

Računarska grafika
(2OER7O02)

OpenGL – Teksture

Auditivne vežbe




Tipovi tekstura

- **Jednodimenzionalne** (1D) teksture su jednodimenzionalna polja teksela. Ove teksture imaju samo jednu koordinatu, koja se najčešće označava slovom *s*.
- **Dvodimenzionalne** (2D) teksture su matrice teksela. To su slike koje imaju dužinu i visinu, i poput tapeta se „lepe“ na površinu objekata. Ove teksture imaju dve koordinate, koje se označavaju slovima *s* i *t*.
- **Trodimenzionalne** (3D) teksture su polja matrica teksela. Osim širine i visine, ove teksture imaju i dubinu.
- **Kubne teksture** sastoje se od šest dvodimenzionalnih tekstura. Mapiranje kubne teksture vrši se tako što se oko datog objekta projektuje opisana kocka, čije su strane poravnate sa koordinatnim osama. Svaka od šest 2D tekstura pripada jednoj stranici kocke.
- Počev od verzije 1.3, OpenGL-a podržava i **višestruko teksturisanje**. To znači da se izgled površine objekta može definisati kao kumulativni efekat različitih tekstura, pri čemu koordinate temena u okviru različitih tekstura mogu biti različite.

Klasa CGLTexture

- **PrepareTexturing()** – globalno podešavanje teksturisanja (nevezano za pojedine objekte),
- **LoadFromFile()** – kreiranje teksture na osnovu datoteke, tj. slike u BMP formatu,
- **Select()** – aktiviranje date teksture i
- **Release()** – oslobađanje resursa koje zauzima tekstura u memoriji grafičke kartice.

CGLTexture	
	m_ID : unsigned int
<ul style="list-style-type: none">◆ CTexture()◆ ~CTexture()◆ LoadFromFile()◆ Select()◆ Release()◆ <<static>> PrepareTexturing()	

Primena tekstura

- Funkcija **glPixelStorei()** definiše način na koji je bitmapa (slika) smeštena u memoriji.

```
glPixelStorei(GL_UNPACK_ALIGNMENT, 4);
```

- Funkcija kojom se definiše način primene tekture, tj. kako se vrednost pročitana iz tekture meša sa bojom fragmenta je:

- ▶ `void glTexEnv{if}(GLenum target, GLenum pname, GLint param);`

glTexEnv

```
void glTexEnv{if}( GLenum target, GLenum pname, GLint param );
```

- *target* mora biti postavljen na **GL_TEXTURE_ENV**,
- *pname* na **GL_TEXTURE_ENV_MODE**,
- *param* može imati jednu od sledećih vrednosti:
 - ▷ **GL_REPLACE** – potpuno menja boju fragmenta podacima iz teksture,
 - ▷ **GL_DECAL** – mešanje boje fragmenta i teksture definisano je providnošću teksture,
 - ▷ **GL_MODULATE** – boja fragmenata je „modulisana“ teksturom i
 - ▷ **GL_BLEND** – boja se dobija mešanjem boje fragmenta, teksture i posebno zadate boje.

Pravila za primenu teksture

Mod	RGB	RGBA
GL_REPLACE	$C = C_t$ $A = A_f$	$C = C_t$ $A = A_t$
GL_DECAL	$C = C_t$ $A = A_f$	$C = C_f(1 - A_t) + C_t A_t$ $A = A_f$
GL_MODULATE	$C = C_f C_t$ $A = A_f$	$C = C_f C_t$ $A = A_f A_t$
GL_BLEND	$C = C_f(1 - C_t) + C_c C_t$ $A = A_f$	$C = C_f(1 - C_t) + C_c C_t$ $A = A_f A_t$

Ako indeksom **t** označimo boju koja potiče od teksture, indeksom **f** boju fragmenta, a indeksom **c** boju definisanu parametrom GL_TEXTURE_ENV_COLOR. Slovo **C** označava bilo koju od tri komponente boje (R, G ili B), a slovo **A** providnost (*alpha*). Kolona RGB odnosi se na slučaj kada tekstura nema providnost (tj. definisana je kao RGB), dok kolona RGBA definiše slučaj kada tekstura ima i providnost.

PrepareTexturing

```
void CGLTexture::PrepareTexturing(bool bEnableLighting)
{
    glPixelStorei(GL_UNPACK_ALIGNMENT, 4);

    if(bEnableLighting)
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
                  GL_MODULATE);
    else
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
                  GL_REPLACE);
}
```

Kreiranje objekata tekstura

Pribavljanje jedinstvenog identifikatora (vraća n trenutno slobodnih identifikatora, u obliku neoznačenih celih brojeva, koje smešta u polje *textureNames*)

▷ `void glGenTextures(GLsizei n, GLuint *textureNames);`

Kreiranje i aktiviranje objekta tekstura

▷ `void glBindTexture(GLenum target, GLuint textureName);`

▷ *target* određuje tip teksture (GL_TEXTURE_1D, **GL_TEXTURE_2D**, GL_TEXTURE_3D ili GL_TEXTURE_CUBE_MAP)

▷ *textureName* je identifikator teksture, koji je vratila funkcija **glGenTextures()**

glBindTexture()

- Kada je prvi put pozvana za dati identifikator, ona kreira objekat tekstura u memoriji grafičke kartice, i selektuje dati objekat. Sve kasnije modifikacije parametara odnosiće se na selektovani objekat.
- Kada se pozove za prethodno kreirani objekat, onda samo selektuje (aktivira) dati objekat.
- A kada se pozove za vrednost 0, isključuje rad sa objektima tekstura i aktivira neimenovanu „podrazumevanu“ teksturu.

```
unsigned int m_ID;  
glGenTextures(1, &m_ID);  
glBindTexture(GL_TEXTURE_2D, m_ID);
```

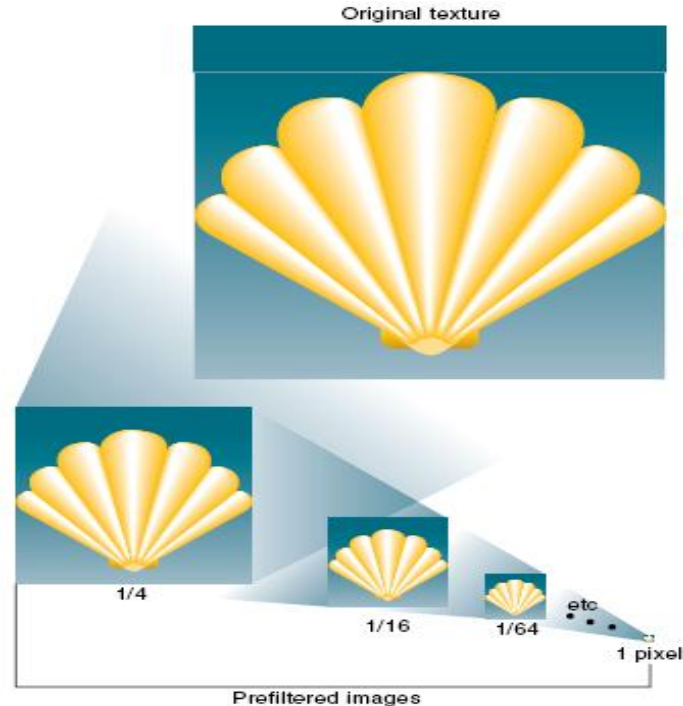
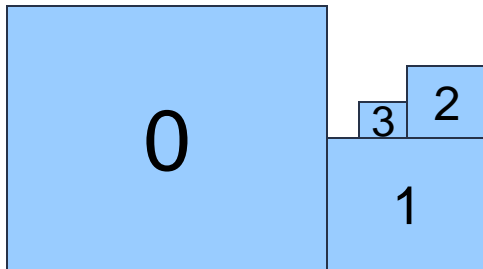
Prebacivanje podataka u teksturu

void **glTexImage2D**(GLenum target, GLint level, GLint internalFormat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *texels);

- *target* – tip teksture (GL_TEXTURE_2D),
- *level* – nivo teksture koji se popunjava,
- *internalFormat* – broj komponenata jednog teksela (1, 2, 3, 4 ili jednu od 55 drugih predefinisanih vrednosti), najčešće koristimo GL_RGB (3) ili GL_RGBA (4)
- *width* – širina u tekselima (mora biti stepen broja 2),
- *height* – visina u tekselima (mora biti stepen broja 2),
- *border* – širina okvira (čitava slika je dimenzija $(2m+border) \times (2n+border)$),
- *format* – format teksela (GL_RGB, GL_RGBA, ...),
- *type* – tip podataka koji formiraju texel (GL_UNSIGNED_BYTE, ...),
- *texels* – pokazivač na bafer sa podacima, koje treba preneti u teksturu.

Mipmaps

Termin *mipmap* iskovao je Lance Williams 1983. u svom radu "Pyramidal Parametrics" (SIGGRAPH 1983 Proceedings), kao skraćenicu latinske fraze *multum in parvo*, (mnoštvo na jednom mestu).



Mipmaps

```
glTexImage2D( GL_TEXTURE_2D, 0, 3, TextureImage->sizeX, TextureImage->sizeY,  
              0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage->data0 );  
glTexImage2D( GL_TEXTURE_2D, 1, 3, TextureImage->sizeX/2, TextureImage->sizeY/2,  
              0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage->data1 );  
glTexImage2D( GL_TEXTURE_2D, 2, 3, TextureImage->sizeX/4, TextureImage->sizeY/4,  
              0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage->data2 );  
glTexImage2D( GL_TEXTURE_2D, 3, 3, TextureImage->sizeX/8, TextureImage->sizeY/8,  
              0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage->data3 );
```



Automatsko generisanje mipmapa

```
int gluBuild2DMipmaps(GLenum target, GLint internalFormat,  
                     GLint width, GLint height,  
                     GLenum format, GLenum type, void *texels);
```

- parametri su isti kao kod `glTexImage2D()`, osim što se ne navodi nivo (jer formira sve nivoe), i border
- čak i kada dimenzije nisu stepen dvojke, ova funkcija normalizuje dimenzije bitmape

Učitavanje tekstone - AUX

■ Uključiti **glaux** biblioteku

```
#include <GL\glaux.h>  
#pragma comment (lib, "glaux.lib")
```

■ Učitati sliku

```
AUX_RGBImageRec* auxDIBImageLoad(LPCWSTR fileName);
```

■ Struktura **AUX_RGBImageRec** ima tri atributa:

- ▷ **data** – bafer u kome je smeštena sadržina slike (pikseli),
- ▷ **sizeX** – širina slike i
- ▷ **sizeY** – visina slike.

Primer učitavanja slike

```
void CGLTexture::LoadFromFile(CString texFileName)
{
    if(m_ID != 0) Release();

    // Alokacija ID-a i kreiranje teksture
    glGenTextures(1, &m_ID);
    glBindTexture(GL_TEXTURE_2D, m_ID);

    // Ucitavanje bitmape
    AUX_RGBImageRec *TextureImage;
    TextureImage = auxDIBImageLoad(texFileName);

    // Kopiranje sadržaja bitmape u teksturu
    glTexImage2D(GL_TEXTURE_2D, 0, 3,
                 TextureImage->sizeX, TextureImage->sizeY,
                 0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage->data);

    // Brisanje bitmape
    if (TextureImage) // ako je učitana slika
    {
        if (TextureImage->data) // i ako sadrži podatke
        {
            free(TextureImage->data); // obrisati podatke
        }
        free(TextureImage); // obrisati samu strukturu
    }
}
```

Učitavanje tekstura - DImage

```
UINT LoadTexture(char* fileName)
{
    UINT texID;
    DImage img;
    img.Load(CString(fileName));
    glPixelStorei(GL_UNPACK_ALIGNMENT, 4);
    glGenTextures(1, &texID);
    glBindTexture(GL_TEXTURE_2D, texID);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                    GL_LINEAR_MIPMAP_LINEAR);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

    gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGBA, img.Width(), img.Height(),
                     GL_BGRA_EXT, GL_UNSIGNED_BYTE, img.GetDIBBits());

    /*glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, img.Width(), img.Height(), 0,
                 GL_BGRA_EXT, GL_UNSIGNED_BYTE, img.GetDIBBits());*/
    return texID;
}
```


Parametri tekstura

Postoji mnoštvo parametara tekstura koji se mogu postaviti funkcijom:

- ▷ `void glTexParameteri(GLenum target, GLenum pname, TYPE param);`
 - ▷ *target* - GL_TEXTURE_2D,
 - ▷ *pname* - naziv parametra,
 - ▷ *param* - njegova vrednost.

Četiri najvažnija parametra teksture su:

- ▷ filter koji se koristi pri uvećanju,
- ▷ filter koji se koristi pri umanjenju,
- ▷ način ponavljanja po S pravcu i
- ▷ način ponavljanja po T pravcu.

Filtriranje

Prilikom formiranja konačne slike, jedan teksel može uticati na formiranje više piksela, ili se više teksela može preslikati u samo jedan piksel. Moguće je i da obe pojave nastanu istovremeno na jednoj teksturi, ukoliko je objekat dugačak, i orijentisan u pravcu pogleda.

Prvi slučaj, kada jedan teksel utiče na formiranje više piksela, naziva se **uvećanje** teksture. Kako će se tekstura ponašati u takvoj situaciji definišemo pozivom funkcije:

▶ `glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, param);`

▶ Parametar *param* može imati jednu od sledeće dve vrednosti:

▶ `GL_NEAREST`, ili

▶ `GL_LINEAR`.

Uvećanje - Magnification



GL_NEAREST



GL_LINEAR

Filtriranje

Kada je objekat koji posmatramo daleko, više teksela stapaju se u samo jedan piksel. Tada kažemo da nastaje **umanjenje** teksture. Kako će se tekstura ponašati pri umanjenju definišemo pozivom funkcije:

- ▷ `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, param);`
 - ▷ Parametar *param* može imati jednu od sledećih vrednosti:
 - ▷ `GL_NEAREST,`
 - ▷ `GL_LINEAR,`
 - ▷ `GL_NEAREST_MIPMAP_NEAREST,`
 - ▷ `GL_NEAREST_MIPMAP_LINEAR,`
 - ▷ `GL_LINEAR_MIPMAP_NEAREST,` ili
 - ▷ `GL_LINEAR_MIPMAP_LINEAR.`

Filtriranje

Efekti vrednosti **GL_NEAREST** i **GL_LINEAR** su identični kao kod filtera uvećanja. Ostale vrednosti moguće su samo ukoliko postoje mipmape. Ako mipmapa nije formirana kompletno (nedostaju neki nivoi), ili korektno (neki od nivoa nije zadat kako treba), a koristi se neki od **GL_*_MIPMAP_*** filtera, OpenGL automatski isključuje teksturisane.

Vrednost **GL_NEAREST_MIPMAP_NEAREST** omogućuje da za boju piksela uzme texel najbliži centru piksela iz odgovarajućeg nivoa mipmape (nivo koji sadrži odgovarajući texel, koji je po veličini najbliži datom pikselu).

Vrednost **GL_LINEAR_MIPMAP_NEAREST** omogućuje da se boja piksela dobija linearnim usrednjavanjem polja 2×2 texela iz nivoa mipmape koji po veličini texela najviše odgovara datom pikselu.

Vrednost **GL_NEAREST_MIPMAP_LINEAR** vrši linearno usrednjavanje boja dva texela, čiji su centri najbliži centru piksela, a uzimaju se iz dva susedna nivoa mipmape sa odgovarajućim veličinama texela.

Vrednost **GL_LINEAR_MIPMAP_LINEAR** vrši linearno usrednjavanje boja dve grupe od po 2×2 texela, iz dva susedna nivoa mipmape. Ovo je najsloženiji tip filtriranja i daje najmekši prikaz.

Umanjenje - Minification



GL_NEAREST



GL_LINEAR



GL_NEAREST_MIPMAP_NEAREST



GL_NEAREST_MIPMAP_LINEAR



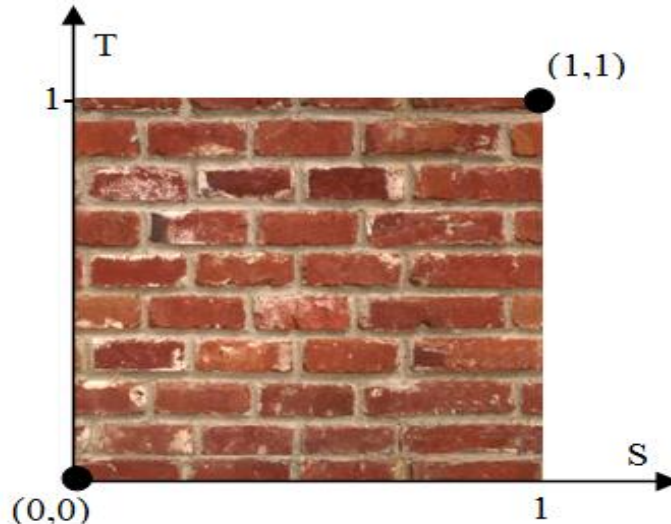
GL_LINEAR_MIPMAP_NEAREST



GL_LINEAR_MIPMAP_LINEAR

Ponavljanje

- Teksture koje se primenjuju na neke objekte mogu biti takve strukture da omogućuju formiranje šablona, koji se zatim mogu ponavljati više puta na površini samih objekata.

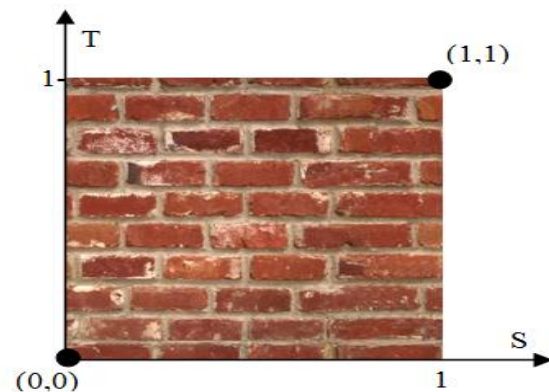


Ponavljjanje

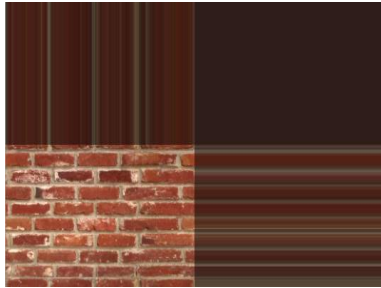
Način ponavljanja teksture, prilikom primene na odgovarajući objekat, može se definisati sledećim programskim segmentom:

- ▶ `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, repParam);`
- ▶ `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, repParam);`

Parametar *repParam* određuje da li se vrši ponavljanje teksture po datom pravcu (S ili T), ili ne. S i T su koordinatne ose teksture koje odgovaraju X i Y osama



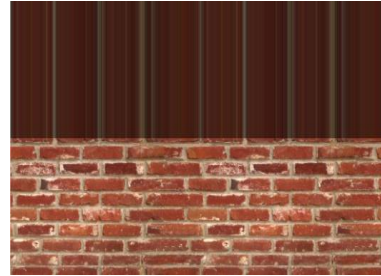
Ponavljjanje



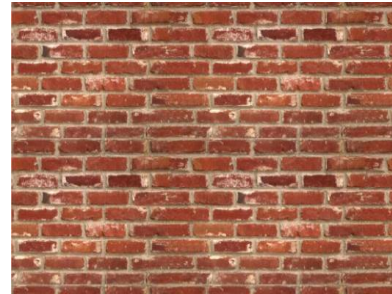
GL_TEXTURE_WRAP_S = GL_CLAMP
GL_TEXTURE_WRAP_T = GL_CLAMP



GL_TEXTURE_WRAP_S = GL_CLAMP
GL_TEXTURE_WRAP_T = GL_REPEAT



GL_TEXTURE_WRAP_S = GL_REPEAT
GL_TEXTURE_WRAP_T = GL_CLAMP

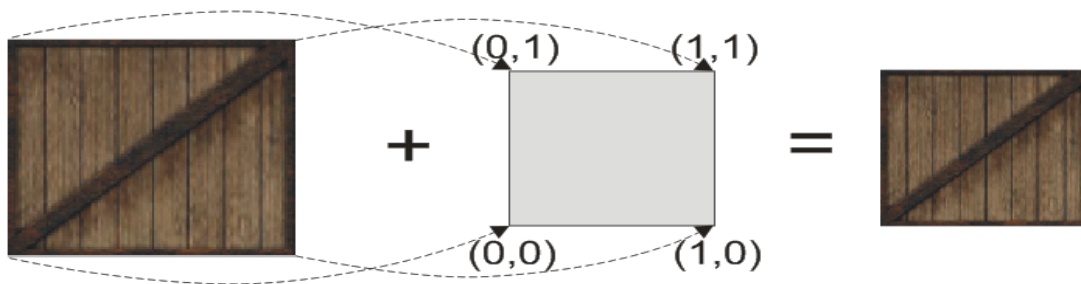


GL_TEXTURE_WRAP_S = GL_REPEAT
GL_TEXTURE_WRAP_T = GL_REPEAT

Teksturne koordinate temena

- Koordinate temena u okviru teksture zadaju se pozivom funkcije:

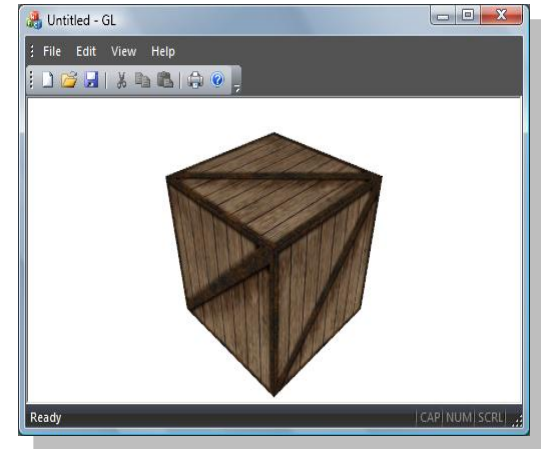
- ▷ **glTexCoord2f(coordS, coordT);**
- ▷ *coordS* teksturna koordinata po S-pravcu,



Primer teksturisanja kocke

```
void CGLRenderer::DrawCube(double a)
{
    glBegin(GL_QUADS);

    // Prednja stranica
    glNormal3d( 0.0, 0.0, 1.0);
    glTexCoord2f(0.0, 1.0);
    glVertex3d(-a/2, a/2, a/2);
    glTexCoord2f(0.0, 0.0);
    glVertex3d(-a/2,-a/2, a/2);
    glTexCoord2f(1.0, 0.0);
    glVertex3d( a/2,-a/2, a/2);
    glTexCoord2f(1.0, 1.0);
    glVertex3d( a/2, a/2, a/2);
    // Ostale stranice ...
    glEnd();
}
```



Definisanje polja temena

```
// Temena
int ind = 0;
vert[ind++]= -a/2; vert[ind++]= -a/2; vert[ind++]= a/2; ind+=5; // vert0
vert[ind++]= a/2; vert[ind++]= -a/2; vert[ind++]= a/2; ind+=5; // vert1
vert[ind++]= a/2; vert[ind++]= a/2; vert[ind++]= a/2; ind+=5; // vert2
vert[ind++]= -a/2; vert[ind++]= a/2; vert[ind++]= a/2; ind+=5; // vert3

vert[ind++]= a/2; vert[ind++]= -a/2; vert[ind++]= a/2; ind+=5; // vert1
vert[ind++]= a/2; vert[ind++]= -a/2; vert[ind++]= -a/2; ind+=5; // vert5
vert[ind++]= a/2; vert[ind++]= a/2; vert[ind++]= -a/2; ind+=5; // vert6
vert[ind++]= a/2; vert[ind++]= a/2; vert[ind++]= a/2; ind+=5; // vert2

vert[ind++]= -a/2; vert[ind++]= a/2; vert[ind++]= -a/2; ind+=5; // vert7
vert[ind++]= a/2; vert[ind++]= a/2; vert[ind++]= -a/2; ind+=5; // vert6
vert[ind++]= a/2; vert[ind++]= -a/2; vert[ind++]= -a/2; ind+=5; // vert5
vert[ind++]= -a/2; vert[ind++]= -a/2; vert[ind++]= -a/2; ind+=5; // vert4

vert[ind++]= -a/2; vert[ind++]= -a/2; vert[ind++]= a/2; ind+=5; // vert0
vert[ind++]= -a/2; vert[ind++]= a/2; vert[ind++]= a/2; ind+=5; // vert3
vert[ind++]= -a/2; vert[ind++]= a/2; vert[ind++]= -a/2; ind+=5; // vert7
vert[ind++]= -a/2; vert[ind++]= -a/2; vert[ind++]= -a/2; ind+=5; // vert4

vert[ind++]= -a/2; vert[ind++]= a/2; vert[ind++]= -a/2; ind+=5; // vert7
vert[ind++]= -a/2; vert[ind++]= a/2; vert[ind++]= a/2; ind+=5; // vert3
vert[ind++]= a/2; vert[ind++]= a/2; vert[ind++]= a/2; ind+=5; // vert2
vert[ind++]= a/2; vert[ind++]= a/2; vert[ind++]= -a/2; ind+=5; // vert6

vert[ind++]= -a/2; vert[ind++]= -a/2; vert[ind++]= a/2; ind+=5; // vert0
vert[ind++]= -a/2; vert[ind++]= -a/2; vert[ind++]= -a/2; ind+=5; // vert4
vert[ind++]= a/2; vert[ind++]= -a/2; vert[ind++]= -a/2; ind+=5; // vert5
vert[ind++]= a/2; vert[ind++]= -a/2; vert[ind++]= a/2; ind+=5; // vert1
```

Definisanje polja temena

```
// Normale
ind = 3;
for(int i=0; i<4; i++)
{
    vert[ind++]= 0; vert[ind++]= 0; vert[ind++]= 1; // prednja
    ind+=5;
}
for(int i=0; i<4; i++)
{
    vert[ind++]= 1; vert[ind++]= 0; vert[ind++]= 0; // desna
    ind+=5;
}
for(int i=0; i<4; i++)
{
    vert[ind++]= 0; vert[ind++]= 0; vert[ind++]= -1; // zadnja
    ind+=5;
}
for(int i=0; i<4; i++)
{
    vert[ind++] = -1; vert[ind++] = 0; vert[ind++] = 0; // leva
    ind+=5;
}
for(int i=0; i<4; i++)
{
    vert[ind++] = 0; vert[ind++] = 1; vert[ind++] = 0; // gornja
    ind+=5;
}
for(int i=0; i<4; i++)
{
    vert[ind++] = 0; vert[ind++] = 1; vert[ind++] = 0; // donja
    ind+=5;
}
```

Definisanje polja temena

x	y	z	n_x	n_y	n_z	s	t
x	y	z	n_x	n_y	n_z	s	t

```
// Tex.kord.  
ind = 6;  
for(int i=0; i<6; i++)  
{  
    vert[ind++]= 0; vert[ind++]= 0; // donja leva  
    ind+=6;  
    vert[ind++]= 1; vert[ind++]= 0; // donja desna  
    ind+=6;  
    vert[ind++]= 1; vert[ind++]= 1; // gornja desna  
    ind+=6;  
    vert[ind++]= 0; vert[ind++]= 1; // donja leva  
    ind+=6;  
}
```

Crtanje preko polja temena

x	y	z	n_x	n_y	n_z	s	t
x	y	z	n_x	n_y	n_z	s	t

```
void CGLRenderer::DrawVACube3()
{
    glVertexPointer(3, GL_FLOAT, 8*sizeof(float), &vert[0]);
    glNormalPointer(GL_FLOAT, 8*sizeof(float), &vert[3]);
    glTexCoordPointer(2, GL_FLOAT, 8*sizeof(float), &vert[6]);

    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_NORMAL_ARRAY);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);

    glDrawArrays(GL_QUADS, 0, 24);

    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_NORMAL_ARRAY);
    glDisableClientState(GL_TEXTURE_COORD_ARRAY);
}
```

Aktiviranje tekstura

■ Inicijalno, teksturisanje je isključeno (isto kao i osvetljenje). Uključivanje 2D teksturisanja ostvaruje se pozivom funkcije:

▷ **glEnable(GL_TEXTURE_2D);**

■ Osim globalnog uključivanja tekstura (pozivom funkcije **glEnable()**), potrebno je aktivirati i odgovarajuću teksturu pre crtanja objekta koji je koristi. To se ostvaruje pozivom funkcije:

▷ **glBindTexture(GL_TEXTURE_2D, ID);**

▷ *ID* identifikator date teksture.

Brisanje tekstura

- Kada prestane potreba za nekom teksturom, potrebno je datu teksturu obrisati
 - ▷ `void glDeleteTextures(GLsizei n, const GLuint *textureNames);`
 - ▷ *n* - koliko se tekstura briše,
 - ▷ *textureNames* - polje identifikatora tih tekstura
- Ako se pozove brisanje teksture koja je trenutno selektovana, aktivira se neimenovana podrazumevana tekstura, čiji je identifikator 0.
- Ako se pokuša brisanje nepostojeće teksture ili teksture sa identifikatorom 0, odgovarajući poziv funkcije **glDeleteTextures()** biće ignorisan.