Lucrare de Laborator OpenGL 3D

In laboratoarele anterioare am observat modul de lucru în 2D in Opengl. In aceasta lucrare vom vedea modul de lucru în Opengl 3D. Vom incepe cu un exemplu simplu - desenarea unei piramide și a unui cub. Creați un proiect nou în Opengl și copiati codul din exemplul de mai jos. Rulati programul și observati ce se intampla. Reveniți apoi și observați explicațiile legate de comenzile utilizate.

Exemplul 1:

Sursa:

https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_Examples.html

```
/*
 * OGL01Shape3D.cpp: 3D Shapes
 */
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h

/* Global variables */
char title[] = "3D Shapes";

/* Initialize OpenGL Graphics */
void initGL() {
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
   glClearDepth(1.0f);                   // Set background depth to farthest
   glEnable(GL_DEPTH_TEST);   // Enable depth testing for z-culling
   glDepthFunc(GL_LEQUAL);    // Set the type of depth-test
   glShadeModel(GL_SMOOTH);   // Enable smooth shading
   glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);  // Nice perspective corrections
}

/* Handler for window-repaint event. Called back when the window first appears and
   whenever the window needs to be re-painted. */
void display() {
```

```
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear color and depth
buffers
   glMatrixMode(GL_MODELVIEW);     // To operate on model-view matrix

   // Render a color-cube consisting of 6 quads with different colors
   glLoadIdentity();                 // Reset the model-view matrix
   glTranslatef(1.5f, 0.0f, -7.0f);  // Move right and into the screen

   glBegin(GL_QUADS);                // Begin drawing the color cube with 6 quads
      // Top face (y = 1.0f)
      // Define vertices in counter-clockwise (CCW) order with normal pointing out
      glColor3f(0.0f, 1.0f, 0.0f);     // Green
      glVertex3f( 1.0f, 1.0f, -1.0f);
      glVertex3f(-1.0f, 1.0f, -1.0f);
      glVertex3f(-1.0f, 1.0f,  1.0f);
      glVertex3f( 1.0f, 1.0f,  1.0f);

      // Bottom face (y = -1.0f)
      glColor3f(1.0f, 0.5f, 0.0f);     // Orange
      glVertex3f( 1.0f, -1.0f,  1.0f);
      glVertex3f(-1.0f, -1.0f,  1.0f);
      glVertex3f(-1.0f, -1.0f, -1.0f);
      glVertex3f( 1.0f, -1.0f, -1.0f);

      // Front face  (z = 1.0f)
      glColor3f(1.0f, 0.0f, 0.0f);     // Red
      glVertex3f( 1.0f,  1.0f, 1.0f);
      glVertex3f(-1.0f,  1.0f, 1.0f);
      glVertex3f(-1.0f, -1.0f, 1.0f);
      glVertex3f( 1.0f, -1.0f, 1.0f);

      // Back face (z = -1.0f)
      glColor3f(1.0f, 1.0f, 0.0f);     // Yellow
      glVertex3f( 1.0f, -1.0f, -1.0f);
      glVertex3f(-1.0f, -1.0f, -1.0f);
      glVertex3f(-1.0f,  1.0f, -1.0f);
      glVertex3f( 1.0f,  1.0f, -1.0f);

      // Left face (x = -1.0f)
      glColor3f(0.0f, 0.0f, 1.0f);     // Blue
      glVertex3f(-1.0f,  1.0f,  1.0f);
      glVertex3f(-1.0f,  1.0f, -1.0f);
      glVertex3f(-1.0f, -1.0f, -1.0f);
      glVertex3f(-1.0f, -1.0f,  1.0f);
```

```
      // Right face (x = 1.0f)
      glColor3f(1.0f, 0.0f, 1.0f);     // Magenta
      glVertex3f(1.0f,  1.0f, -1.0f);
      glVertex3f(1.0f,  1.0f,  1.0f);
      glVertex3f(1.0f, -1.0f,  1.0f);
      glVertex3f(1.0f, -1.0f, -1.0f);
   glEnd();  // End of drawing color-cube

   // Render a pyramid consists of 4 triangles
   glLoadIdentity();                  // Reset the model-view matrix
   glTranslatef(-1.5f, 0.0f, -6.0f);  // Move left and into the screen

   glBegin(GL_TRIANGLES);           // Begin drawing the pyramid with 4 triangles
      // Front
      glColor3f(1.0f, 0.0f, 0.0f);     // Red
      glVertex3f( 0.0f, 1.0f, 0.0f);
      glColor3f(0.0f, 1.0f, 0.0f);     // Green
      glVertex3f(-1.0f, -1.0f, 1.0f);
      glColor3f(0.0f, 0.0f, 1.0f);     // Blue
      glVertex3f(1.0f, -1.0f, 1.0f);

      // Right
      glColor3f(1.0f, 0.0f, 0.0f);     // Red
      glVertex3f(0.0f, 1.0f, 0.0f);
      glColor3f(0.0f, 0.0f, 1.0f);     // Blue
      glVertex3f(1.0f, -1.0f, 1.0f);
      glColor3f(0.0f, 1.0f, 0.0f);     // Green
      glVertex3f(1.0f, -1.0f, -1.0f);

      // Back
      glColor3f(1.0f, 0.0f, 0.0f);     // Red
      glVertex3f(0.0f, 1.0f, 0.0f);
      glColor3f(0.0f, 1.0f, 0.0f);     // Green
      glVertex3f(1.0f, -1.0f, -1.0f);
      glColor3f(0.0f, 0.0f, 1.0f);     // Blue
      glVertex3f(-1.0f, -1.0f, -1.0f);

      // Left
      glColor3f(1.0f,0.0f,0.0f);       // Red
      glVertex3f( 0.0f, 1.0f, 0.0f);
      glColor3f(0.0f,0.0f,1.0f);       // Blue
      glVertex3f(-1.0f,-1.0f,-1.0f);
      glColor3f(0.0f,1.0f,0.0f);       // Green
      glVertex3f(-1.0f,-1.0f, 1.0f);
   glEnd();   // Done drawing the pyramid
```

```
   glutSwapBuffers();  // Swap the front and back frame buffers (double buffering)
}

/* Handler for window re-size event. Called back when the window first appears and
   whenever the window is re-sized with its new width and height */
void reshape(GLsizei width, GLsizei height) {  // GLsizei for non-negative integer
   // Compute aspect ratio of the new window
   if (height == 0) height = 1;                // To prevent divide by 0
   GLfloat aspect = (GLfloat)width / (GLfloat)height;

   // Set the viewport to cover the new window
   glViewport(0, 0, width, height);

   // Set the aspect ratio of the clipping volume to match the viewport
   glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
   glLoadIdentity();             // Reset
   // Enable perspective projection with fovy, aspect, zNear and zFar
   gluPerspective(45.0f, aspect, 0.1f, 100.0f);
}

/* Main function: GLUT runs as a console application starting at main() */
int main(int argc, char** argv) {
   glutInit(&argc, argv);            // Initialize GLUT
   glutInitDisplayMode(GLUT_DOUBLE); // Enable double buffered mode
   glutInitWindowSize(640, 480);   // Set the window's initial width & height
   glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
   glutCreateWindow(title);          // Create window with the given title
   glutDisplayFunc(display);       // Register callback handler for window re-paint
event
   glutReshapeFunc(reshape);       // Register callback handler for window re-size
event
   initGL();                       // Our own OpenGL initialization
   glutMainLoop();                 // Enter the infinite event-processing loop
   return 0;
}
```

Am observat pana acum ca programul e functional - deseneaza o piramida si un cub (Figura 1). Unele comenzi ar trebui sa fie deja familiare intrucat le-am vazut in laboratoarele anterioare. In cele ce urmeaza vom discuta programul pas cu pas, punctand elementele cele mai importante, dar si pe cele noi.

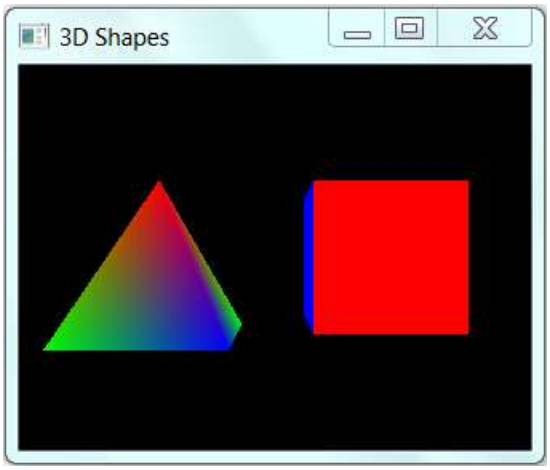


Figura 1

GLUT Setup - main()

Program contine functiile initGL(), display() si reshape().

Astfel, in main avem urmatoarele comenzi de baza:

- glutInit(&argc, argv);

  Inițializează GLUT.

- glutInitWindowSize(640, 480);

  glutInitWindowPosition(50, 50);

  glutCreateWindow(title);

  Crează o fereastră cu titlu, dimensiune și poziție predefinite.

- glutDisplayFunc(display);
- glutReshapeFunc(reshape);
- glutInitDisplayMode(GLUT_DOUBLE)
- initGL();
- glutMainLoop();

Majoritatea comenzilor de aici au fost deja discutate în laboratoarele anterioare, prin urmare nu le vom rediscuta.

Operații de inițializare

Funcția initGL() realizează inițializările. Este apelată din main() o singură data.

glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Setează culoarea de fundal la negru și opac

glClearDepth(1.0f); // Setează adâncimea fundalului

În display() apelăm *glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)* pentru setarea culorii de fundal la negru și opac, și adâncimea fundalului va fi cea mai îndepărtată.
glEnable(GL_DEPTH_TEST); // Permite testarea în adâncime pentru z.

glDepthFunc(GL_LEQUAL); // Setează tipul de testare în adâncime.

Este necesar să permitem testarea în adâncime pentru a șterge suprafețele ascunse și pentru a seta funcția de testare în adâncime.

glShadeModel(GL_SMOOTH); // Umbre fine (interpolarea culorilor)

glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST); // Comanda realizează corectările necesare pentru un afișaj corect – de multe ori este necesar să găsim un echilibru între viteza de procesare și calitatea afișajului.

Definirea cubului și a piramidei

În OpenGL un obiect este format din primitive, cum ar fi triunghi, pătrat, poligon, punct sau linie. O primitivă este definită prin unul sau mai multe vertexuri. Cubul este format din șase pătrate – cele șase fețe. Fiecare pătrat este format din patru vertexuri în sens invers acelor de ceasornic. Toate patru vertexurile au aceeași culoare. Cubul este definit în sistem de coordonate proprii (sistem de coordonate local) având originiea în centrul cubului.

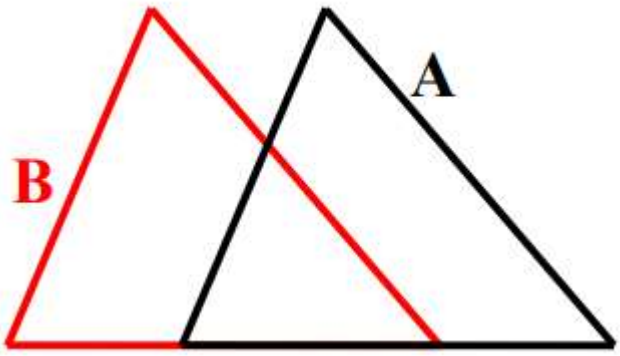
La fel, piramida este formată din patru triunghiuri (fără bază). Fiecare triunghi este format din 3 vertexuri definite în sens invers acelor de ceasornic. Fiecare vertex principal al piramidei va avea o altă culoare. Culorile triunghiurilor sunt interpolate. Ca și în cazul cubului, piramida ese definită în sistemul propriu de coordonate.

Vor fi necesare transformări geometrice pentru a aduce cubul și piramida în același sistem de coordonate (sistem global de coordonate).

- TRANSFORMAREA DE MODELARE

Comenzile pentru transformarea de modelare a cubului :
glLoadIdentity(); // Reset model-view matrix
glTranslatef(1.5f, 0.0f, -7.0f); // Move right and into the screen
Comenzile pentru transformarea de modelare a piramidei :
glLoadIdentity(); // Reset model-view matrix
glTranslatef(-1.5f, 0.0f, -6.0f); // Move left and into the screen

Exemplu pentru transformarea de modelare a unui triunghi în 2D:



- **TRANSFORMAREA DE VIZUALIZARE**

Dorim poziționarea observatorului în scenă prin comanda:
gluLookAt(ox,oy,oz,cx,cy,cz,upx,upy,upz)
Poziția default este la *gluLookAt(0.0, 0.0, 0.0, 0.0, 0.0, -100.0, 0.0, 1.0, 0.0).* În exemplul de față ne aflăm deja la această poziție, prin urmare nu mai sunt necesare modificări.

- **TRANSFORMAREA ÎN VIEWPORT**

```
void reshape(GLsizei width, GLsizei height) {
  glViewport(0, 0, width, height);
```

Sistemul reapelează reshape() atunci când fereastra apare prima dată și oridecâteori fereastra este redimensionată. Dorim ca spațiul nostru viewport să acopere întreaga fereastră, din colțul stânga jos (0,0).

- **TRANSFORMAREA DE PROIECȚIE**

```
GLfloat aspect = (GLfloat)width / (GLfloat)height; // Compute aspect ratio of window
glMatrixMode(GL_PROJECTION); // To operate on the Projection matrix
glLoadIdentity(); // Reset
gluPerspective(45.0f, aspect, 0.1f, 100.0f); // Perspective projection: fovy, aspect, near, far
```

Ecranul are un camp limitat de vizualizare. Proiecția determină care vedere va fo capturată de cameră. Astfel, există două tipuri de proiecții: cea în perspectivă și cea paralelă. În cazul proiecției de perspectivă, obiectele mai îndepărtate de cameră apar mai mici decât cele aflate mai aproape de cameră. În cazul proiecției paralele, obiectele apar la fel indiferent unde se află pe axa Z. Acesta este practic un caz special de proiecție de perspectivă în care camera se află foarte departe. În acest laborator vom lucra cu proiecția în perspectivă.

Astfel, comanda gluPerspective() este folosită pentru a permite proiecția în perspectivă și pentru a seta unghiul dintre planul de jos și cel de sus (denumit fovy), valoarea lățime/înălțime, zAproape și zDeparte. În exemplul nostru, fovy are valoarea de 45 grade.

Raportul lățime/înălțime este setat ca viewport pentru a evita distorsiunile. zAproape va fi 0.1 iar zDeparte va fi 100.

Exemplul 2 – animații în 3D

În exemplul următor vom adăuga animații astfel încât cubul desenat la exemplul 1 să se rotească.

```
/*
 * OGL02Animation.cpp: 3D Shapes with animation
 */
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h

/* Global variables */
char title[] = "3D Shapes with animation";
GLfloat anglePyramid = 0.0f;  // Rotational angle for pyramid [NEW]
GLfloat angleCube = 0.0f;     // Rotational angle for cube [NEW]
int refreshMills = 15;        // refresh interval in milliseconds [NEW]

/* Initialize OpenGL Graphics */
void initGL() {
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
   glClearDepth(1.0f);                   // Set background depth to farthest
   glEnable(GL_DEPTH_TEST);   // Enable depth testing for z-culling
   glDepthFunc(GL_LEQUAL);    // Set the type of depth-test
   glShadeModel(GL_SMOOTH);   // Enable smooth shading
   glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);  // Nice perspective corrections
}

/* Handler for window-repaint event. Called back when the window first appears and
   whenever the window needs to be re-painted. */
void display() {
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear color and depth
buffers
   glMatrixMode(GL_MODELVIEW);     // To operate on model-view matrix

   // Render a color-cube consisting of 6 quads with different colors
   glLoadIdentity();                 // Reset the model-view matrix
   glTranslatef(1.5f, 0.0f, -7.0f);  // Move right and into the screen
   glRotatef(angleCube, 1.0f, 1.0f, 1.0f);  // Rotate about (1,1,1)-axis [NEW]

   glBegin(GL_QUADS);                // Begin drawing the color cube with 6 quads
      // Top face (y = 1.0f)
      // Define vertices in counter-clockwise (CCW) order with normal pointing out
      glColor3f(0.0f, 1.0f, 0.0f);     // Green
```

```
      glVertex3f( 1.0f, 1.0f, -1.0f);
      glVertex3f(-1.0f, 1.0f, -1.0f);
      glVertex3f(-1.0f, 1.0f,  1.0f);
      glVertex3f( 1.0f, 1.0f,  1.0f);

      // Bottom face (y = -1.0f)
      glColor3f(1.0f, 0.5f, 0.0f);     // Orange
      glVertex3f( 1.0f, -1.0f,  1.0f);
      glVertex3f(-1.0f, -1.0f,  1.0f);
      glVertex3f(-1.0f, -1.0f, -1.0f);
      glVertex3f( 1.0f, -1.0f, -1.0f);

      // Front face  (z = 1.0f)
      glColor3f(1.0f, 0.0f, 0.0f);     // Red
      glVertex3f( 1.0f,  1.0f, 1.0f);
      glVertex3f(-1.0f,  1.0f, 1.0f);
      glVertex3f(-1.0f, -1.0f, 1.0f);
      glVertex3f( 1.0f, -1.0f, 1.0f);

      // Back face (z = -1.0f)
      glColor3f(1.0f, 1.0f, 0.0f);     // Yellow
      glVertex3f( 1.0f, -1.0f, -1.0f);
      glVertex3f(-1.0f, -1.0f, -1.0f);
      glVertex3f(-1.0f,  1.0f, -1.0f);
      glVertex3f( 1.0f,  1.0f, -1.0f);

      // Left face (x = -1.0f)
      glColor3f(0.0f, 0.0f, 1.0f);     // Blue
      glVertex3f(-1.0f,  1.0f,  1.0f);
      glVertex3f(-1.0f,  1.0f, -1.0f);
      glVertex3f(-1.0f, -1.0f, -1.0f);
      glVertex3f(-1.0f, -1.0f,  1.0f);

      // Right face (x = 1.0f)
      glColor3f(1.0f, 0.0f, 1.0f);     // Magenta
      glVertex3f(1.0f,  1.0f, -1.0f);
      glVertex3f(1.0f,  1.0f,  1.0f);
      glVertex3f(1.0f, -1.0f,  1.0f);
      glVertex3f(1.0f, -1.0f, -1.0f);
   glEnd();  // End of drawing color-cube

   glutSwapBuffers();  // Swap the front and back frame buffers (double buffering)

   // Update the rotational angle after each refresh [NEW]
```

```
   angleCube -= 0.15f;
}

/* Called back when timer expired [NEW] */
void timer(int value) {
   glutPostRedisplay();      // Post re-paint request to activate display()
   glutTimerFunc(refreshMills, timer, 0); // next timer call milliseconds later
}

/* Handler for window re-size event. Called back when the window first appears and
   whenever the window is re-sized with its new width and height */
void reshape(GLsizei width, GLsizei height) {  // GLsizei for non-negative integer
   // Compute aspect ratio of the new window
   if (height == 0) height = 1;                // To prevent divide by 0
   GLfloat aspect = (GLfloat)width / (GLfloat)height;

   // Set the viewport to cover the new window
   glViewport(0, 0, width, height);

   // Set the aspect ratio of the clipping volume to match the viewport
   glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
   glLoadIdentity();             // Reset
   // Enable perspective projection with fovy, aspect, zNear and zFar
   gluPerspective(45.0f, aspect, 0.1f, 100.0f);
}

/* Main function: GLUT runs as a console application starting at main() */
int main(int argc, char** argv) {
   glutInit(&argc, argv);            // Initialize GLUT
   glutInitDisplayMode(GLUT_DOUBLE); // Enable double buffered mode
   glutInitWindowSize(640, 480);   // Set the window's initial width & height
   glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
   glutCreateWindow(title);          // Create window with the given title
   glutDisplayFunc(display);       // Register callback handler for window re-paint
event
   glutReshapeFunc(reshape);       // Register callback handler for window re-size
event
   initGL();                       // Our own OpenGL initialization
   glutTimerFunc(0, timer, 0);     // First timer call immediately [NEW]
   glutMainLoop();                 // Enter the infinite event-processing loop
   return 0;
}
```

Vedem că au apărut unele instrucțiuni noi.
GLfloat angleCube = 0.0f; //Variabila în care reținem unghiul de rotație
int refreshMills = 15; //Inervalul de reîmprospătare în milisecunde

```
void timer(int value) {
  glutPostRedisplay(); // cere redesenare pentru a activa funcția display()
  glutTimerFunc(refreshMills, timer, 0); // următorul apel al funcției timer
}
```

Pentru animație, definim o funcție numită *timer()* care trimite o cerere de redesanre pentru a activa funcția *display()* când timer-ul a expirat, iar apoi îl rulează din nou. Apelăm *timer()* pentru prima oară în main() prin *glutTimerFunc(0, timer, 0).*

```
glRotatef(angleCube, 1.0f, 1.0f, 1.0f); // Rotește cubul cu (1,1,1) pe axe
.....

angleCube -= 0.15f; // update la valoarea unghiului cubului
```

În funcția *display()* rotim cubul pornind de la unghiul său de rotație, iar apoi salvăm noua valoare a unghiului.

Exerciții:

1. Rulați exemplele 1 și 2. Creați câte un proiect diferit pentru fiecare exemplu.
2. Separați programul de la exemplul 1 în două programe separate: unul în care să desenați doar cubul, altul în care să desenați doar piramida.
3. Realizați un program care să deseneze o piramidă și să o rotească.