(0.0, 0.0,1.0);
   glPolygonStipple(halftone);
   glRectf(25.0, 125.0, 325.0, 225.0);
 glDisable(GL_POLYGON_STIPPLE);
 k++;
 glFlush();
}
void Timer( int iValue ){
  glutPostRedisplay();   glutTimerFunc( 10, Timer, iValue );
}
```

Primer: "pomeranje" pčelice

```
void init (void){
   glClearColor(0.0, 0.0, 0.0, 0.0);   glShadeModel(GL_FLAT);
}

void reshape(int w, int h){
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);
}

int main(int argc, char** argv){
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(350, 250);
   glutCreateWindow(argv[0]);
   init();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutTimerFunc( 10, Timer, 10 );
   glutMainLoop(); return 0;
}
```

Izlaz