

Računarska grafika
(20ER7002)

Uvod u OpenGL

Auditivne vežbe



Šta je OpenGL?

OpenGL je softverski interfejs prema grafičkom hardveru.

Interfejs (u verziji 4.5) čini preko 500 komandi omogućujući specificiranje objekata i operacija za kreiranje interaktivnih 3D aplikacija.

Započet razvoj: 1992.

Verzija 1.1 (1996.god., Impl.1997.god.)

Tekuća verzija: 4.6 (2017.god.)







Istorijat

Verzija	Datum objavljivanja	Važna novina
OpenGL 1.0	1. jul 1992.	Iris GL
OpenGL 1.1	1996/97.	<i>Vertex Array, Texture Objects, ...</i>
OpenGL 1.2	16. mart 1998.	<i>3D Texturing, BGRA Pixel Formats, ...</i>
OpenGL 1.2.1	14. oktobar 1998.	Uvode se ARB ekstenzije i <i>multitexture</i>
OpenGL 1.3	14. avgust 2001.	<i>Compressed Textures, Cube Map Textures, Multisample, Multitexture ...</i>
OpenGL 1.4	24. jul 2002.	<i>Depth Textures and Shadows, Fog Coordinate, Multiple Draw Arrays, ...</i> usvojena nova ekstenzija ARB vertex program (najava novog doba)
OpenGL 1.5	29. jul 2003.	<i>Buffer Objects, Shadow Functions, ...</i> , i nove ekstenzije ARB shader objects , ARB vertex shader i ARB fragment shader
OpenGL 2.0	7. septembar 2004.	OpenGL Shading Language (verzije 1.10 i novije), teksture ne moraju biti dimenzija 2^N , <i>Point Sprites,...</i>

Šta je potrebno da bi se koristio OpenGL pod Windowsom?

- skup biblioteka (***opengl32.lib***, ***glu32.lib***, a poželjno je i ***glaux.lib*** i ***glut.lib***),
- zaglavlja (***gl.h***, ***glu.h***, ***glaux.h***, ***glut.h***),
- *run-time* biblioteka (***OpenGL32.dll***) – mora biti iste verzije kao i biblioteke, ili novija, i
- klijentski drajver (***Installable Client Driver - IDC***) – dolazi uz drajver grafičke kartice i omogućuje hardversku akceleraciju i dodatne funkcije

Priprema Visual C++ projekta za rad sa OpenGL-om

CGLRenderer
 <code>m_hrc : HGLRC</code>
 <code>CreateGLContext(CDC* pDC) : bool</code>
 <code>PrepareScene(CDC* pDC) : void</code>
 <code>Reshape(CDC* pDC, int w, int h) : void</code>
 <code>DrawScene(CDC* pDC) : void</code>
 <code>DestroyScene(CDC* pDC) : void</code>

Sve funkcije vezane za OpenGL grupisaćemo u jednu klasu, nazvanu **CGLRenderer**:

■ atributi

- ▷ HGLRC **m_hrc** – *OpenGL Rendering Context* (OpenGL server koji vrši iscrtavanje)

■ metode

- ▷ *bool* **CreateGLContext**(CDC* pDC) – kreira *OpenGL Rendering Context*,
- ▷ *void* **PrepareScene**(CDC* pDC) – inicijalizuje scenu,
- ▷ *void* **DestroyScene**(CDC* pDC) – oslobađa resurse alocirane u drugim funkcijama ove klase,
- ▷ *void* **Reshape**(CDC* pDC, int w, int h) – kod koji treba da se izvrši svaki put kada se promeni veličina prozora ili pogleda i
- ▷ *void* **DrawScene**(CDC* pDC) – iscrtava scenu

bool CGLRenderer::CreateGLContext (CDC* pDC)

```
bool CGLRenderer::CreateGLContext(CDC* pDC)
{
    PIXELFORMATDESCRIPTOR pfd ;
    memset(&pfd, 0, sizeof(PIXELFORMATDESCRIPTOR));
    pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);
    pfd.nVersion = 1;
    pfd.dwFlags = PFD_DOUBLEBUFFER | PFD_SUPPORT_OPENGL | PFD_DRAW_TO_WINDOW;
    pfd.iPixelFormat = PFD_TYPE_RGBA;
    pfd.cColorBits = 32;
    pfd.cDepthBits = 32;
    pfd.iLayerType = PFD_MAIN_PLANE;

    int nPixelFormat = ChoosePixelFormat(pDC->m_hDC, &pfd);

    if (nPixelFormat == 0) return false;

    BOOL bResult = SetPixelFormat (pDC->m_hDC, nPixelFormat, &pfd);

    if (!bResult) return false;

    m_hrc = wglCreateContext(pDC->m_hDC);

    if (!m_hrc) return false;

    return true;
}
```

Korišćene funkcije

int ChoosePixelFormat(HDC hdc, CONST PIXELFORMATDESCRIPTOR * ppfd)	
<p><i>hdc</i> – handle DC-a u kome se traži format piksela najpribližniji zadatom <i>ppfd</i> – pokazivač na strukturu koja opisuje željeni format piksela Funkcija vraća indeks formata piksela najpribližniji zatatom formatu. Ukoliko ne uspe, funkcija vraća 0.</p>	

BOOL SetPixelFormat(HDC hdc, int iPixelFormat, CONST PIXELFORMATDESCRIPTOR * ppfd)	
<p><i>hdc</i> – handle DC-a čiji se format piksela postavlja <i>iPixelFormat</i> – indeks koji određuje format piksela koji se postavlja <i>ppfd</i> – pokazivač na strukturu koja opisuje format piksela (uglavnom nema nikakvu ulogu pri ovom pozivu) Funkcija vraća TRUE, ukoliko postavljanje formata piksela uspe. U protivnom, vraća FALSE.</p>	

Korišćene funkcije

HGLRC wglCreateContext(HDC hdc)

hdc – *handle* DC-a za koji se kreira odgovarajući OpenGL *rendering context*
Funkcija kreira OpenGL *rendering context* pogodan za iscrtavanje na uređaju čiji je DC prosleđen kao parametar. OpenGL *rendering context* ima isti format piksela kao i prosleđeni DC. Ako funkcija uspe, vraća *handle* kreiranog OpenGL *rendering context*-a . U protivnom, vraća NULL.

BOOL wglDeleteContext(HGLRC hglrc)

hglrc - *handle* OpenGL *rendering context* koji funkcija briše
Funkcija briše OpenGL *rendering context* čiji je *handle* prosleđen. Ako uspe, funkcija vraća TRUE. U suprotnom vraća FALSE.

bool CGLRenderer::PrepareScene (CDC* pDC)

```
void CGLRenderer::PrepareScene (CDC *pDC)
{
    wglMakeCurrent (pDC->m_hDC, m_hrc);
    //-----
    glClearColor (1.0, 1.0, 1.0, 0.0);
    //-----
    wglMakeCurrent (NULL, NULL);
}
```

bool CGLRenderer::Reshape (CDC* pDC)

```
void CGLRenderer::Reshape (CDC *pDC, int w, int h)
{
    wglMakeCurrent (pDC->m_hDC, m_hrc);
    //-----
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluPerspective(40, (double)w/(double)h, 1, 100);
    glMatrixMode (GL_MODELVIEW);
    //-----
    wglMakeCurrent (NULL, NULL);
}
```

bool CGLRenderer::DrawScene (CDC* pDC)

```
void CGLRenderer::DrawScene (CDC *pDC)
{
    wglMakeCurrent (pDC->m_hDC, m_hrc);
    //-----
    glClear (GL_COLOR_BUFFER_BIT);
    glLoadIdentity ();

    glBegin(GL_TRIANGLES);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(-1.0, -1.0, -5.0);
        glColor3f(0.0, 1.0, 0.0);
        glVertex3f(1.0, -1.0, -5.0);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(1.0, 1.0, -5.0);
    glEnd();

    glFlush ();
    //-----
    SwapBuffers (pDC->m_hDC);
    wglMakeCurrent (NULL, NULL);
}
```

bool CGLRenderer::DestroyScene (CDC* pDC)

```
void CGLRenderer::DestroyScene (CDC *pDC)
{
    wglMakeCurrent (pDC->m_hDC, m_hrc);
    // ...
    wglMakeCurrent (NULL, NULL);
    if (m_hrc)
    {
        wglDeleteContext (m_hrc);
        m_hrc = NULL;
    }
}
```

Korišćene funkcije

BOOL wglMakeCurrent(HDC hdc, HGLRC hglrc)

hdc – *handle* DC-a uređaja na kome će OpenGL funkcije koje slede vršiti iscrtavanje
hglrc - *handle* OpenGL *rendering context*-a koji postaje tekući za nit koja je pozvala ovu funkciju

Funkcija postavlja tekući OpenGL *rendering context* za nit koja je pozvala. Ako je ***hglrc*** NULL, funkcija oslobađa DC koji je do tada koristio OpenGL *rendering context*, i nakon toga deselektuje tekući *rendering context*.

BOOL SwapBuffers(HDC hdc)

hdc – *handle* DC-a čiji se baferi menjaju

Funkcija zamenjuje prednji (*front*) i zadnji (*back*) bafer prozora, ako format piksela uključuje *back*-bafer. Funkcija vraća TRUE, ako uspe. U protivnom, vraća FALSE.

Deklaracije i linkovanje

- Deklaracije svih OpenGL funkcija nalaze se u dve datoteke: **gl.h** i **glu.h**.
- Dodati sledeća dva reda na početak **GLRenderer.cpp** datoteke:

```
#include <GL\gl.h>  
#include <GL\glu.h>
```

Uključivanje biblioteka

Programsko linkovanje

Programsko uključivanje statičkih biblioteka ostvaruje se pomoću direktive **#pragma**. Na primer, za dodavanje podrške za OpenGL u datoteci **StdAfx.h** treba dodati sledeće linije koda:

```
#pragma comment (lib, "OpenGL32.lib")  
#pragma comment (lib, "glu32.lib")
```

Ukoliko postoje različite verzije biblioteka (*release* i *debug*) uključivanje se obavlja na sledeći način:

```
#pragma message("Automatsko povezivanje biblioteke")  
#ifndef _DEBUG  
#pragma comment( lib, "lib_release.lib")  
#else  
#pragma comment( lib, "lib_debug.lib")  
#endif
```

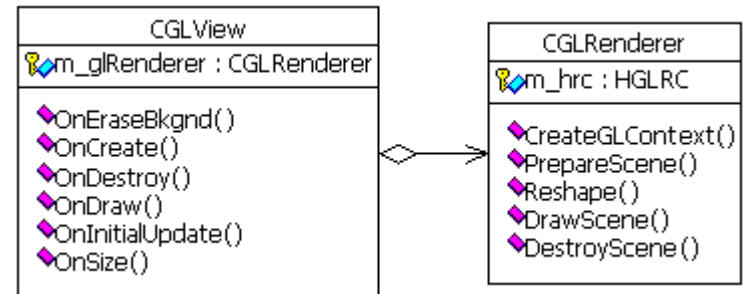
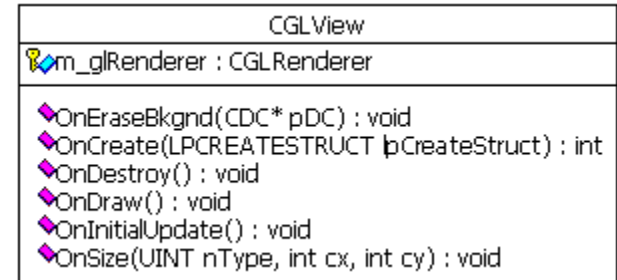
Povezivanjem sa klasom pogleda

U zaglavlju klase CGLView dodati

```
protected:  
    CGLRenderer m_glRenderer;
```

a ispred deklaracije klase CGLView

```
#include "GLRenderer.h"
```



Dodavanje metoda klase pogleda

- Potrebno je dodati rukovaoce za odgovarajuće Windows poruke (*message handlers*), i to:
 - ▷ **WM_CREATE,**
 - ▷ **WM_ERASEBKGND,**
 - ▷ **WM_SIZE** i
 - ▷ **WM_DESTROY,**
- i jednu predefinisanu funkciju okvira
 - ▷ **OnInitialUpdate**

int CGLView::OnCreate(LPCREATESTRUCT lpCreateStruct)

```
int COGLView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    CDC* pDC = GetDC();
    m_glRenderer.CreateGLContext(pDC);
    ReleaseDC(pDC);

    return 0;
}
```

void CGLView::OnSize(UINT nType, int cx, int cy)

```
void CGLView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    CDC* pDC = GetDC();
    m_glRenderer.Reshape(pDC, cx, cy);
    ReleaseDC(pDC);
}
```

void CGLView::OnDestroy()

```
void CGLView::OnDestroy()  
{  
    CView::OnDestroy();  
  
    CDC* pDC = GetDC();  
    m_glRenderer.DestroyScene(pDC);  
    ReleaseDC(pDC);  
  
}
```

void CGLView::OnInitialUpdate()

```
void CGLView::OnInitialUpdate()  
{  
    CView::OnInitialUpdate();  
  
    CDC* pDC = GetDC();  
    m_glRenderer.PrepareScene(pDC);  
    ReleaseDC(pDC);  
}
```

void CGLView::OnDraw(CDC* pDC)

```
void CGLView::OnDraw(CDC* pDC)
{
    CGLDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

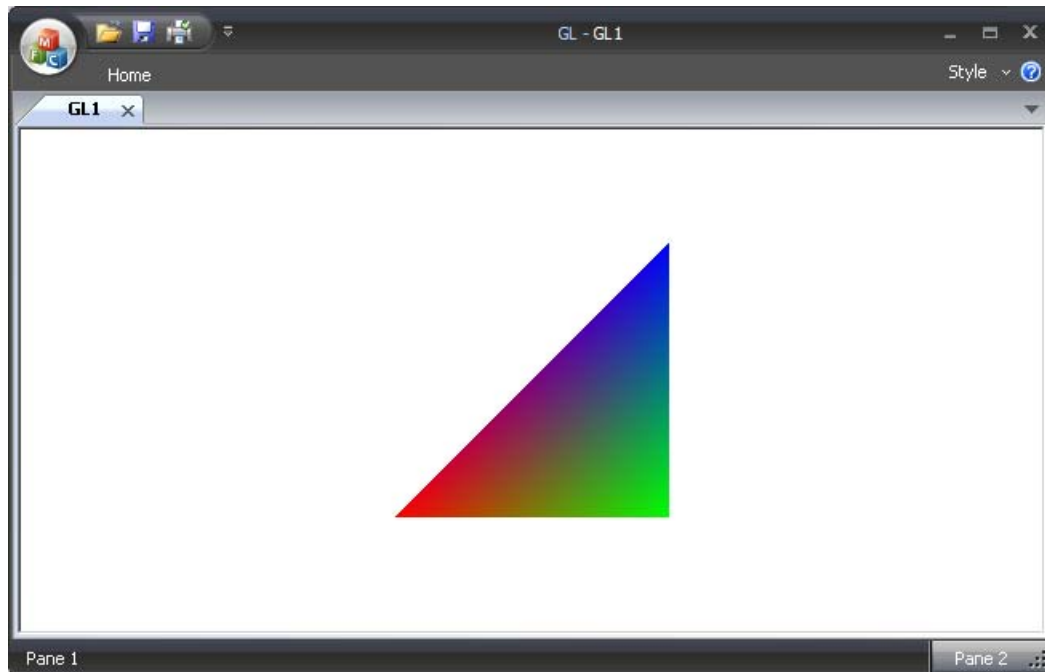
    m_glRenderer.DrawScene(pDC);
}
```

void CGLView::OnEraseBkgnd(CDC* pDC)

```
BOOL CGLView::OnEraseBkgnd(CDC* pDC)
{
    return TRUE;

    //return CView::OnEraseBkgnd(pDC);
}
```

Prva aplikacija



Sintaksa OpenGL-a

Imena svih predefinisanih konstanti podležu sledećim pravilima:

- ispisane su isključivo velikim slovima,
- reči su povezane crticama za podvlačenje i
- imaju prefiks **GL_**.

GL_TRIANGLE_STRIP

Sintaksa OpenGL-a

Imena funkcija sastoje se od:

- prefiksa (**gl** – za funkcije iz jezgra OpenGL-a i **glu** – za funkcije iz *Utility Library*)
- smislenog naziva funkcije, pri čemu svaka reč počinje velikim slovom i
- sufiksa, koji se sastoji od:
 - ▷ broja parametara,
 - ▷ tipa parametara i
 - ▷ indikatora vektora.

```
glColor3f(red, green, blue);
```

```
glColor3fv(vec);
```

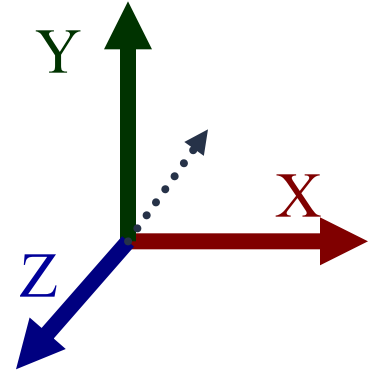
Tip argumenata funkcije

Slovo u sufiksu	Dužina podatka [b]	Tip u jeziku C	Tip u OpenGL-u
b	8	char	GLbyte
s	16	short	GLshort
i	32	int/long	GLint, GLsizei
f	32	float	GLfloat, GLclampf
d	64	double	GLdouble, GLclampd
ub	8	unsigned char	GLubyte, GLboolean
us	16	unsigned short	GLushort
ui	32	unsigned int/long	GLuint, GLenum, GLbitfield

```
void glColor3b( GLbyte red, GLbyte green, GLbyte blue );  
void glColor3d( GLdouble red, GLdouble green, GLdouble blue );  
void glColor3f( GLfloat red, GLfloat green, GLfloat blue );  
void glColor3i( GLint red, GLint green, GLint blue );  
void glColor3s( GLshort red, GLshort green, GLshort blue );  
void glColor4f( GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha );  
void glColor4fv( const GLfloat *v );
```

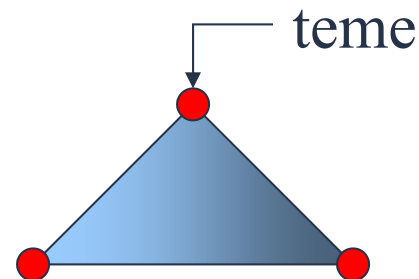
Koordinatni sistem

- OpenGL koristi desni koordinatni sistem
 - ▷ X raste udesno
 - ▷ Y raste naviše
 - ▷ Z raste prema posmatraču
- Inicijalno posmatrač je u koordinatnom početku i gleda u pravcu negativne Z-ose



Temena

- Sve primitive definisane su preko temena
- Svako teme ima svoje prostorne koordinate (primarni atributi)
- Prostorne koordinate definišu se funkcijom **glVertex*()**



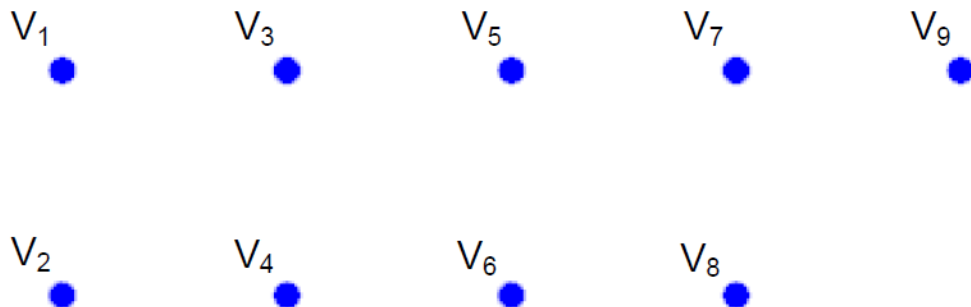
```
glVertex3f(-1.0, -2.0, -5.0);
```

Tipovi primitiva

Parametar	Tip primitive
GL_POINTS	Pojedinačne tačke
GL_LINES	Pojedinačne duži, definisane sa po dve tačke
GL_LINE_STRIP	Serijski međusobno povezanih duži
GL_LINE_LOOP	Serijski međusobno povezanih duži, pri čemu su povezane i prva i poslednja, tako da formiraju petlju
GL_TRIANGLES	Pojedinačni trouglovi, definisani grupama od po 3 temena
GL_TRIANGLE_STRIP	Traka sačinjena od trouglova koji dele po jednu stranicu
GL_TRIANGLE_FAN	Lepeza sačinjena od trouglova
GL_QUADS	Pojedinačni četvorouglovi
GL_QUAD_STRIP	Traka sačinjena od četvorouglova
GL_POLYGON	Jednostavan konveksan poligon (višeugao)

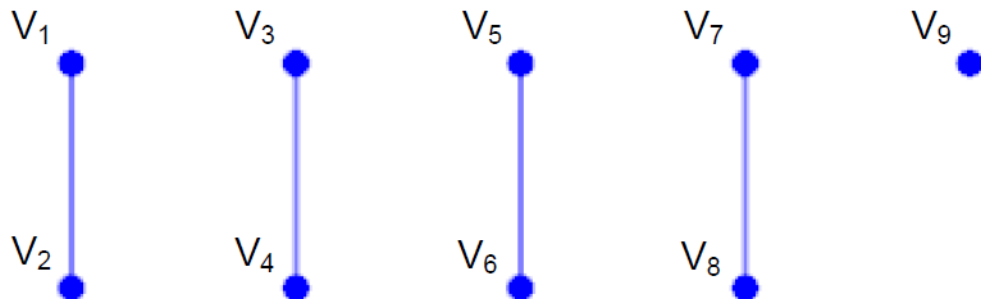
Pojedinačne tačke

```
glLineWidth(2.0);  
glPointSize(10);  
glBegin(GL_POINTS) ;  
    glColor3f(0.5, 0.5, 1.0);  
    glVertex2f(-2.0, 1.0); // V1  
    glVertex2f(-2.0, 0.0); // V2  
    glVertex2f(-1.0, 1.0); // V3  
    glVertex2f(-1.0, 0.0); // V4  
    glVertex2f( 0.0, 1.0); // V5  
    glVertex2f( 0.0, 0.0); // V6  
    glVertex2f( 1.0, 1.0); // V7  
    glVertex2f( 1.0, 0.0); // V8  
    glVertex2f( 2.0, 1.0); // V9  
glEnd() ;
```



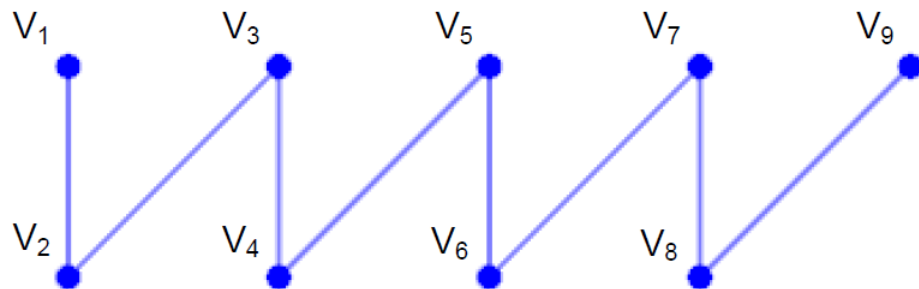
Duži

```
glLineWidth(2.0);  
glPointSize(10);  
glBegin(GL_LINES);  
    glColor3f(0.5, 0.5, 1.0);  
    glVertex2f(-2.0, 1.0); // V1  
    glVertex2f(-2.0, 0.0); // V2  
    glVertex2f(-1.0, 1.0); // V3  
    glVertex2f(-1.0, 0.0); // V4  
    glVertex2f( 0.0, 1.0); // V5  
    glVertex2f( 0.0, 0.0); // V6  
    glVertex2f( 1.0, 1.0); // V7  
    glVertex2f( 1.0, 0.0); // V8  
    glVertex2f( 2.0, 1.0); // V9  
glEnd();
```



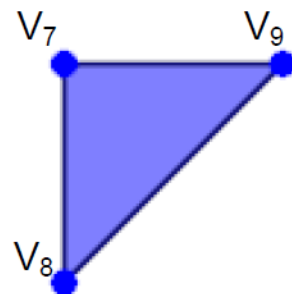
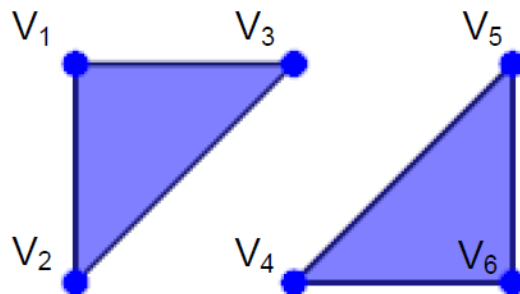
Polilinije

```
glLineWidth(2.0);  
glPointSize(10);  
glBegin(GL_LINE_STRIP) ;  
    glColor3f(0.5, 0.5, 1.0);  
    glVertex2f(-2.0, 1.0); // V1  
    glVertex2f(-2.0, 0.0); // V2  
    glVertex2f(-1.0, 1.0); // V3  
    glVertex2f(-1.0, 0.0); // V4  
    glVertex2f( 0.0, 1.0); // V5  
    glVertex2f( 0.0, 0.0); // V6  
    glVertex2f( 1.0, 1.0); // V7  
    glVertex2f( 1.0, 0.0); // V8  
    glVertex2f( 2.0, 1.0); // V9  
glEnd() ;
```



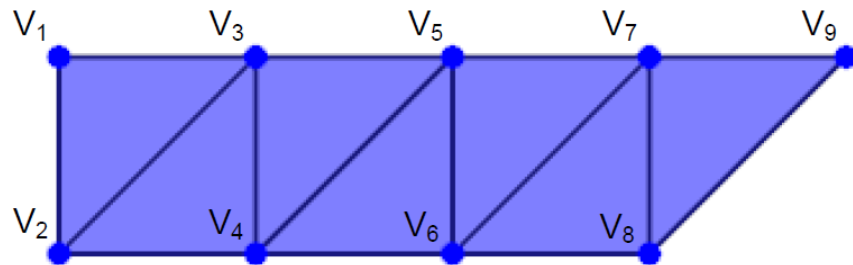
Trouglovi

```
glLineWidth(2.0);  
glPointSize(10);  
glBegin(GL_TRIANGLES) ;  
    glColor3f(0.5, 0.5, 1.0);  
    glVertex2f(-2.0, 1.0); // V1  
    glVertex2f(-2.0, 0.0); // V2  
    glVertex2f(-1.0, 1.0); // V3  
    glVertex2f(-1.0, 0.0); // V4  
    glVertex2f( 0.0, 1.0); // V5  
    glVertex2f( 0.0, 0.0); // V6  
    glVertex2f( 1.0, 1.0); // V7  
    glVertex2f( 1.0, 0.0); // V8  
    glVertex2f( 2.0, 1.0); // V9  
glEnd() ;
```



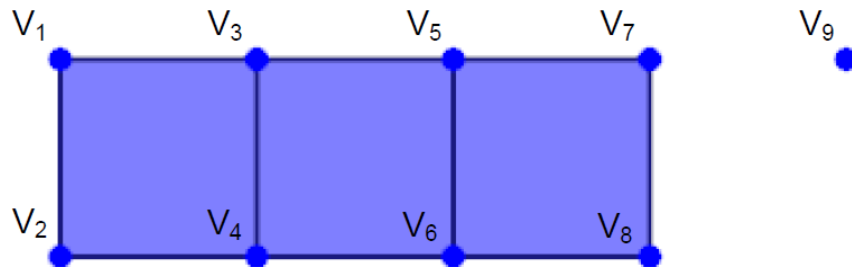
Trake trouglova

```
glLineWidth(2.0);  
glPointSize(10);  
glBegin(GL_TRIANGLE_STRIP);  
    glColor3f(0.5, 0.5, 1.0);  
    glVertex2f(-2.0, 1.0); // V1  
    glVertex2f(-2.0, 0.0); // V2  
    glVertex2f(-1.0, 1.0); // V3  
    glVertex2f(-1.0, 0.0); // V4  
    glVertex2f( 0.0, 1.0); // V5  
    glVertex2f( 0.0, 0.0); // V6  
    glVertex2f( 1.0, 1.0); // V7  
    glVertex2f( 1.0, 0.0); // V8  
    glVertex2f( 2.0, 1.0); // V9  
glEnd();
```

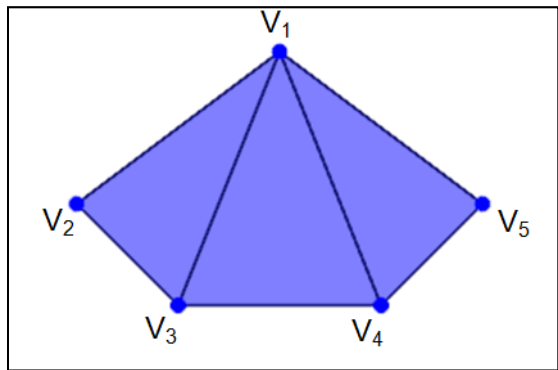


Trake četvorouglova

```
glLineWidth(2.0);  
glPointSize(10);  
glBegin(GL_QUAD_STRIP) ;  
    glColor3f(0.5, 0.5, 1.0);  
    glVertex2f(-2.0, 1.0); // v1  
    glVertex2f(-2.0, 0.0); // v2  
    glVertex2f(-1.0, 1.0); // v3  
    glVertex2f(-1.0, 0.0); // v4  
    glVertex2f( 0.0, 1.0); // v5  
    glVertex2f( 0.0, 0.0); // v6  
    glVertex2f( 1.0, 1.0); // v7  
    glVertex2f( 1.0, 0.0); // v8  
    glVertex2f( 2.0, 1.0); // v9  
glEnd() ;
```

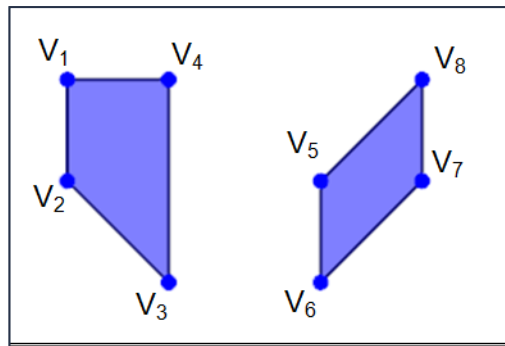
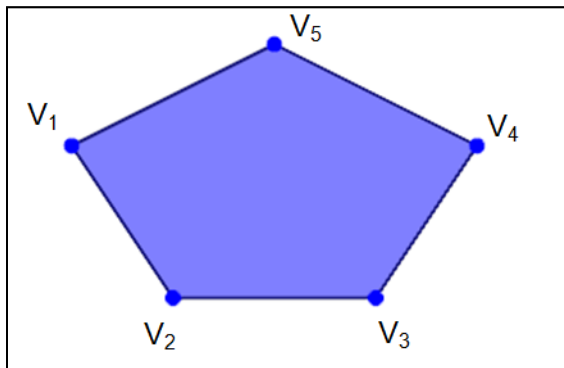


Ostale primitive



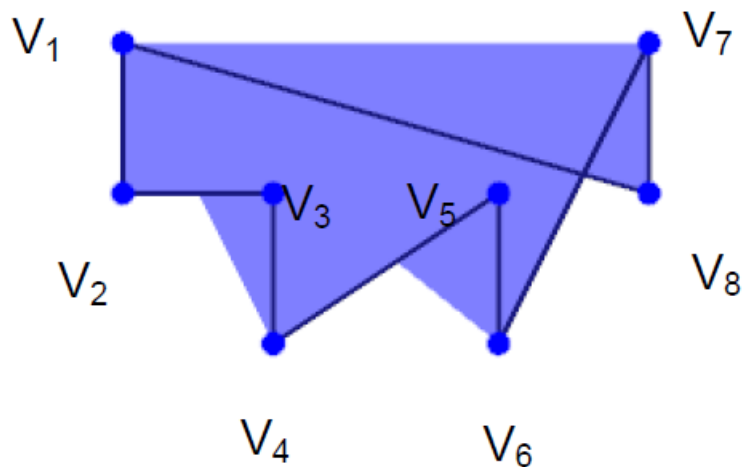
GL_TRIANGLE_FAN

GL_POLYGON

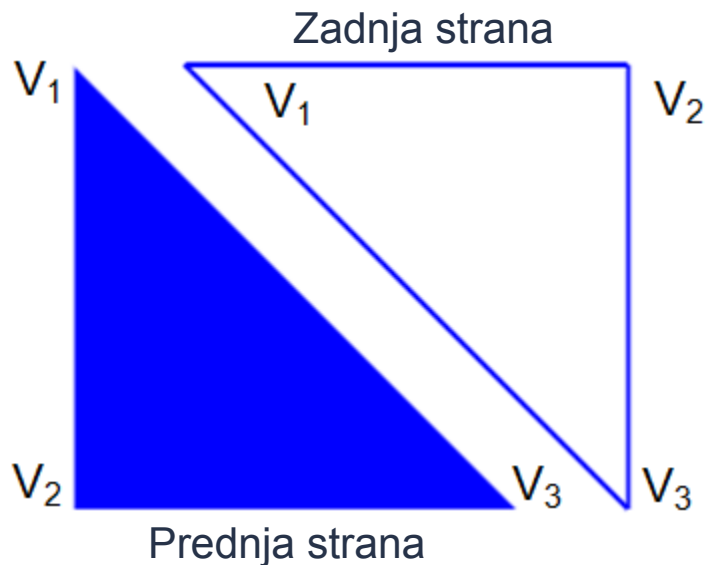


GL_QUADS

Dodatne napomene



Nekonveksne figure
ne iscrtavaju se pravilno



Strana je određena redosledom
zadavanja temena

Način senčenja

```
glShadeModel(GL_SMOOTH);  
glBegin(GL_TRIANGLES);  
    glColor3f (1.0, 0.0, 0.0);  
    glVertex3f (0.0,-1.0, 0.0);  
    glColor3f (0.0, 1.0, 0.0);  
    glVertex3f (2.0,-1.0, 0.0);  
    glColor3f (0.0, 0.0, 1.0);  
    glVertex3f (0.0, 1.0, 0.0);  
glEnd();
```



GL_SMOOTH



GL_FLAT

Primer crtanja kocke

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

    // Prednja stranica
    glVertex3d(-a/2, a/2, a/2);
    glVertex3d(-a/2,-a/2, a/2);
    glVertex3d( a/2,-a/2, a/2);
    glVertex3d( a/2, a/2, a/2);

    // Desna stranica
    glVertex3d( a/2, a/2, a/2);
    glVertex3d( a/2,-a/2, a/2);
    glVertex3d( a/2,-a/2,-a/2);
    glVertex3d( a/2, a/2,-a/2);

    // Zadnja stranica
    glVertex3d( a/2, a/2,-a/2);
    glVertex3d( a/2,-a/2,-a/2);
    glVertex3d(-a/2,-a/2,-a/2);
    glVertex3d(-a/2, a/2,-a/2);

    // Leva stranica
    glVertex3d(-a/2, a/2,-a/2);
    glVertex3d(-a/2,-a/2,-a/2);
    glVertex3d(-a/2,-a/2, a/2);
    glVertex3d(-a/2, a/2, a/2);

    // Gornja stranica
    glVertex3d(-a/2, a/2, a/2);
    glVertex3d( a/2, a/2, a/2);
    glVertex3d( a/2, a/2,-a/2);
    glVertex3d(-a/2, a/2,-a/2);

    // Donja stranica
    glVertex3d(-a/2,-a/2,-a/2);
    glVertex3d( a/2,-a/2,-a/2);
    glVertex3d( a/2,-a/2, a/2);
    glVertex3d(-a/2,-a/2, a/2);

    glEnd();
}
```


Primer crtanja kocke 2

```
void CGLRenderer::DrawCube(double dSize)
{
    double a = dSize;
    glBegin(GL_QUAD_STRIP);

    // Prednja leva vertikalna ivica
    glVertex3d(-a/2, a/2, a/2);
    glVertex3d(-a/2,-a/2, a/2);

    // Prednja desna vertikalna ivica
    glVertex3d( a/2, a/2, a/2);
    glVertex3d( a/2,-a/2, a/2);

    // Zadnja desna vertikalna ivica
    glVertex3d( a/2, a/2,-a/2);
    glVertex3d( a/2,-a/2,-a/2);

    // Zadnja leva vertikalna ivica
    glVertex3d(-a/2, a/2,-a/2);
    glVertex3d(-a/2,-a/2,-a/2);

    // Prednja leva vertikalna ivica
    glVertex3d(-a/2, a/2, a/2);
    glVertex3d(-a/2,-a/2, a/2);

    glEnd();
}
```

```
glBegin(GL_QUADS);

    // Gornja stranica
    glVertex3d(-a/2, a/2, a/2);
    glVertex3d( a/2, a/2, a/2);
    glVertex3d( a/2, a/2,-a/2);
    glVertex3d(-a/2, a/2,-a/2);

    // Donja stranica
    glVertex3d(-a/2,-a/2,-a/2);
    glVertex3d( a/2,-a/2,-a/2);
    glVertex3d( a/2,-a/2, a/2);
    glVertex3d(-a/2,-a/2, a/2);

    glEnd();
}
```

Polja temena

- Ubrzavaju iscrtavanje
- Koraci
 1. Popunjavanje polja podacima
 2. Definisanje pokazivača na polja (**glVertexPointer()/glColorPointer()/...**)
 3. Aktiviranje polja (**glEnableClientState()**)
 4. Crtanje geometrije (**glElementArray()**, **glDrawArrays()**, **glDrawElements()**,...)
 5. Deaktiviranje polja (**glDisableClientState()**)

```
void CGLRenderer::PrepareVACube(float a)
```

```
{
```

```
    vert[0] = -a/2; vert[1] = -a/2; vert[2] =  a/2; // vert0  
    vert[3] =  a/2; vert[4] = -a/2; vert[5] =  a/2; // vert1  
    vert[6] =  a/2; vert[7] =  a/2; vert[8] =  a/2; // vert2  
    vert[9] = -a/2; vert[10]=  a/2; vert[11]=  a/2; // vert3  
    vert[12]= -a/2; vert[13]= -a/2; vert[14]= -a/2; // vert4  
    vert[15]=  a/2; vert[16]= -a/2; vert[17]= -a/2; // vert5  
    vert[18]=  a/2; vert[19]=  a/2; vert[20]= -a/2; // vert6  
    vert[21]= -a/2; vert[22]=  a/2; vert[23]= -a/2; // vert7
```

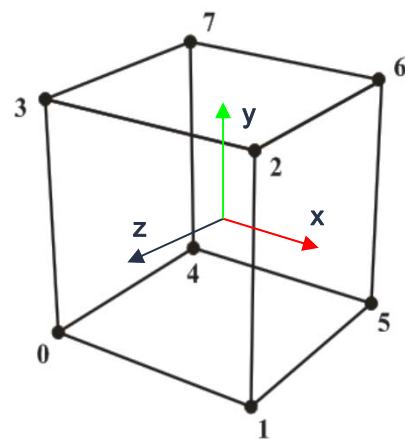
```
    col[0] = 0.0; col[1] = 0.0; col[2] = 0.0; // col0  
    col[3] = 1.0; col[4] = 0.0; col[5] = 0.0; // col1  
    col[6] = 1.0; col[7] = 1.0; col[8] = 0.0; // col2  
    col[9] = 1.0; col[10]= 1.0; col[11]= 1.0; // col3  
    col[12]= 0.0; col[13]= 1.0; col[14]= 0.0; // col4  
    col[15]= 0.0; col[16]= 1.0; col[17]= 1.0; // col5  
    col[18]= 0.0; col[19]= 0.0; col[20]= 1.0; // col6  
    col[21]= 0.0; col[22]= 0.0; col[23]= 0.0; // col7
```

```
    // Ideksi
```

```
    ind[0] = 0; ind[1] = 1; ind[2] = 2; ind[3] = 3; // quad0  
    ind[4] = 1; ind[5] = 5; ind[6] = 6; ind[7] = 2; // quad1  
    ind[8] = 7; ind[9] = 6; ind[10]= 5; ind[11]= 4; // quad2  
    ind[12]= 0; ind[13]= 3; ind[14]= 7; ind[15]= 4; // quad3  
    ind[16]= 7; ind[17]= 3; ind[18]= 2; ind[19]= 6; // quad4  
    ind[20]= 0; ind[21]= 4; ind[22]= 5; ind[23]= 1; // quad5
```

```
}
```

Definisanje polja temena



Definisanje pokazivača na polja

- void **glVertexPointer**(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);
 - ▷ *size* definiše broj koordinata po jednom temenu (**2**, **3** ili **4**)
 - ▷ *type* definiše tip podataka (**GL_SHORT**, **GL_INT**, **GL_FLOAT**, ili **GL_DOUBLE**),
 - ▷ *stride* rastojanje u bajtovima između dva sukcesivna temena. (*stride* = 0 - gusto pakovana, tj. da nema mešanja tipova atributa u okviru istog polja).
- void **glColorPointer**(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);
- void **glNormalPointer**(GLenum type, GLsizei stride, const GLvoid *pointer); - nema size jer se podrazumeva 3.
- void **glTexCoordPointer**(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);

Aktiviranje/Deaktiviranje polja

- void **glEnableClientState**(GLenum array);
 - ▷ kao parametar navodi se:
 - ▷ GL_VERTEX_ARRAY – ako želimo aktivirati polje koordinata,
 - ▷ GL_COLOR_ARRAY – ako želimo aktivirati boje,
 - ▷ GL_NORMAL_ARRAY – normale,
 - ▷ GL_TEXTURE_COORD_ARRAY – teksturne koordinate,
 - ▷ GL_INDEX_ARRAY – indeksi,
 - ▷ itd.
- void **glDisableClientState**(GLenum array)

Crtanje

- **glArrayElement()** – proizvoljno “skakanje” po polju,
- **glDrawArrays()** – sekvencijalni pristup elementima u polju,
- **glDrawElements()** – metodično skakanje po polju na osnovu polja indeksa

glDrawElements

- void **glDrawElements**(GLenum *mode*, GLsizei *count*, GLenum *type*, const GLvoid **indices*);
 - ▷ *mode* – tip primitive koja se crta (GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_TRIANGLES, GL_QUAD_STRIP, GL_QUADS, ili GL_POLYGON)
 - ▷ *count* – broj indeksa u polju indeksa
 - ▷ *type* - tip polja indeksa(GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT ili GL_UNSIGNED_INT)
 - ▷ *indices* – pokazivač na polje indeksa

Crtanje kocke funkcijom `glDrawElements`

```
void CGLRenderer::DrawVACube()
{
    glVertexPointer(3, GL_FLOAT, 0, vert);
    glColorPointer(3, GL_FLOAT, 0, col);
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_COLOR_ARRAY);

    glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, ind);

    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_COLOR_ARRAY);
}
```

*Zašto 24 indeksa? $6 \text{ strana} * 4 \text{ temena} = 24$*

Koliko indeksa treba ako se za iscrtavanje koriste `GL_TRIANGLES`?


```
void CGLRenderer::PrepareVACube(float a)
{
```

```
    // Zajednicko polje
```

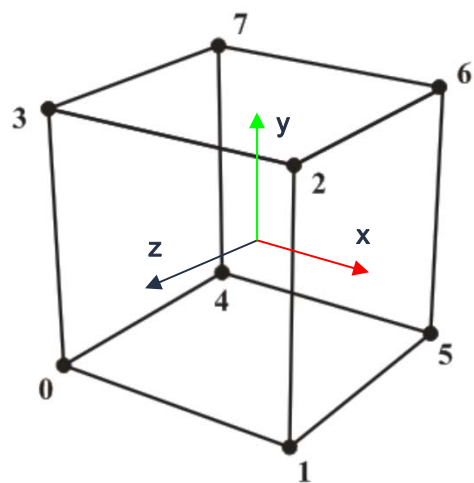
```
    buf[0] = 0.0; buf[1] = 0.0; buf[2] = 0.0; // col0
    buf[3] = -a/2; buf[4] = -a/2; buf[5] = a/2; // ver0
    buf[6] = 1.0; buf[7] = 0.0; buf[8] = 0.0; // col1
    buf[9] = a/2; buf[10] = -a/2; buf[11] = a/2; // ver1
    buf[12] = 1.0; buf[13] = 1.0; buf[14] = 0.0; // col2
    buf[15] = a/2; buf[16] = a/2; buf[17] = a/2; // ver2
    buf[18] = 1.0; buf[19] = 1.0; buf[20] = 1.0; // col3
    buf[21] = -a/2; buf[22] = a/2; buf[23] = a/2; // ver3
    buf[24] = 0.0; buf[25] = 1.0; buf[26] = 0.0; // col4
    buf[27] = -a/2; buf[28] = -a/2; buf[29] = -a/2; // ver4
    buf[30] = 0.0; buf[31] = 1.0; buf[32] = 1.0; // col5
    buf[33] = a/2; buf[34] = -a/2; buf[35] = -a/2; // ver5
    buf[36] = 0.0; buf[37] = 0.0; buf[38] = 1.0; // col6
    buf[39] = a/2; buf[40] = a/2; buf[41] = -a/2; // ver6
    buf[42] = 0.0; buf[43] = 0.0; buf[44] = 0.0; // col7
    buf[45] = -a/2; buf[46] = a/2; buf[47] = -a/2; // ver7
```

```
    // Ideksi
```

```
    ind[0] = 0; ind[1] = 1; ind[2] = 2; ind[3] = 3; // quad0
    ind[4] = 1; ind[5] = 5; ind[6] = 6; ind[7] = 2; // quad1
    ind[8] = 7; ind[9] = 6; ind[10] = 5; ind[11] = 4; // quad2
    ind[12] = 0; ind[13] = 3; ind[14] = 7; ind[15] = 4; // quad3
    ind[16] = 7; ind[17] = 3; ind[18] = 2; ind[19] = 6; // quad4
    ind[20] = 0; ind[21] = 4; ind[22] = 5; ind[23] = 1; // quad5
```

```
}
```

Kombinovanje atributa u jednom vektoru



Crtanje kocke funkcijom glDrawElements

```
buf[0] = 0.0; buf[1] = 0.0; buf[2] = 0.0; // col0  
buf[3] = -a/2; buf[4] = -a/2; buf[5] = a/2; // ver0
```

```
void CGLRenderer::DrawVACube()  
{  
    glVertexPointer(3, GL_FLOAT, 6*sizeof(float), &buf[3]);  
    glColorPointer( 3, GL_FLOAT, 6*sizeof(float), &buf[0]);  
  
    glEnableClientState(GL_VERTEX_ARRAY);  
    glEnableClientState(GL_COLOR_ARRAY);  
  
    glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, ind);  
  
    glDisableClientState(GL_VERTEX_ARRAY);  
    glDisableClientState(GL_COLOR_ARRAY);  
}
```

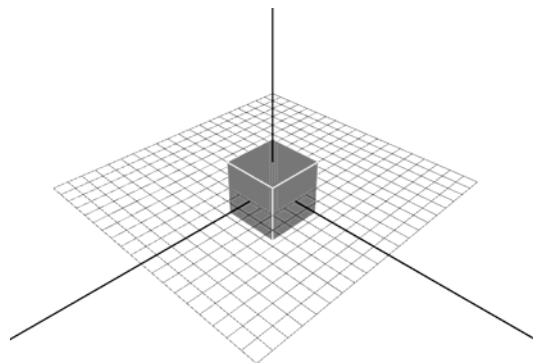
Zadatak

Napisati sledeće funkcije:

- ▷ void **DrawGrid**(double dSize, int nSteps) – koja iscrtava mrežu dimenzija $dSize \times dSize$ oko koordinatnog početka, u XZ-ravni sa $nSteps$ koraka,
- ▷ void **DrawCube**(double dSize) – koja iscrtava kocku stranice $dSize$ sa centrom u koordinatnom početku i
- ▷ void **DrawAxes**(double len) – koja iscrtava koordinatne ose dužine len .

U okviru funkcije **CGLRenderer::DrawScene()** formirati scenu koja se sastoji od:

- ▷ mreže dimenzija 5x5, sa 20 koraka po svakoj osi, crne boje i debljine 1 piksel,
- ▷ kocke dimenzija 1x1x1 sive bije sa ivicama bele boje debljine 2 piksela i
- ▷ koordinatnih osa dužina 10, crne boje i debljine 3 piksela.



Pitanja

