

### **Lucrarea nr. 3**

## **Elemente de grafică în OpenGL**

### ***1. Scopul lucrării:***

Lucrarea de față își propune prezentarea conceptelor și tehniciilor de bază OpenGL. În această lucrare se prezintă biblioteca OpenGL – AUX, tipurile de date, modele de culoare, bufferele, funcțiile CallBack și modul de construire al aplicațiilor grafice simple utilizând această extensie. Sunt prezentate modalitățile de utilizare a bibliotecii OpenGL în aplicații Windows dezvoltate folosind platforma Microsoft Foundation Classes (MFC). Lucrarea se încheie cu prezentarea primitivelor OpenGL.

### ***2. Noțiuni teoretice:***

#### **2.1. OpenGL - Open Graphics Library – prezentare generală**

Open Graphics Library (OpenGL) este una dintre cele mai răspândite biblioteci pentru grafica tridimensională.

**OpenGL este definit ca “interfața software pentru hardware-ul grafic”**

OpenGL este o bibliotecă portabilă pentru grafică și modelare 3D. Implementările pentru diferite platforme (Windows, Linux, UNIXes, Solaris, MacOS, etc.) sunt optimizate pentru dispozitive grafice cu acceleratori 3D *on board*, iar funcțiile bibliotecii care nu sunt suportate direct de către hardware-ul grafic sunt emulate software. Biblioteca OpenGL este utilizată în cele mai diverse domenii, de la aplicații CAD sau medicale la realizarea de filme cu ajutorul calculatorului (scene generate pe calculator din Terminator Two: Judgement Day sau StarWars sunt binecunoscute tuturor). OpenGL este o bibliotecă **procedurală**, adică programatorul trebuie să specifice fiecare pas ce trebuie executat pentru a obține rezultatul final.

Trecutul OpenGL-ului se ”pierde” în laboratoarele firmei Silicon Graphics Inc.(SGI) SUA în anii ’80. Inițial a fost IRIS GL, un API pentru modelarea și generarea imaginilor 3D folosind stațiile grafice IRIS. Ca urmare a eforturilor celor de la SGI orientate în direcția standardizării acestui API, la 1 iulie 1992 a apărut versiunea 1.0 a specificațiilor standardului OpenGL. OpenGL Architecture Review Board (ARB) este un consorțiu (fondat de către SGI, Digital Equipment Corporation, IBM, Intel și Microsoft) care se ocupă cu evoluția standardului în timp. În decembrie 1995 au fost definitivate și au fost publicate specificațiile versiunii 1.1. Versiunea curentă este 1.5 lansată oficial în luna iulie 2003. De viitorul bibliotecii se îngrijește firma 3DLabs ce lucrează deja la specificațiile noii generații OpenGL 2.0 care va fi probabil un limbaj de umbrire (Shading Language), un nou standard grafic pentru anii ce vor urma.

În paralel cu evoluția standardului OpenGL sunt dezvoltate numeroase biblioteci auxiliare numite extensii. Dintre cele mai cunoscute extensii amintim GLUT (OpenGL Utility Toolkit), GLU (OpenGL Graphics System Utility) sau GLX (OpenGL Grafics with X Window System) (*Tabelul 1*). Deși nu sunt parte integrantă a standardului OpenGL, ele sunt publice și monitorizate de către firma SGI prin programul OpenGL Extension Registry.

Denumire	Fișier bibliotecă	Fișier header	Prefix	Conținut
OpenGL	opengl32.lib	gl.h	gl	Funcții standard OpenGL
OpenGL Utility Library	glu32.lib	glu.h	glu	Funcții utilitare
OpenGL AUX Library Toolkit	glaux.lib	glaux.h	aux	Funcții auxiliare

Tabelul 1 – Organizare OpenGL

Folosind extensiile, dezvoltatorii de produse hardware pentru grafica 3D pot să încorporeze noi funcționalități în acestea și să ofere programatorilor un punct de acces pentru ele. Nu de puține ori, cele mai inovative facilități dintr-o extensie sunt incluse în următoarea versiune a standardului OpenGL. Însă, pentru a păstra portabilitatea acestei biblioteci pe diverse platforme hardware și software, standardul OpenGL nu oferă (și probabil nici nu va oferi) mecanisme pentru managementul ferestrelor, pentru accesul la sistemul de fișiere sau pentru operațiile de intrare- ieșire. Acestea vor rămâne în sarcina unor extensii precum GLUT, AUX sau GLX.

Extensia GLUT implementează un sistem de ferestre portabil (Win 32, Mac OS și Linux/UNIX) care ușurează învățarea OpenGL și scrierea primelor aplicații (AUX este o aplicație similară, mult simplificată). Extensia GLX este folosită pentru implementările Unix ale OpenGL pentru a administra interacțiunea cu sistemul de ferestre X Window System și pentru a codifica OpenGL în protocolul X stream pentru randare la distanță (remote).

GLU (OpenGL Graphics System Utility) oferă o colecție de funcții pentru crearea texturilor mipmapped, transformă coordonatele între spațiul ecran și cel obiect și desenează supafe de gradul patru precum și supafe NURBS (supafe neuniforme – **Nonuniform Rational B-Splines**).

Direcția recentă în construirea hardware-ului grafic este de a înlocui funcționalitatea fixată, bine definită, cu cea programabilă în sectoare care au devenit extrem de complexe (cum ar fi procesarea vârfurilor și a fragmentelor).

Pentru dezvoltarea aplicațiilor utilizând biblioteca OpenGL, cel mai frecvent se utilizează limbajul de programare C/C++. Pentru accesarea din alte limbi de programare au fost dezvoltate așa numitele *Java Bindings*. Cele mai multe au fost dezvoltate pentru limbajul Java: Java3D, JOGL, GL4Java(OpenGL for Java), YAJOGLB(Yet Another Java OpenGL Binding), Tree (interfață Java pentru Mesa 3D), JavaOpenGL. Mai sunt disponibile bindinguri pentru Fortran (f90GL), Perl(OpenGL Perl Module, Perl OpenGL Bindings), Pike(Pikes GL), Python(PyOpenGL, PyGlut), Ada(Ada OpenGL binding) și controalele ActiveX pentru legătura cu Visual Basic.

## 2.2. Arhitectura OpenGL sub sistemul de operare Windows

Implementarea standardului OpenGL sub sistemul de operare Windows (95/98/ME/NT/2000) este realizată de către firma Microsoft Corp. și se încadrează în arhitectura multi-nivel a sistemului de operare. Implementarea Microsoft a standardului

OpenGL pentru Windows conține pe lângă standardul 1.1 și două extensii: GLU și AUX. Modulul OpenGL conține implementarea funcțiilor standard OpenGL conform specificațiilor OpenGL ARB. Extensia OpenGL Utility implementează funcții utilitare pentru generarea corpurilor 3D complexe, pentru definirea proiecției perspective, pentru construirea și maparea texturilor 1D și 2D, facilități ce fac lucru cu OpenGL mult mai ușor.

Extensia **OpenGL AUX** Library implementează un cadru de lucru independent de platformă pentru apelul funcțiilor OpenGL. Conține funcții auxiliare pentru tratarea operațiilor de intrare-ieșire (tastatura, mouse), pentru managementul ferestrelor, pentru inițializarea motorului grafic OpenGL, etc. Cu toate că aceste funcții nu fac parte din standardul OpenGL, extensia AUX (sau GLUT) este disponibilă pe diverse platforme (Solaris, UNIX, etc.).

### 2.2.1. Tipuri de date OpenGL

Pentru a sigura portabilitatea codului pe diverse arhitecturi hardware și software este necesară introducerea unor tipuri de date care să aibă aceeași reprezentare pe toate sistemele. În tabelul 2 sunt prezentate tipurile de date introduse în standardul OpenGL.

Tip de date	Reprezentare	Echivalent C	Sufix
GLbyte	întreg pe 8 biți	signed char	b
GLshort	întreg pe 16 biți	short	s
GLint, GLsize	întreg pe 32 de biți	long	l
GLfloat, GLclampf	virgulă mobilă pe 32 biți	float	f
GLdouble, GLclampd	virgulă mobilă pe 64 biți	double	d
GLubyte, GLboolean	întreg fără semn pe 8 biți	unsigned char	ub
GLushort	întreg fără semn pe 16 biți	unsigned short	us
GLuint, GLenum, GLbitfield	întreg fără semn pe 32 de biți	unsigned long	ui

Tabelul 2 – Tipuri de date OpenGL

*Pentru realizarea unor aplicații portable trebuie folosite doar tipurile de date OpenGL, altfel compilatoarele C/C++ vor reprezenta diferit variabilele, în funcție de sistemul de operare pe care se va compila aplicația.*

### 2.2.2. Convenția folosită la asocierea numelui funcțiilor

Toate funcțiile OpenGL (atât cele specificate în standard, cât și cele introduse în extensiile Utility sau AUX) respectă următoarea convenție de construire a numelui:

*<prefix><nume funcție><număr argumente><tip argumente>*

unde:

<prefix> reprezintă prefixul bibliotecii (standard sau extensie) în care este definită funcția (vezi coloana a patra din tabelul 1).

<nume funcție> reprezintă un verb care definește acțiunea realizată de această funcție.

<număr argumente> reprezintă numărul de argumente ale funcției (optional).

<tip argumente> reprezintă tipul argumentului (optional) (vezi coloana a patra din tabelul 2).

De exemplu, funcția **glColor3f** face parte din modulul OpenGL standard și are 3 parametrii de tip GLfloat.

### 2.2.3. Contextul de randare OpenGL

**Contextul de randare OpenGL** (OpenGL rendering state) reprezintă colecția tuturor variabilelor de stare utilizate de OpenGL la generarea unei scene. Contextul de randare este structurat logic în mai multe categorii (*Tabelul 3*).

Categorie	Descriere	Exemple de variabile
Desenare	folosite la desenarea primitiveelor cu glBegin/glEnd	GL_POINT_SMOOTH GL_POINT_SIZE GL_LINE_STIPPLE
Buffer de adâncime	activează testul de adâncime	GL_DEPTH_TEST
Buffer şablon	activează modul de lucru cu acest buffer	GL_STENCIL_TEST
Iluminare	folosite la controlul iluminării, definirea materialelor și culorilor, specificarea normalelor, etc.	GL_LIGHT* GL_LIGHTING GL_NORMALIZE GL_COLOR_MATERIAL
Texturare	folosite la texturare	GL_TEXTURE_1D GL_TEXTURE_2D GL_MAP* TEXTURE_COORD*
Pixel	folosite la transpunerea și stocarea pixelilor și a modurilor de mapare	GL_MAP_COLOR GL_PIXEL_MAP_* GL_RGB GL_LUMINANCE GL_COLOR_INDEX

Tabelul 3 – Contextul de randare OpenGL

Variabilele de stare au diferite tipuri: boolean, întreg, flotant, vector de booleene, întregi sau flotante. Pentru a consulta/modifica aceste variabile, OpenGL pune la dispoziția programatorului un set de funcții diferențiate după tipul variabilei (vezi *tabelul 4*).

Pentru a salva/restaura starea actuală a contextului de randare în stiva contextului, se folosesc funcțiile:

```
void glPushAttrib(GLuint bits);
void glPopAttrib();
```

unde bits poate fi GL\_ACCUM\_BUFFER\_BIT, GL\_LIST\_BIT, GL\_COLOR\_BUFFER\_BIT, GL\_DEPTH\_BUFFER\_BIT, GL\_EVAL\_BIT, etc.

Tip variabilă	Consultare	Modificare
GLboolean	glIsEnabled glIsDisabled	glEnable glDisable
GLboolean[]	glGetBooleanv	
GLint, GLint[]	glGetIntegerv	
GLfloat, GLfloat[]	glGetFloatv	
GLdouble, GLdouble[]	glGetDoublev	

Tabelul 4 – Funcții de acces la variabilele de stare

## 2.2.4. Codificarea culorii

În biblioteca OpenGL sunt implementate două modele de culoare: modelul **RGBA** și modelul tabelei de culori (paleta de culori). În modelul RGBA culoarea este reprezentată prin componente R, G, B (rosu, verde, albastru) și transparență A cu valori în intervalul [0, 1]. În modelul tabelei de culori, culoarea unei primitive sau a unui pixel reprezintă un indice într-o tabelă de culori în care sunt memorate pentru fiecare intrare (index) componentele corespunzătoare R, G, B, A ale culorii.

Culoarea finală a unui pixel de pe ecran este stocată în buffer-ul de culoare și depinde de mai multe condiții, fiind o combinație între culorile primitive ce se proiectează în acel pixel, intensitățile surselor de lumină din scenă, tehnica de anti-aliasing folosită, texturarea și modelul de iluminare folosit.

Culoarea curentă de desenare se stabilește folosind funcția *glColor* și este stocată în variabila de stare *GL\_CURRENT\_COLOR*.

## 2.2.5. Buffere OpenGL

**Un buffer** OpenGL este o zonă de memorie (tablou dimensional) în care se stochează informații despre imaginea ce se desenează în fereastra de lucru sau într-o zonă ascunsă.

OpenGL gestionează patru buffere pentru fiecare fereastră:

- **bufferul de culoare** (*GL\_COLOR\_BUFFER\_BIT*): stochează culorile pixelilor ferestrei de lucru.
- **bufferul de adâncime** (*GL\_DEPTH\_BUFFER\_BIT*): stochează distanța de la fiecare pixel la poziția observatorului și este folosit în algoritmul z-buffer pentru eliminarea suprafeteelor ascunse.
- **buffer şablon** (*GL\_STENCIL\_BUFFER\_BIT*): are mai multe aplicații în generarea scenelor complexe, cum ar fi restricționarea desenării scenei la o zonă de ecran, generarea umbrelor pentru surse de lumină multiple, etc.
- **bufferul de acumulare** (*GL\_ACCUMULATION\_BUFFER\_BIT*): este cel mai simplu și are aplicații în simularea unor efecte (full-screen, anti-aliasing, motion blur, câmp de adâncime – depth of field), etc.

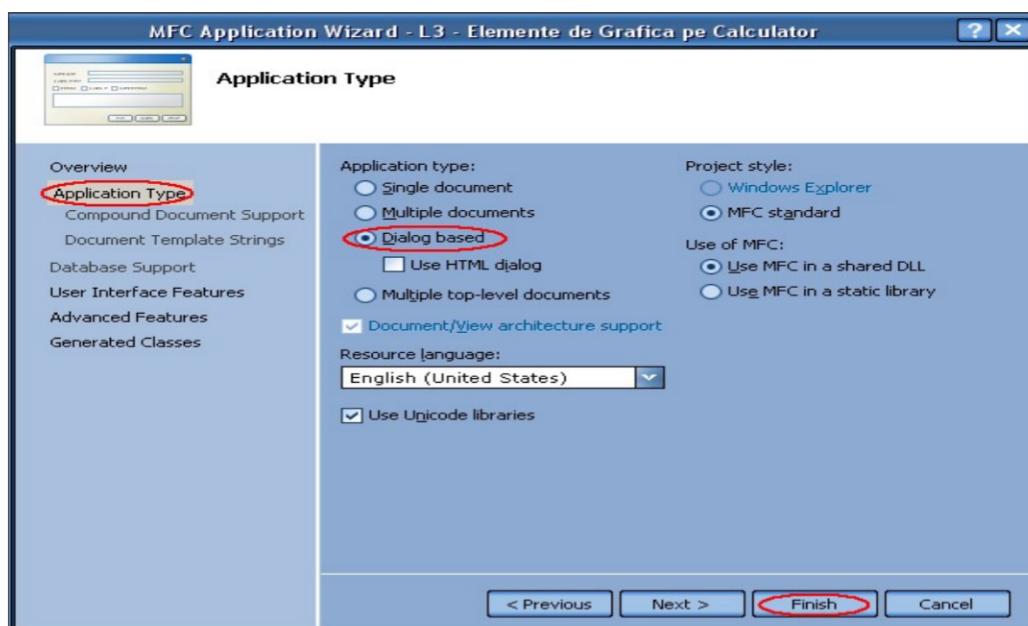
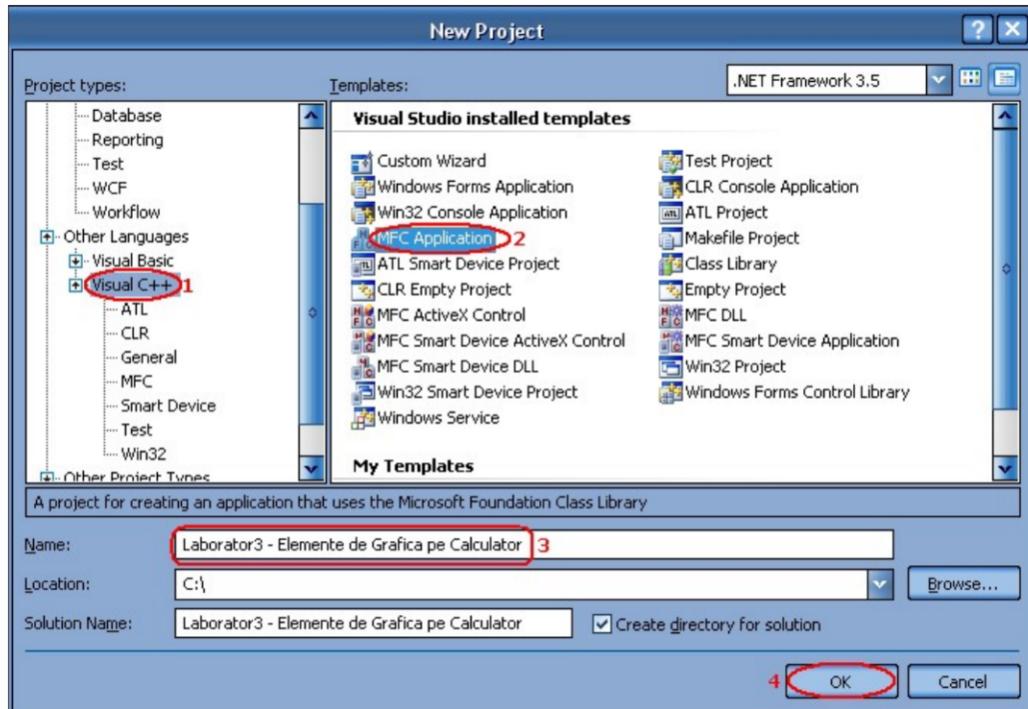
Identifierul din paranteze reprezintă argumentul ce trebuie trimis funcției *glClear* în vederea inițializării conținutului întregului buffer.

## 2.3. Aplicații grafice folosind OpenGL sub sistemul de operare Windows

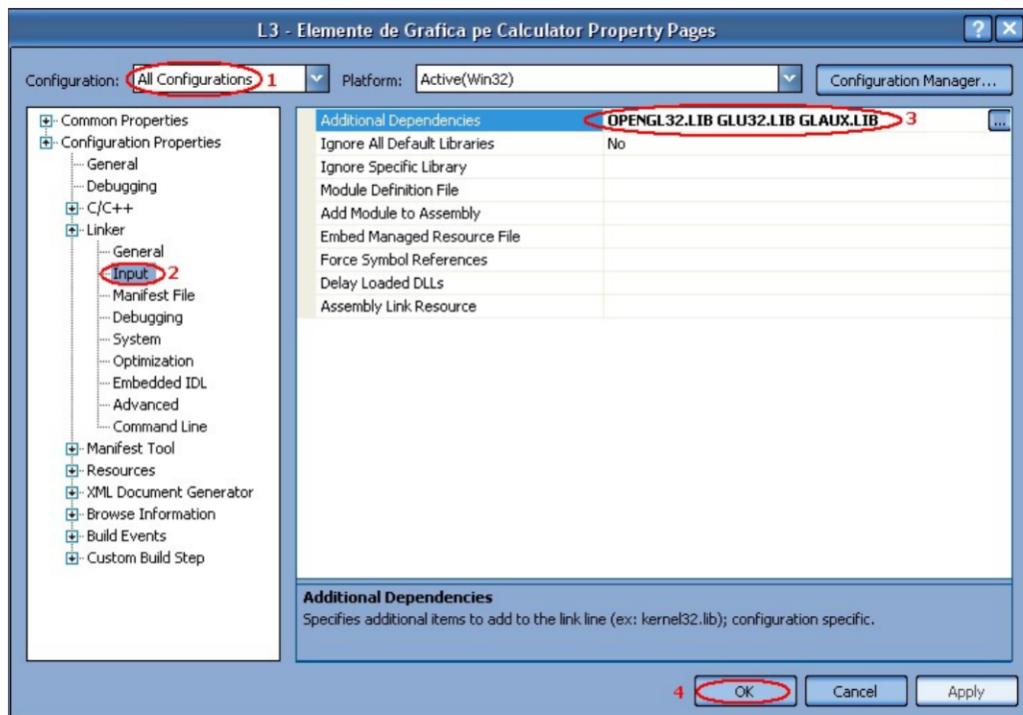
### OpenGL și MFC

Toate exemplele din acest laborator utilizează biblioteca AUX pentru crearea ferestrei de lucru OpenGL și pentru tratarea evenimentelor de intrare-iesire. Etapele ce trebuie urmate pentru a transforma un context de afișare Windows într-un dispozitiv de randare OpenGL sunt următoarele:

1. **Folosind Microsoft Visual Studio, se creează un proiect de tip aplicație MFC AppWizard.**



2. Se adaugă bibliotecile **OPENGL32.LIB GLU32.LIB GLAUX.LIB** la lista de biblioteci a editorului de legături.



3. Se adaugă în fișierul header *stdafx.h*:

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

**Exemplul 1:** Cel mai simplu program OpenGL (Figura 1)

```
void CTeste_OpenenglDlg::OnBnClickedButton1()
{
    //Stabilirea ferestrei de lucru OpenGL
    auxInitDisplayMode(AUX_SINGLE|AUX_RGB);
    auxInitPosition(100,100,250,250);
    auxInitWindowA("Primul Exemplu OpenGL");

    //Desenează fundalul albastru
    glClearColor(0.0f,0.0f,1.0f,1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
```



Figura 1 – Umpierea ferestrei de lucru OpenGL cu albastru

Exemplul de mai sus realizează umplerea ferestrei de lucru cu culoarea albastră. Primele trei apeluri de funcții AUX (numite și **comenzi** AUX sau comenzi OpenGL) inițializează mediul de lucru OpenGL.

Comanda

```
auxInitDisplayMode(AUX_SINGLE|AUX_RGB);
```

indică bibliotecii AUX ce mod de afișare să folosească la crearea ferestrei și anume: o fereastră de lucru single-buffer (AUX\_SINGLE), iar modelul de culoare folosit este RGBA (AUX\_RGBA). Într-o fereastră de lucru single-buffer toate comenziile de desenare sunt executate în zona afișată. Spre deosebire de acest mod de lucru, în modul double-buffer comenziile sunt executate într-o zonă ascunsă (off screen) după care zona vizibilă și zona ascunsă sunt interschimbate foarte rapid utilizând funcția auxSwapBuffers. Acest mod de lucru este de preferat în cazul animațiilor pentru a elimina efectul de “clipire”. Modul RGBA înseamnă că pentru specificarea culorii se vor folosi intensități de roșu, verde și albastru.

#### Apelul

```
auxInitPosition(100,100,250,250);
```

creează o fereastră de dimensiune 250 \* 250 pixeli, poziționată inițial la coordonatele 100 \* 100. În sistemul de operare Windows originea sistemului este în colțul stânga-sus al ecranului și acest sistem este utilizat în exemplul de mai sus. Spre deosebire de acest mod de lucru, OpenGL mapează întotdeauna (indiferent de mediul grafic folosit) coordonatele considerând originea sistemului de coordonate al ferestrei în colțul stânga-jos al ferestrei de lucru. În același timp, funcția auxInitPosition mai creează și următoarele:

- Un viewport de dimensiune egală cu dimensiunea inițială a ferestrei (adică 250 \* 250) și cu originea în colțul din stânga-jos al ferestrei.
- Un volum de vedere (clipping volume) cuprins între 0 și 250 pe axe Ox și Oy și între 1.0 și -1.0 pe axa Oz.

#### Apelul

```
auxInitWindowA("Primul Exemplu OpenGL");
```

creează fereastra de lucru OpenGL cu titlul “Primul Exemplu OpenGL”.

Următoarele trei comenzi OpenGL ale programului – și anume apelurile funcțiilor glClearColor, glClear, glFlush – realizează umplerea cu albastru a fundalului ferestrei de lucru. Funcția glClearColor stabilește culoarea de stergere (fundalul) a ferestrei, iar funcția glClear sterge efectiv bufferul specificat ca argument. În continuare, funcția glFlush “forțează” executarea tuturor comenziilor OpenGL emise până în acel moment și neexecutate. OpenGL folosește intern un *pipeline de randare* (rendering pipeline) care procesează comenziile secvențial. Rolul acestuia este acela de a crește viteza de randare a scenei prin procesarea în grup a comenziilor emise, așa încât la un moment dat pot exista comenzi neexecutate.

Exemplul prezentat mai sus nu redesenează fereastra de lucru atunci când este nevoie și nici nu tratează corespunzător evenimentul de redimensionare a ferestrei. Pentru a înlătura aceste neajunsuri, vom utiliza mijloacele puse la dispoziție de biblioteca AUX, și anume *funcțiile callback*.

Pentru a înregistra funcția corespunzătoare unui eveniment se folosesc funcții puse la dispoziție de către biblioteca AUX. În *tabelul 5* sunt prezentate evenimentele ce pot apărea în sistem și modul lor de tratare.

Eveniment	Funcția de înregistrare	Prototip funcție CALLBACK	Momentul apelului
Redesenarea ferestrei	auxMainLoop	void CALLBACK MainFunc(void)	La redesenarea ferestrei
Redimensionarea ferestrei	auxReshapeFunc	void CALLBACK ChangeSize(GLsizei w, GLsizei h)	La redimensionarea ferestrei
Idle	auxIdleFunc	void CALLBACK IdleFunc(void)	Când nu se execută altceva
Apăsarea unei taste	auxKeyFunc	void CALLBACK	La apăsarea tastei

taste		KeyFunc(void)	indicate
Acționarea butoanelor mouse-ului	auxMouseFunc	void CALLBACK MouseFunc(AUX_EVENTRC* event)	La apăsarea sau eliberarea unui buton al mouse-ului

Tabelul 5 - Funcțiile CALLBACK

În exemplul următor vom folosi funcțiile auxMainLoop și auxReshapeFunc pentru a înregistra funcțiile de redesenare (RedeseneazăScena) și de redimensionare (ModificăDimensiune) astfel încât neajunsurile din primul exemplu vor fi eliminate.

### Exemplul 2: Redesenarea și redimensionarea ferestrei de lucru OpenGL

```

void CTeste_OpenglDlg::OnBnClickedButton1() //functie atasata la buton
{
    //Stabilirea ferestrei de lucru OpenGL
    auxInitDisplayMode(AUX_SINGLE|AUX_RGB);
    auxInitPosition(100,100,250,250);
    auxInitWindow("Al doilea Exemplu OpenGl");

    //Înregistrarea funcțiilor CALLBACK
    auxReshapeFunc(ModificaDimensiune);
    auxMainLoop(RedeseneazaScena);
}

//Apelata de biblioteca AUX pentru redesenarea ferestrei OpenGL
void CALLBACK RedeseneazaScena(void)
{
    //Culoarea de stergere (albastru)
    glClearColor(0.0f,0.0f,1.0f,1.0f);
    //Stergerea ferestrei
    glClear(GL_COLOR_BUFFER_BIT);
    //Culoarea de desenare (rosu)
    glColor3f(1.0f,0.0f,0.0f);
    //Desenare dreptunghi rosu
    glRectf(100.0f, 150.0f, 150.0f, 100.0f);
    glFlush();
}

//Apelata de biblioteca AUX când s-a schimbat dimensiunea ferestrei
void CALLBACK ModificaDimensiune(
    GLsizei w, //noua lățime a ferestrei
    GLsizei h) //noua înălțime a ferestrei
{

    if(h==0) h=1;
    //Stabilirea viewportului la dimensiunea ferestrei
    glViewport(0,0,w,h);
    //Reinitializează sistemul de coordonate
    glLoadIdentity();
    //Stabilește volumul de vedere folosind o proiecție ortografică
    if(w<=h)
        glOrtho(0.0f, 250.0f, 0.0f, 250.0f*h/w, 1.0f, -1.0f);
    else
        glOrtho(0.0f, 250.0f*w/h, 0.0f, 250.0f, 1.0f, -1.0f);
}

```

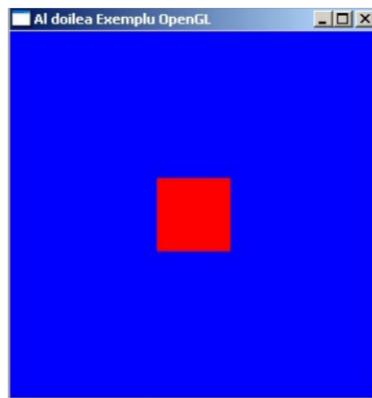


Figura 2 – Pătrat în OpenGL

La redimensionarea ferestrei OpenGL, funcția `ModificaDimensiune` schimbă `viewportul` (*un viewport este regiunea dreptunghiulară din fereastra de lucru OpenGL în care va fi proiectat volumul de vedere 3D*) (prin apelul `glViewport(0, 0, w, h)`) și volumul de vedere (prin apelul funcției `glOrtho`) astfel încât dimensiunea `viewportului` rămâne tot timpul egală cu cea a ferestrei, iar volumul de vedere este scalat pe axele Ox și Oy astfel încât pătratul desenat să rămână pătrat indiferent de dimensiunile ferestrei (înmulțirea/împărțirea cu factorul de aspect  $h/w$ ).

Un `viewport` este regiunea dreptunghiulară din fereastra de lucru OpenGL în care va fi proiectat volumul de vedere 3D. Figura 3 arată un `viewport` mapat pe întreaga dimensiune a ferestrei (stânga), respectiv doar pe o zonă din fereastră (dreapta). În cazul definirii unui `viewport` mai mic decât fereastra OpenGL, randarea scenei se va face doar în zona definită de `viewport`.

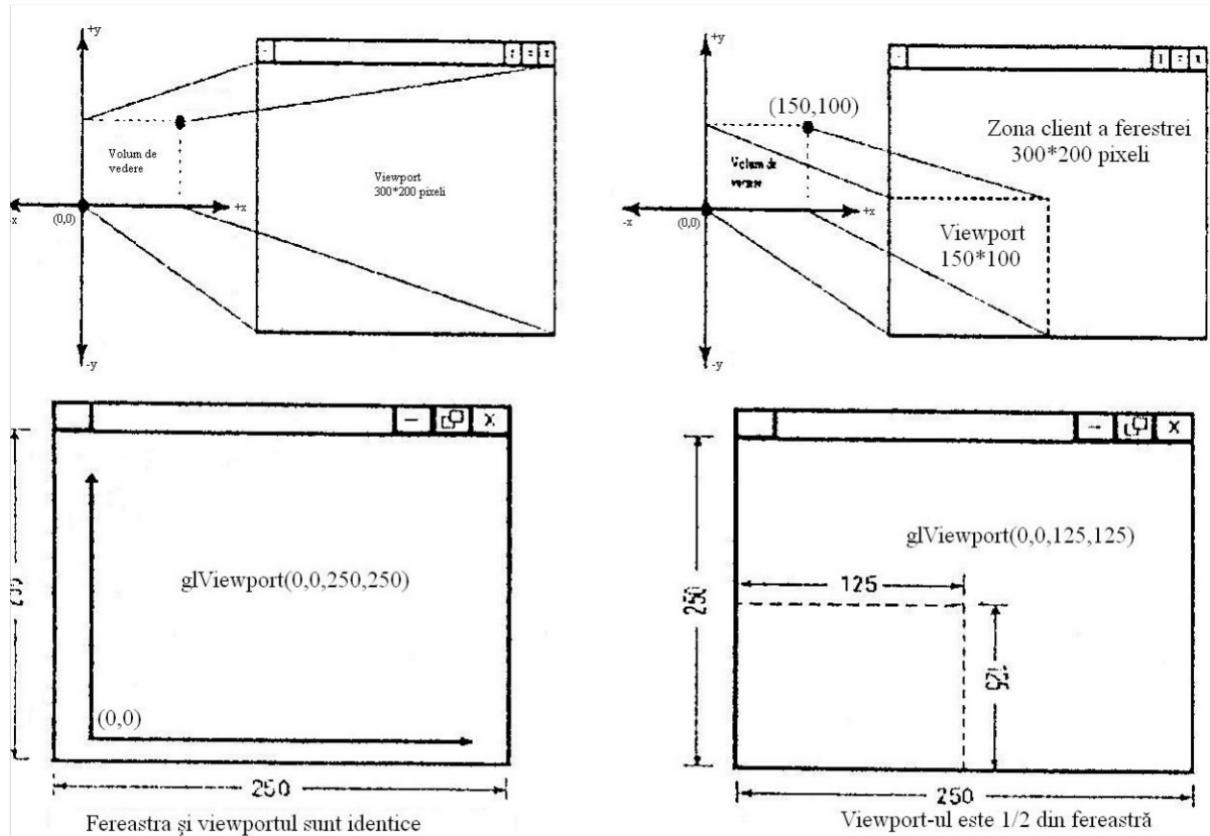
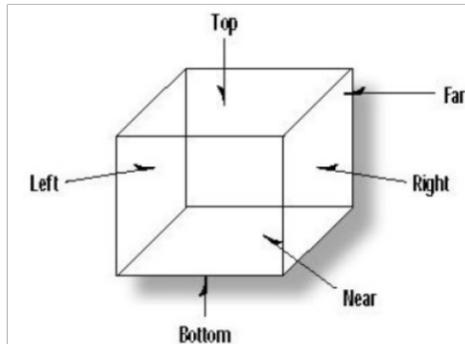


Figura 3 – Viewportul OpenGL

În standardul OpenGL se pot defini două tipuri de volume de vedere :

- **Paralelipipedic**, corespunzător unei **proiecții ortogonale (paralele)**, folosind funcția `void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)`.



**Figure 2-13** The clipping volume for an orthographic projection

- **Trunchi de piramidă**, corespunzător unei **proiecții perspective**, folosind funcția `void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)`, unde `fovy` este unghiul câmpului de vedere în direcția verticală, `aspect` reprezintă rata de aspect (raportul lățime/înălțime), iar `zNear` și `zFar` specifică adâncimea planului apropiat, respectiv îndepărtat al trunchiului de piramidă.

În perspectivă, razele vizuale pleacă din punctele obiectului și sunt concurente în centrul de proiecție al ochiului (fig. 3.1), în timp ce în proiecția paralelă razele vizuale sunt considerate paralele. După adoptarea tipului proiecției, punctele acesteia pot fi calculate ușor pe baza datelor geometrice ale obiectului, precum și a poziției relative față de punctul de observare și respectiv ecran.

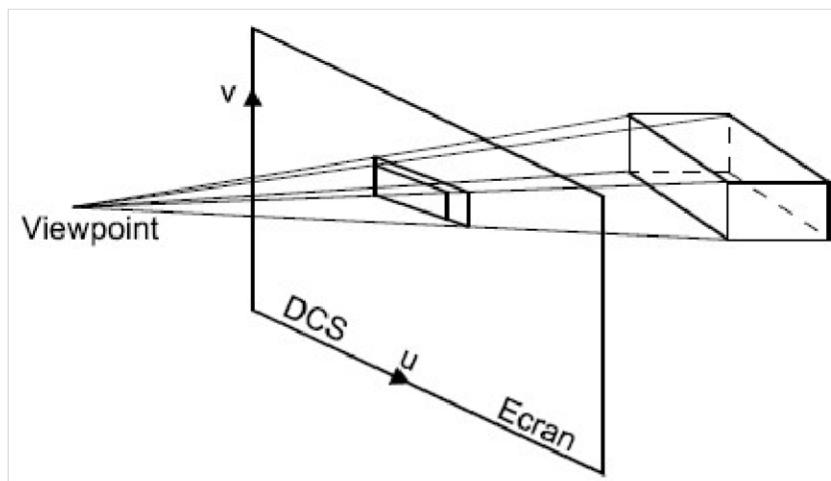
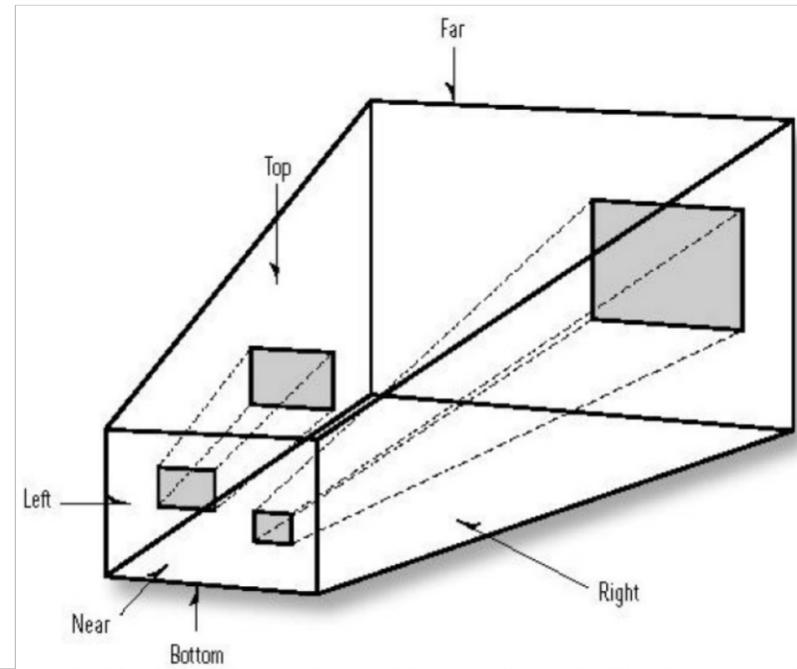


Figura 3.1 – Proiecție perspectivă



**Figure 2-14** The clipping volume for a perspective projection

Exemplul următor demonstrează realizarea unei aplicații simple utilizând funcțiile bibliotecii AUX. Se utilizează funcția auxIdleFunc pentru a înregistra funcția de procesare a timpului idle al sistemului. Un alt aspect care trebuie menționat este utilizarea modului de afișare double-buffering pentru îmbunătățirea vitezei de generare a scenei. În acest scop, funcției auxInitDisplayMode îi este trimis argumentul AUX\_DOUBLE, iar la sfârșitul funcției de redesenare este apelată funcția auxSwapBuffers pentru a interzimba cele 2 buffere: cel vizibil cu cel ascuns.

### Exemplul 3: Animație folosind biblioteca AUX

```
//variabile globale
GLfloat x1; GLfloat y1; GLsizei rsize; GLfloat xstep;
GLfloat ystep; GLfloat windowHeight; GLfloat windowHeight;

//Inițializare variabile globale în funcția OnInitDialog()
//Poziția și mărimea inițială a pătratului
x1 = 100.0f;
y1 = 150.0f;
rsize = 50;
//Mărimea pasului pe Ox și Oy
xstep = 1.0f;
ystep = 1.0f;

void CALLBACK ModificaDimensiune(GLsizei w, GLsizei h)
{
    if (h==0) h=1;
    glViewport(0,0,w,h);
    glLoadIdentity();
    if (w<=h)
    {
        windowHeight = 250.0f*h/w;
        windowHeight = 250.0f;
    }
    else
```

```
{  
    windowWidth = 250.0f*w/h;  
    windowHeight = 250.0f;  
}  
//Stabilirea volumului de vedere  
glOrtho(0.0f,windowWidth,0.0f,windowHeight,1.0f,-1.0f);  
}  
  
  
void CALLBACK RedeseneazaScena(void)  
{  
    glClearColor(0.0f,0.0f,1.0f,1.0f);  
    glClear(GL_COLOR_BUFFER_BIT);  
  
    glColor3f(1.0f,0.0f,0.0f);  
    glRectf(x1,y1,x1+rsize,y1+rsize);  
    glFlush();  
    auxSwapBuffers();  
}  
  
void CALLBACK AvanseazaAnimatie(void)  
{  
    //Inversarea directiei cand s-au atins marginile (st. sau dr.)  
    if(x1>windowWidth-rsize||x1<0)  
    {  
        xstep = -xstep;  
    }  
    //Inversarea directiei cand s-au atins marginile (sus sau jos)  
    if(y1>windowHeight-rsize||y1<0)  
    {  
        ystep = -ystep;  
    }  
    //Repozitionare in fereastra  
    if(x1>windowWidth-rsize)  
        x1 = windowWidth-rsize-1;  
    if(y1>windowHeight-rsize)  
        y1 = windowHeight-rsize-1;  
    x1 += xstep;  
    y1 +=ystep;  
    //Redesenarea scenei  
    RedeseneazaScena();  
}  
  
void CTeste_OpenGLDlg::OnBnClickedButton1() //functie atasata la buton  
{  
    //Stabilirea ferestrei de lucru OpenGL  
    auxInitDisplayMode(AUX_DOUBLE|AUX_RGB4);  
    auxInitPosition(10,10,500,500);  
    auxInitWindow("Animație în OpenGL");  
  
    //Înregistrarea funcțiilor CALLBACK  
    auxReshapeFunc(ModificaDimensiune);  
    auxIdleFunc(AvanseazaAnimatie);  
    auxMainLoop(RedeseneazaScena);  
}
```

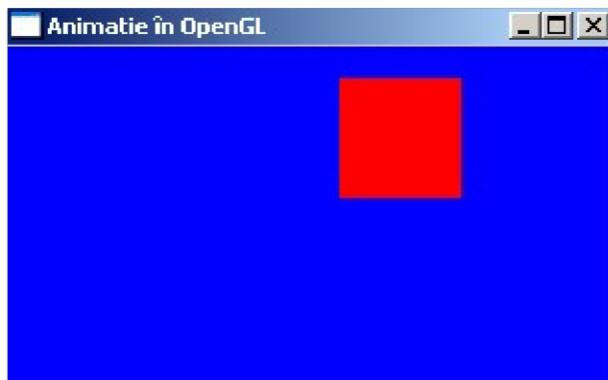


Figura 4 – Animație folosind OpenGL

## Primitive OpenGL

Toate obiectele unei scene OpenGL sunt alcătuite din forme simple, de mici dimensiuni, aranjate și combinate în diverse moduri numite ***primitive***.

**Primitivele OpenGL** sunt “cărămizile” constructive cu ajutorul cărora sunt definite toate obiectele unei scene 2D sau 3D.

Primitivele OpenGL sunt în număr de 10 și sunt identificate printr-o constantă întreagă: puncte (GL\_POINTS), linii (GL\_LINES, GL\_LINE\_STRIP, GL\_LINE\_LOOP), triunghiuri (GL\_TRIANGLES, GL\_TRIANGLE\_STRIP, GL\_TRIANGLE\_FAN), patrulatere (GL\_QUADS, GL\_QUAD\_STRIP) și poligoane (GL\_POLYGON). Primul lucru care trebuie făcut atunci când construim lumi 3D cu ajutorul OpenGL este construirea volumului de vedere, care va fi zona de lucru în spațiul 3D. În exemplele din acest laborator vom folosi ca volum de vedere un cub centrat în origine cu latura de 200 de unități (*Figura 5*), definit în funcția ModificaDimensiune din exemplul următor (exemplul 4).

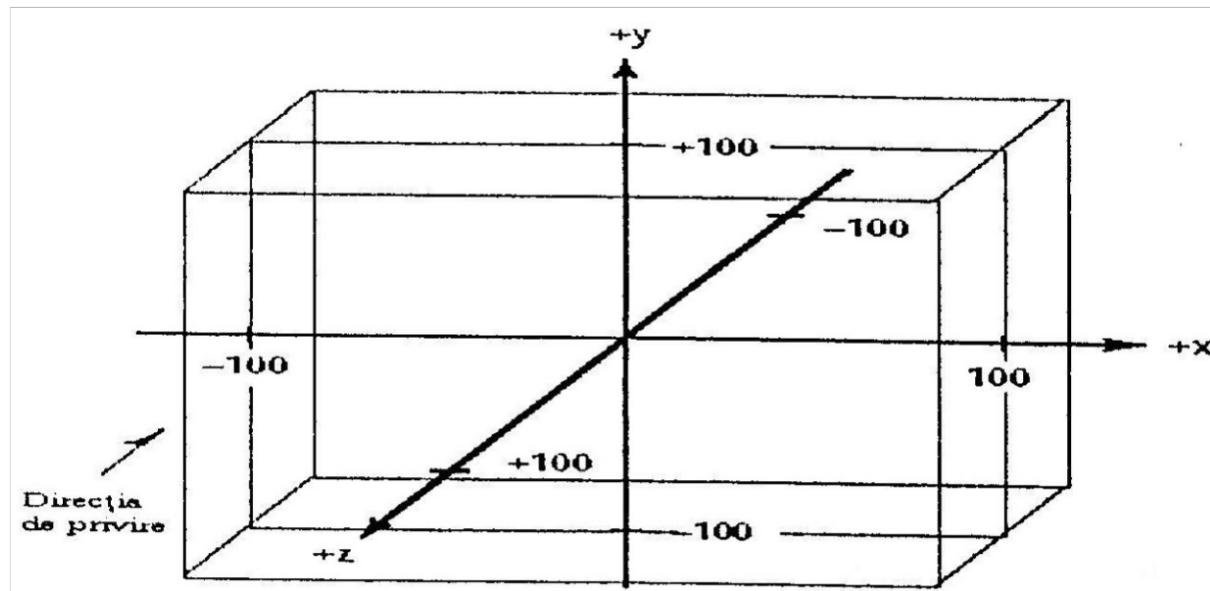


Figura 5 – Volumul de vedere OpenGL

Pentru a specifica un vârf al unei primitive, numit și **vertex** (un punct, capătul unei linii sau vârful unui poligon), vom folosi funcția `glVertex`. Biblioteca OpenGL pune la dispoziția programatorilor mai multe variante ale acestei funcții, cu două până la patru argumente. Spre exemplu, apelul

```
glVertex3f(50.0f, 50.0f, 20.0f);
```

definește un vârf la coordonatele (50, 50, 20). Un vârf are asociat un set de proprietăți compus din:

- Cordonatele (x, y, z).
- Culoarea.
- Normala în acel vârf.
- Cordonatele texturii în acel vârf.

Pentru a include una sau mai multe primitive de același tip într-o scenă se folosesc funcțiile `glBegin` și `glEnd`. Prima marchează începutul unei primitive (sau începutul unui set de primitive de același tip, de exemplu un set de puncte), în timp ce `glEnd` marchează sfârșitul primitivei.

```
glBegin(TIP_PRIMITIVA);
glVertex(...);
.....
glEnd();
```

unde `TIP_PRIMITIVA` poate fi unul dintre identificatorii de primitivă (`GL_POINTS`, `GL_LINES`, etc.). Stabilirea proprietăților primitivei ce urmează a fi desenată (dimensiunea unui punct, stilul liniei, etc.) trebuie făcută înainte de apelul `glBegin` pentru a fi luate în considerare.

## Desenare puncte în OpenGL

Exemplul ce urmează desenează trei puncte în planul xOy (z =0).

```
glBegin(GL_POINTS);
glVertex3f(0.0f, 0.0f, 0.0f);
glVertex3f(100.0f, 0.0f, 0.0f);
glVertex3f(100.0f, 100.0f, 0.0f);
glEnd();
```

Stabilirea dimensiunii în pixeli a unui punct se face utilizând funcția

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

unde `size` reprezintă dimensiunea în pixeli a punctului.

Limitele inferioară și superioară admisibile pentru dimensiunea unui punct sunt controlate de variabila de stare `GL_POINT_SIZE_RANGE`. Mai mult, nu toate valorile cuprinse în intervalul de mai sus sunt valori valide pentru dimensiunea unui punct. Intervalul dintre două valori valide este dat de variabila de stare: `GL_POINT_SIZE_GRANULARITY`.

Exemplul următor va desena o spirală compusă din 3 spire (*Figura 6*), înfășurată de-a lungul axei Oz. Pentru aceasta se vor genera punctele pe circumferința unui cerc, iar la fiecare punct generat se vor modifica adâncimea și dimensiunea punctului.

### **Exemplu 4: Crearea unei spirale din puncte de dimensiuni variabile**

```
//variabile globale
#define GL_PI 3.1415f;
#define RADIUS 50.0f;

//Construirea volumului de vedere
void CALLBACK ModificaDimensiune(GLsizei w, GLsizei h)
```

```
{  
    GLfloat nRange = 100.0f;  
  
    if (h==0) h = 1;  
  
    // Stabilirea viewportului la dimensiunea ferestrei  
    glViewport(0, 0, w, h);  
  
    // Initializeaza matricea de proiectie cu matricea identitate  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
  
    // Stabileste volumul de vedere folosind o proiectie ortografica  
    if (w<=h)  
        glOrtho(-nRange, nRange, -nRange*h/w, nRange*h/w, -nRange, nRange);  
    else  
        glOrtho(-nRange*w/h, nRange*w/h, -nRange, nRange, -nRange, nRange);  
  
    // Initializarea matricii modelului  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
}  
  
void CALLBACK DeseneazaSpirala(void  
{  
    // Coordonatele si unghiul  
    GLfloat x,y,z = -50.0f,angle;  
    GLfloat xRot = 30.f, yRot = 30.f;  
  
    // Domeniul pentru dimensiunea punctului  
    GLfloat sizes[2];  
  
    // Granularitatea dimensiunii punctului  
    GLfloat step;  
  
    // Dimensiunea curenta  
    GLfloat curSize;  
  
    glClear(GL_COLOR_BUFFER_BIT);  
    glColor3f(1.0f, 0.0f, 0.0f);  
  
    // Obtinerea domeniului si granularitatii valide pentru dimensiunea  
    // punctului  
    glGetFloatv(GL_POINT_SIZE_RANGE,sizes);  
    glGetFloatv(GL_POINT_SIZE_GRANULARITY,&step);  
  
    // Stabileste dimensiunea initiala a punctului  
    curSize = sizes[0];  
  
    // Rotirea spiralei  
    glPushMatrix();  
    glRotatef(xRot, 1.0f, 0.0f, 0.0f);  
    glRotatef(yRot, 0.0f, 1.0f, 0.0f);  
  
    // Genereaza trei spire  
    for(angle=0.0f; angle<=(2.0f*GL_PI)*3.0f; angle+=0.1f, z+=0.5f,  
    curSize+=step)  
    {  
        // Calculeaza coordonatele x, y pe un cerc  
        x = RADIUS*(GLfloat)sin(angle);  
        y = RADIUS*(GLfloat)cos(angle);  
    }  
}
```

```

    // Stabileste dimensiunea punctului
    glPointSize(curSize);

    // Deseneaza punctul
    glBegin(GL_POINTS);
    glVertex3f(x, y, z);
    glEnd();
}

// Restaurare sistem de coordonate
glPopMatrix();
auxSwapBuffers();
glFlush();
}

void CTeste_OpenGLDlg::OnBnClickedButton1()
{
    // Stabilirea fereastrăi de lucru OpenGL
    auxInitDisplayMode(AUX_DOUBLE|AUX_RGB);
    auxInitPosition(10, 10, 500, 500);
    auxInitWindow("Spirala - GL_POINTS");

    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();

    // Înregistrarea funcțiilor CALLBACK
    auxReshapeFunc(ModificaDimensiune);
    auxMainLoop(DeseneazaSpirala);
}

```

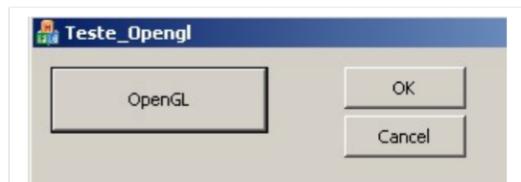
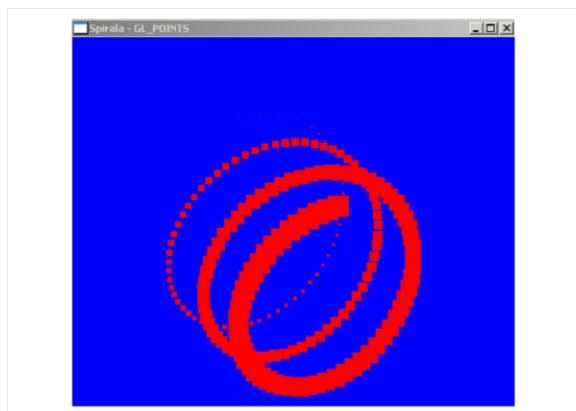


Figura 6 Spirala creată din puncte

### **Întrebări:**

### **Exerciții:**

[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/opengl/openglstart\\_9uw5.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/opengl/openglstart_9uw5.asp)  
[www.opengl.org](http://www.opengl.org)