Laborator 6

MODELAREA OBIECTELOR

În grafica pe calculator, imaginea unui obiect tridimensional se generaza pornind de *modelul obiectului*, care este o descriere matematica a proprietatilor obiectului.

Proprietatile obiectelor tridimensionale care se modeleaza în aplicatiile grafice se pot împarti în doua categorii: *forma* si *atribute de aspect.* Informatia de forma a unui obiect este diferita de celelalte atribute ale obiectului, deoarece forma este aceea care determina modul în care obiectul apare în redarea grafica si toate celelalte atribute se coreleaza cu forma obiectului (de exemplu, culoarea se specifica pentru fiecare element de suprafata a obiectului).

Din punct de vedere al formei, obiectele tridimensionale reprezentate în grafica pe calculator pot fi obiecte *solide* sau obiecte *deformabile.* Un solid este un obiect tridimensional a carui forma si dimensiuni nu se modifica în functie de timp sau de pozitia în scena (proprietatea de forma volumetrica invarianta). Majoritatea aplicatiilor grafice se bazeaza pe scene compuse din solide, dar exista si aplicatii în care obiectele reprezentate îsi modifica forma si dimensiunile într-un mod predefinit sau ca urmare a unor actiuni interactive (de exemplu, în simulari ale interventiilor chirurgicale). Chiar si reprezentarea unor astfel de obiecte (obiecte deformabile) se bazeaza pe un model al unui solid care se modifica în cursul experimentului de realitate virtuala. În lucrarea de fata se vor prezenta modele ale solidelor care stau la baza majoritatii prelucrarilor grafice.

Modelarea solidelor este o tehnica de proiectare, vizualizare si analiza a modului în care obiectele reale se reprezinta în calculator. În ordinea importantei si a frecventei de utilizare, metodele de modelare si reprezentare a obiectelor sunt urmatoarele:

1. *Modelarea poligonala.* În acesta forma de reprezentare, obiectele sunt aproximate printr-o retea de fete care sunt poligoane planare.
2. *Modelarea prin retele de petice parametrice bicubice (bicubic parametric patches).* Obiectele sunt aproximate prin retele de elemente spatiale numite petice. Acestea sunt reprezentate prin polinoame cu doua variabile parametrice, în mod obisnuit cubice.
3. *Modelarea prin compunerea obiectelor (Constructive Solid Geometry - CSG).* Obiectele sunt reprezentate prin colectii de obiecte elementare, cum sunt cilindri, sfere, poliedre.
4. *Modelarea prin divizare spatiala.* Obiectele sunt încorporate în spatiu, prin atribuirea unei etichete fiecarui element spatial, în functie de obiectul care ocupa elementul respectiv.

Pentru început va rog sŁ va asigurați ca bibloteca OpenGl este descŁrcatŁ ki funcționalŁ prin introducerea urmŁtorului fragment de cod, în care se deschide o fereastrŁ graficŁ în se configureaza un pŁtrat.

De asemenea, apare printul " `Buna ziua dragi studenti` " în fereastra consolei.Codul este o specificare ilustrativŁ a utilizŁrii bibliotecii la deschiderea ferestrei grafice în gestionarea buclei de afiare.

```
#include <stdio.h>
#include <GL/glut.h>

    void display(void)
    {
     glClear( GL_COLOR_BUFFER_BIT);
     glColor3f(0.0, 1.0, 0.0);
     glBegin(GL_POLYGON);
      glVertex3f(2.0, 4.0, 0.0);
      glVertex3f(8.0, 4.0, 0.0);
      glVertex3f(8.0, 6.0, 0.0);
      glVertex3f(2.0, 6.0, 0.0);
     glEnd();
     glFlush();
    }

    int main(int argc, char **argv)
    {
     printf("Buna ziua dragi studenti\n");
     glutInit(&argc, argv);
     glutInitDisplayMode ( GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

     glutInitWindowPosition(100,100);
     glutInitWindowSize(300,300);
     glutCreateWindow ("dreptunghi");

     glClearColor(0.0, 0.0, 0.0, 0.0);         // fundal negru
     glMatrixMode(GL_PROJECTION);              // setarea proiectiei de vizualizare
     glLoadIdentity();                           // incarca matricea de identitate
     glOrtho(0.0, 10.0, 0.0, 10.0, -1.0, 1.0);   // setarea spatiului de vizualizare 10x10x2

     glutDisplayFunc(display);
     glutMainLoop();

     return 0;
    }
```

Următorul exemplu arată cum să aplicăm funcții de transformare pe o formă (pătrat) colorată în OpenGL / GLUT prin utilizarea glRotatef (), glTranslatef () ,şi a funcțiilor de cronometrare.

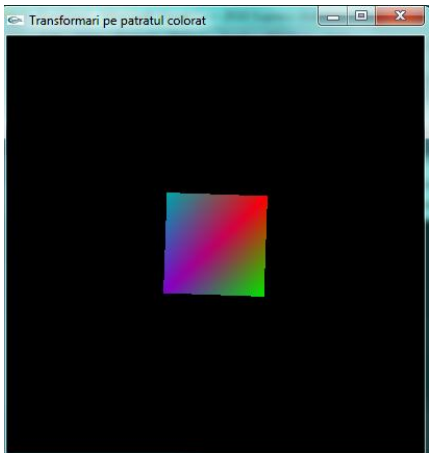
Se vor schimba valorile glRotatef și glTranslate pentru a experimenta diferite transformări aplicate pătratului. Forma este desenată în 2D.

Acest program transforma un pătrat de culoare cu un background întunecat prin utilizarea funcțiilor de transformare

```
/*
/*
* TransformariPatratColorat.cpp
*
*   Pe acest program au fost aplicate transformari unui patrat colorat pe fundal negru
*   utilizand
*   Functii de Transformare  : glRotatef(),glTranslatef()
```

```
*   Functia de Timp  : glutTimerFunc() a fost chemata la fiecare 25 milisecunde
*
* Pentru a complila:
*   gcc TransformariPatratColorat.cpp -lglut -lGL -lGLU
*/
#include <iostream>
#include <stdlib.h>
#include <GL/glut.h>

using namespace std;

//Apelata cand o tasta este apasata
void handleKeypress(unsigned char key, int x, int y) {
    switch (key) {
        case 27: //tasta Esc
            exit(0);
    }
}

//Initializare 3D rendering
void initRendering() {
    glEnable(GL_DEPTH_TEST);
}

//Apelata cand fereastra este dimensionata
void handleResize(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (double)w / (double)h, 1.0, 200.0);
}

float _angle = 30.0f;
float _cameraAngle = 0.0f;

//Desenarea scenei 3D
void drawScene() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW); //Comutare la desenul in vederea de perspectiva
    glLoadIdentity(); //Resetarea la vederea in perspectiva
    glRotatef(-_cameraAngle, 0.0f, 1.0f, 0.0f); //Roteste camera
    glTranslatef(0.0f, 0.0f, -5.0f); //Translateaza 5 unitati

    glPushMatrix(); //Salvati transformarile efectuate
    glTranslatef(0.0f, 0.0f, 0.0f); //Translateaza
    glRotatef(_angle, 0.0f, 0.0f, 1.0f); //Roteste dupa axa z

    /* Deseneaza un poligon de o unitate */
    glBegin(GL_POLYGON);
    glColor3f(0.5f, 0.0f, 0.8f);
    glVertex3f(-0.5f, -0.5f, 0.0f);
    glColor3f(0.0f, 0.9f, 0.0f);
    glVertex3f(0.5f, -0.5f, 0.0f);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f(0.5f, 0.5f, 0.0f);
    glColor3f(0.0f, 0.65f, 0.65f);
    glVertex3f(-0.5f, 0.5f, 0.0f);

    glEnd();

    glutSwapBuffers();
}

void update(int value) {
    _angle += 2.0f;
    if (_angle > 360) {
        _angle -= 360;
    }
```

```
    glutPostRedisplay(); //Comunica GLUT ca afisajul s-a schimbat

    //Comunica GLUT pentru a apela din nou modificare în 25 de milisecunde
    glutTimerFunc(25, update, 0);
}

int main(int argc, char** argv) {
    //Initializare GLUT
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(400, 400);

    //Creare fereastra
    glutCreateWindow("Transformari pe patratul colorat");
    initRendering();

    //Setarea  functiilor de afisare
    glutDisplayFunc(drawScene);
    glutKeyboardFunc(handleKeypress);
    glutReshapeFunc(handleResize);

    //Adauga timpul
    glutTimerFunc(25, update, 0);

    glutMainLoop();
}
```



MODELAREA POLIGONALA A OBIECTELOR

Modelarea poligonala, în care un obiect este reprezentat printr-o retea de poligoane planare care aproximeaza suprafata de frontiera (boundary representation ï B-rep), este forma ñclasicaò folosita în grafica pe calculator. Motivele utilizarii extinse a acestei forme de reprezentare sunt usurinta de modelare si posibilitatea de redare rapida a imaginii obiectelor.
Pentru obiectele reprezentate poligonal s-au dezvoltat algoritmi de redare eficienti, care asigura calculul umbririi, eliminarea suprafetelor ascunse, texturare, anti-aliasing, frecvent implementati hardware în sistemele grafice. În reprezentarea poligonala, un obiect tridimensional este compus dintr-o colectie de fete, fiecare fata fiind o suprafata plana reprezentata printr-un poligon.

Triangularizarea poligoanelor. O proprietate importanta a poligoanelor este proprietatea de triangularizare. Se demonstreaza ca orice poligon P poate fi împartit în triunghiuri prin adaugarea a zero sau mai multe diagonale. O diagonala a unui poligon este un segment de dreapta între doua vârfuri a si b, astfel încât segmentul ab nu atinge linia poligonala ŒP în alte puncte decât vârfurile a si b, de început si de sfârsit ale segmentului.
Teorema triangularizarii, care asigura ca orice poligon poate fi divizat în triunghiuri (care sunt sigur suprafete plane), reprezinta suportul celei mai eficiente metode de generare (redare) a imaginii obiectelor tridimensionale: obiectele se reprezinta prin fete poligonale, fiecare poligon se descompune în triunghiuri si triunghiurile sunt generate prin algoritmi implementati hardware.
REPREZENTAREA POLIEDRELOR
În modelarea si reprezentarea prin suprafata de frontiera, obiectele sunt aproximate prin poliedre si modelul lor este reprezentat prin suprafata poliedrului, compusa dintr-o colectie de poligoane.Un poliedru reprezinta generalizarea în spatiul tridimensional a unui poligon din planul al: poliedrul este o regiune finita a spatiului a carui suprafata de frontiera este compusa dintr-un numar finit de fete poligonale plane.
Suprafata de frontiera a unui poliedru contine trei tipuri de elemente geometrice: vârfurile (punctele), care sunt zero-dimensionale, muchiile (segmentele),care sunt unidimensionale si fetele (poligoanele), care sunt bidimensionale (fig. 2)

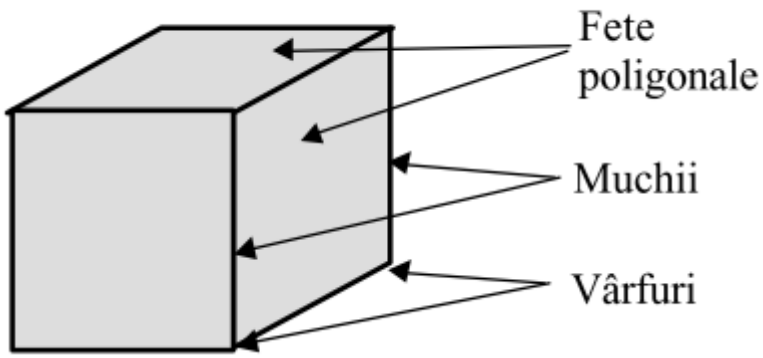


Fig. 2 Reprezentarea prin suprafata de frontiera a unui poliedru

In exemplul de mai jos se modeleazŁ **un tor.**

```
// afisare lista .cpp

#include <math.h>
#include <GL/glut.h>

GLuint theTorus;

/* Desenarea unui tor */
static void torus(int numc, int numt)
{
    float M_PI = 3.1415926;
    int i, j, k;
    double s, t, x, y, z, twopi;
    twopi = 2 * M_PI;
    for (i = 0; i < numc; i++)
    {
```

```
        glBegin(GL_LINE_STRIP);
        for (j = 0; j <= numt; j++)
        {
            for (k = 1; k >= 0; k--)
            {
                s = (i + k) % numc + 0.5;
                t = j % numt;
                x = (1.5+.5*cos(s*twopi/numc))*cos(t*twopi/numt);
                y = (1.5+.5*cos(s*twopi/numc))*sin(t*twopi/numt);
                z = .5 * sin(s * twopi / numc);
                glVertex3f(x, y, z);
            }
        }
        glEnd();
    }
}

/* Creati lista de afisare pentru Tor si initializati starea*/
static void init(void)
{
    theTorus = glGenLists (1);
    glNewList(theTorus, GL_COMPILE);
        torus(10, 20);
    glEndList();
    glShadeModel(GL_FLAT);
    glClearColor(0.2, 0.0, 0.2, 0.0);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f (0.0, 1.0, 1.0);
    glCallList(theTorus);
    glFlush();
}

void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(30, (GLfloat) w/(GLfloat) h, 1.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
}

/* Roteste dupa axa-x cand tastezi "x" ; roteste dupa axa-y cand
tastezi "y" tasteaza ; "i" pentru a readuce torul la vederea initiala */
void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
        case 'x':
        case 'X':
            glRotatef(10.,1.0,0.0,0.0);
            glutPostRedisplay();
        break;
```

```
        case 'y':
        case 'Y':
            glRotatef(10.,0.0,1.0,0.0);
            glutPostRedisplay();
        break;
        case 'i':
        case 'I':
            glLoadIdentity();
            gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
            glutPostRedisplay();
        break;
        case 27:
            exit(0);
        break;
    }
}

void main(int argc, char **argv)
{
    glutInitWindowSize(700, 700);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow(argv[0]);
    init();
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutDisplayFunc(display);
    glutMainLoop();
}
```

In urmŁtorul exemplul se modeleazŁ **un icosaedru.**

.

```
// openGL_icosaedru
//

#include <gl/glut.h>


#define X .525731112119133606
#define Z .850650808352039932

static GLfloat vdata[12][3] =
{
    {-X, 0.0, Z}, {X, 0.0, Z}, {-X, 0.0, -Z}, {X, 0.0, -Z},
    {0.0, Z, X}, {0.0, Z, -X}, {0.0, -Z, X}, {0.0, -Z, -X},
    {Z, X, 0.0}, {-Z, X, 0.0}, {Z, -X, 0.0}, {-Z, -X, 0.0}
};

static GLuint tindices[20][3] =
{
    {0,4,1}, {0,9,4}, {9,5,4}, {4,5,8}, {4,8,1},
    {8,10,1}, {8,3,10}, {5,3,8}, {5,2,3}, {2,7,3},
    {7,10,3}, {7,6,10}, {7,11,6}, {11,0,6}, {0,1,6},
    {6,1,10}, {9,0,11}, {9,11,2}, {9,2,5}, {7,2,11}
};
```

```
void render(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_TRIANGLES);
     for (int i = 0; i < 20; i++)
     {
        glColor3f(1.0,0.0,0.0);                    // vertex rosu
        glVertex3fv(&vdata[tindices[i][0]][0]);
        glColor3f(0.0,1.0,0.0);                    // vertex verde
        glVertex3fv(&vdata[tindices[i][1]][0]);
        glColor3f(0.0,0.0,1.0);                    // vertex albastru
        glVertex3fv(&vdata[tindices[i][2]][0]);
     }
  glEnd();
  glFlush();
}

void main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DEPTH | GLUT_SINGLE | GLUT_RGBA);
  glutInitWindowPosition(100,100);
  glutInitWindowSize(500,500);
  glutCreateWindow("icosaedru");
  glutDisplayFunc(render);
  glutMainLoop();
}
```

În unele secven”Hde exemplu cod regŁsim vârfurile unui icosaedru regulat (care este un compus din douŁzeci de fe”eHcare , fiecare fa”Hde care este un triunghi echilateral). Un icosaedru poate fi consideratŁo aproximare durŁa unei sferŁ. Exemplul define”Hde nodurile ”Hitriunghiurile care formeazŁun icosaedru .

```
#define X .525731112119133606
#define Z .850650808352039932

static GLfloat vdata[12][3] =
{
    {-X, 0.0, Z}, {X, 0.0, Z}, {-X, 0.0, -Z}, {X, 0.0, -Z},
    {0.0, Z, X}, {0.0, Z, -X}, {0.0, -Z, X}, {0.0, -Z, -X},
    {Z, X, 0.0}, {-Z, X, 0.0}, {Z, -X, 0.0}, {-Z, -X, 0.0}
};

static GLuint tindices[20][3] =
{
    {0,4,1}, {0,9,4}, {9,5,4}, {4,5,8}, {4,8,1},
    {8,10,1}, {8,3,10}, {5,3,8}, {5,2,3}, {2,7,3},
    {7,10,3}, {7,6,10}, {7,11,6}, {11,0,6}, {0,1,6},
    {6,1,10}, {9,0,11}, {9,11,2}, {9,2,5}, {7,2,11}
};

int i;

glBegin(GL_TRIANGLES);
```

```
    for (i = 0; i < 20; i++)
    {
        /* color information here */
        glVertex3fv(&vdata[tindices[i][0]][0]);
        glVertex3fv(&vdata[tindices[i][1]][0]);
        glVertex3fv(&vdata[tindices[i][2]][0]);
    }
glEnd();
```

Urmatorul program , modelează un cub zburator. Programul are efecte de animaţie prin deplasarea camerei pe orbită în jurul unui obiect staţionar.

În cadrul programului se indică faptul că facem un zbor pe deasupra, setând forma camerei (gluPerspective în mod GL_PROJECTION), dar setăm şi poziţia de orientare a camerei (gluLookAt în mod GL_MODELVIEW) pentru fiecare cadru.
Animaţia decurge la un timp de de 16.6667 ms .

```
// Acest program este o privire in jurul unui cub colorat RGB .
// Cubul este constituit ca un poliedru convex

#ifdef __APPLE_CC__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include <cmath>

// Cubul are colturi opuse (0,0,0) and (1,1,1), care sunt negre
// respectiv albe .   Axa x- este configurata i rosu,axa y in
// verde, si axa z este configurata in albastru. Pozitia cubului
// si culorile sunt fixe.
namespace Cube {

const int NUM_VERTICES = 8;
const int NUM_FACES = 6;

GLint vertices[NUM_VERTICES][3] = {
  {0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {0, 1, 1},
  {1, 0, 0}, {1, 0, 1}, {1, 1, 0}, {1, 1, 1}};

GLint faces[NUM_FACES][4] = {
  {1, 5, 7, 3}, {5, 4, 6, 7}, {4, 0, 2, 6},
  {3, 7, 6, 2}, {0, 1, 3, 2}, {0, 4, 5, 1}};

GLfloat vertexColors[NUM_VERTICES][3] = {
  {0.0, 0.0, 0.0}, {0.0, 0.0, 1.0}, {0.0, 1.0, 0.0}, {0.0, 1.0, 1.0},
  {1.0, 0.0, 0.0}, {1.0, 0.0, 1.0}, {1.0, 1.0, 0.0}, {1.0, 1.0, 1.0}};

void draw() {
  glBegin(GL_QUADS);
  for (int i = 0; i < NUM_FACES; i++) {
    for (int j = 0; j < 4; j++) {
      glColor3fv((GLfloat*)&vertexColors[faces[i][j]]);
      glVertex3iv((GLint*)&vertices[faces[i][j]]);
```

```
    }
  }
  glEnd();
}
}

// Afisare si animatie. Pentru a desena am sters fereastra.
// Deoarece fereastra principala are doua buffere am sters una pentru a face
// desenul vizibil. Animatia se realizeaza prin miscarea succesiva a
// camerei si desenului.
void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  Cube::draw();
  glFlush();
  glutSwapBuffers();
}

// Vom zbura in jurul cubului prin miscarea camerei in lungul orbitei dupa curba
//  u->(8*cos(u), 7*cos(u)-1, 4*cos(u/3)+2).  Pastram dispozitivul camerei
//  in centrul cubului (0.5, 0.5, 0.5) si modificam vectorul pana la atingerea
// unui efect de rostogolire.
void timer(int v) {
  static GLfloat u = 0.0;
  u += 0.01;
  glLoadIdentity();
  gluLookAt(8*cos(u), 7*cos(u)-1, 4*cos(u/3)+2, .5, .5, .5, cos(u), 1, 0);
  glutPostRedisplay();
  glutTimerFunc(1000/60.0, timer, v);
}

// Cand fereastra este remodelata trebuie sa resetam valorile camerei
//corespunzatoare noii ferestre. Setam fereastra de afisare poarta la (0,0)-(w,h).  Setam
// camera pentru a avea 60 de grade pe vertical ,campul vederii, situat la w/h, apropiere
// la distanta 0.5 si o departare de 40.
void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0, GLfloat(w) / GLfloat(h), 0.5, 40.0);
  glMatrixMode(GL_MODELVIEW);
}

// Initializarea specifica aplicatiei: Urmatorul lucru
// este sa activam transparenta
// deoarece este vorba de un poliedru convex.
void init() {
  glEnable(GL_CULL_FACE);
  glCullFace(GL_BACK);
}

// Meniul principal pentru aplicatia GLUT.
int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutCreateWindow("The RGB Color Cube");
  glutReshapeFunc(reshape);
  glutTimerFunc(100, timer, 0);
```

```
  glutDisplayFunc(display);
  init();
  glutMainLoop();
}
```

Configurarea mediului vizual

DacŁ veŞi crea programe în Win32, de OpenGL, veŞi avea nevoie sŁ cunoaкteŞi o serie de funcŞii de configurare a mediului vizual. Mai întâi se vor expune o serie de secvenŞe de cod în care sunt implicate astfel de funcŞii:

```
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0f,(GLfloat)w/(GLfloat)h, 1.0, 400.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

În aceastŁ secvenŞŁ avem urmŁtoarele funcŞii:
glViewport stabileкte parametrii ferestrei software de desenare(Viewport) specificaŞi prin poziŞia colŞului stânga - sus , кi lungimea w, кi înŁlŞimea h a ferestrei.
glMatrixMode(GL_PROJECTION) stabileкte cŁ se va lucra cu matricea de proiecŞie;
glLoadIdentity() stabileкte cŁ matricea de proiecŞie va fi resetatŁ la starea iniŞialŁ;
gluPerspective face modificŁri asupra matricei de proiecŞie ï stabilind unghiul percepŞiei ï 60.0 de grade, aspectul proiecŞiei adicŁ raportul dintre înŁlŞimea кi lungimea imaginii, cel mai apropiat кi cel mai îndepŁrtat punct pe oz pe baza cŁrora se stabileкte planurile de decupare a scenei, astfel cŁ ce este înafara acestor limite nu este afiкat ;
glMatrixMode(GL_MODELVIEW) ï se stabileкte cŁ se va lucra cu matricea modelelor 3D ;
glLoadIdentity() ï se seteazŁ mediul 3D la starea iniŞialŁ;

De obicei o astfel de secvenŞŁ este apelatŁ la fiecare redimensionare a ferestrei,sau înainte de prima afiкare a ferestrei aplicaŞiei.

O altŁ funcŞie de configurare ar fi urmŁtoarea:
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdoublecenterx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy,GLdouble upz); unde
eyex, eyey, eyez ï este poziŞia observatorului sau mai bine spus a ochiului acestuia;
centerx, centerzy, center z este centrul scenei spre care priveкte.
upx, upy, upz ï specficŁ direcŞia privirii observatorului.

Exemplu:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(locX, locY, locZ, dirX, dirY, dirZ, 0.0f, 1.0f, 0.0f);
```

În acest exemplu matricea de vizualizare este resetatŁ, ќi apoi setatŁ în raport cu ochiul bservatorului.
Implicit observatorul e situat in origine, priveste in directia negativa a axei oz, iar directia sus a planului de vizualizare este directia pozitiva a axei oy (0,1,0).
In acest moment matricea de modelare-vizualizare este matricea identitate.
Cand apelati gluLookAt transformarea descrisa de aceasta este compusa cu cea deja existenta. De aceea daca o apelati de doua ori, al doilea apel s-ar putea sa produca alte rezultate decat cele pe care le asteptati.
Folosind aceastŁ funcŞe putem crea un program prin care sŁ explorŁm o anumitŁ scenŁ.

Inainte de gluLookAt puteti apela secventa :
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
pentru a incarca matricea identitate in matricea de modelare-vizualizare.
Obiectele le puteti aseza/transforma in scena folosind:
ÅglScalef(GLfloat x, GLfloat y, GLfloat z)
ÅglTranslatef(GLfloat x, GLfloat y, GLfloat z)
ÅglRotatef (GLfloat unghi, GLfloat x, GLfloat y, GLfloat z)
Singura a carei semnificatie nu e evidenta este glRotatef. Aceasta functie
roteste in sens trigonometric in jurul axei x,y,z cu unghiul unghi.

UrmŁtorul program openGL deseneazŁ un cub cŁruia i s-a aplicat o transformare de modelare (scalare). Transformarea de vizualizare, redatŁ în exemplu prin funcĭia **gluLookAt ()**, poziĭioneazŁ camera (observatorul). Sunt de asemenea specificate ќi o transformare de proiecĭie ĭi o transformarea viewport.

O astfel de functie modifica matricea curenta, astfel incat transformarea se va aplica tuturor obiectelor pe care le vom desena de acum incolo, lasandu-le neschimbate pe cele deja desenate. Insa comportamentul lor este diferit de ceea ce ne-am astepta. Transformarile sunt aplicate obiectului in ordinea inversa apelurilor.
Pana acum am vazut cum stabilim pozitia si orientarea observatorului in raport cu obiectele din lume. Insa pentru a face scena sa apara pe ecran trebuie sa delimitam spatiul vizibil: volumul de vizualizare. OpenGL permite mai multe moduri de a face acest lucru.
Proiectia ortografica se realizeaza cu ajutorul functiei:
ÅglOrtho(GLdouble left, GLdouble right, GLdouble bottom,
GLdouble top, GLdouble near, GLdouble far)
Volumul de vizualizare este un paralelipiped dreptunghic. Nu uitati insa ca pozitia observatorul este implicit in origine si priveste in directia negativa a axei oz. Deci obiectele cu z intre -near si -far vor fi vizibile.
Proiectia perspectiva se poate realiza in doua moduri:
ÅglFrustum(GLdouble left, GLdouble right, GLdouble bottom,
GLdouble top, GLdouble near, GLdouble far)
left, right, bottom, top se refera in acest caz la fereastra din planul apropiat.
Cum stim pozitia observatorului (e cea implicita sau cea setata de noi cu gluLookAt) putem calcula volumul de vizualizare. Distantele near si far trebuie sa fie pozitive.

Un alt mod de a specifica acest volum de vizualizare este cu ajutorul functiei: void gluPerspective(GLdouble fovy, GLdouble aspect, GLdoublenear, GLdouble far)
Ati recunoscut deja parametrii near si far. Parametrul aspect reprezinta raportul dintre lungimea si inaltimea ferestrei in planul de aproape. Pentru a obtine o imagine nedistorsionata acest raport trebuie sa corespunda cu cel al ferestrei de afisare. Parametrul fovy este unghiul de vizualizare in planul xOz si trebuie sa fie in intervalul [0.0, 180.0]
Transformarea in poarta de vizualizare se defineste folosind functia:
ÅglViewport (GLint px, GLint py, GLint pz, GLsizei width, GLsizeiheight) px, py reprezinta coordonatele in fereastra ale coltului stanga jos al portii.
Implicit sunt 0,0 ;width, height sunt latimea si inaltimea portii. Valorile implicite sunt date de latimea si inaltimea ferestrei in care se afiseaza.

```
// cub_transformat
//

#include <GL/glut.h>

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

void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glLoadIdentity (); /* sterge matricea de identitate */
   /* transformarea de vizualizare */
   gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
   glScalef (1.0, 2.0, 1.0); /* transformarea de modelare */
   glutWireCube (1.0);
   glFlush ();
}

void reshape (int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
   glMatrixMode (GL_MODELVIEW);
}

void main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMainLoop();
}
```

Exercițiu: Se vor schimba valorile `gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);` pentru a experimenta diferite transformări de vizualizare aplicate cubului.

TRANSFORMARI GEOMETRICE ÎN SPATIU

Obiectele unei scenei pot fi modificate sau manevrate în spatiul tridimensional folosind diferite transformari geometrice. Dintre acestea, cele mai importante sunt: translatia, care modifica

localizarea obiectului; rotatia, care modifica orientarea; scalarea, care modifica dimensiunea obiectului. Aceste transformari sunt denumite transformari geometrice primitive.

Transformarea in 2D

```
// o rotatie simpla 2D

#include <GL/glut.h>

GLfloat yang = 1.2;

void display()
{
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
   glRotatef(yang,0.0,1.0,0.0);
   glBegin(GL_POLYGON);
      glColor3f(1.0,0.0,0.0);
      glVertex2f(0.0,  0.45);
      glColor3f(0.0,1.0,0.0);
      glVertex2f(-0.45, -0.45);
      glColor3f(0.0,0.0,1.0);
      glVertex2f(0.45, -0.45);
   glEnd();
   glutSwapBuffers();
   glFlush();
}

void init()
{
   glClearColor(1.0,1.0,1.0,1.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
}


void main(int argc, char** argv)
{
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE |
GLUT_RGBA);
   glutInitWindowSize(400,400);
   glutInitWindowPosition(0,0);
   glutCreateWindow("Rotatie 2D ");
   glutDisplayFunc(display);
   glutIdleFunc(display);
   init();
   glutMainLoop();
}
```

În exemplul prezentat mai sus am aplicat o transformare de rotație simplă 2D. În codul nostru sursă s-a desenat un triunghi (amestec roșu-verde-albastru) și s-a adăugat codul necesar pentru a se roti.

În primul rând am comunicat GLUT că dorim un tampon **dublu** **glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA).** De asemenea, s-a şters tamponul de profunzime (avem mai mult de unul), care se realizează prin includerea GL_DEPTH_BUFFER_BIT la lista parametrilor funcției **glClear ().** In cazul in care folosim un buffer dublu, la sfarsitul lui *f trebuie sa cerem schimbarea bufferelor prin glutSwapBuffers( ) .

Exercițiu1: Translatați și rotiți triunghiul. În scopul de a menține punctul nostru de vedere va trebui să aduceți funcția glLoadIdentity () înainte de fiecare traducere. Apelarea glLoadIdentity () resetează matricea de transformarea astfel rotație noastră este, de asemenea resetat. Pentru a face triunghiul să se rotească trebuie să măriți continuu mărimea unghiului de rotație până la 360.0 grade. Putem realiza acest lucru prin modificarea funcția de afișare () în modul următor.

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
   glTranslatef(0.4,-0.25,0.0);
   glRotatef(yang,0.0,1.0,0.0);
   glBegin(GL_POLYGON);
      glColor3f(1.0,0.0,0.0);
      glVertex2f(-0.0,  0.45);
      glColor3f(0.0,1.0,0.0);
      glVertex2f(-0.45, -0.5);
      glColor3f(0.0,0.0,1.0);
      glVertex2f( 0.45, -0.45);
   glEnd();
   yang = yang + 1.2;
   if (yang>360.0)
    yang = 0.0;
   glutSwapBuffers();
   glFlush();
}
```

Exercițiu2:SŁ presupunem cŁ dorim sŁ genereze o animaŒHe în care douŁ triunghiuri sunt rotite independent în jurul a douŁ axe diferite. ModificaŒHfuncŒHa de afiŒHare () pentru a desena un alt triunghi reprezentat în partea din stânga sus a ferestrei de vizualizare ŒHse roteŒHe în jurul axei x.

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
   glTranslatef(0.4,-0.25,0.0);
   glRotatef(yang,0.0,1.0,0.0);
   glBegin(GL_POLYGON);
      glColor3f(1.0,0.0,0.0);
      glVertex2f(-0.0,  0.45);
```

```
      glColor3f(0.0,1.0,0.0);
      glVertex2f(-0.45, -0.5);
      glColor3f(0.0,0.0,1.0);
      glVertex2f( 0.45, -0.45);
   glEnd();
   glLoadIdentity();
   glTranslatef(-0.4,0.25,0.0);
   glRotatef(xang,1.0,0.0,0.0);
   glBegin(GL_POLYGON);
      glColor3f(1.0,1.0,0.0);
      glVertex2f(-0.0,  0.45);
      glColor3f(0.0,1.0,1.0);
      glVertex2f(-0.45, -0.5);
      glColor3f(1.0,0.0,1.0);
      glVertex2f( 0.45, -0.45);
   glEnd();
   yang = yang + 1.2;
   if (yang>360.0)
    yang = 0.0;
   xang = xang - 0.5;
   if (xang<0.0)
    xang = 360.0;
   glutSwapBuffers();
   glFlush();
}
```

Exerciţiu3: Pentru a înţelege şi aprecia importanţa celui de al doilea glLoadIdentity () (afişate cu caractere aldine mai sus) executaţi programul cu şi fără acest apel de funcţie.

Transformari ale obiectelor 3D in OpenGL

Pentru a înţelege transformarea 3D a obiectelor , vom înlocui triunghiul din dreapta-jos în programul nostru de transformare 2D de mai sus, cu o piramidă. Vom inlocui programul cu următoarea secvenţă de cod:

```
glBegin(GL_TRIANGLES);
     glColor3f(1.0,1.0,0.0);
     glVertex3f( 0.0, 0.5, 0.0);
     glVertex3f(-0.5,-0.5, 0.5);
     glVertex3f( 0.5,-0.5, 0.5);
     glColor3f(0.0,1.0,0.0);
     glVertex3f( 0.0, 0.5, 0.0);
     glVertex3f( 0.5,-0.5, 0.5);
     glVertex3f( 0.5,-0.5,-0.5);
     glColor3f(0.0,0.0,1.0);
     glVertex3f( 0.0, 0.5, 0.0);
     glVertex3f( 0.5,-0.5,-0.5);
     glVertex3f(-0.5,-0.5,-0.5);
     glColor3f(1.0,0.0,0.0);
     glVertex3f( 0.0, 0.5, 0.0);
```

```
    glVertex3f(-0.5,-0.5,-0.5);
    glVertex3f(-0.5,-0.5, 0.5);
  glEnd();
```

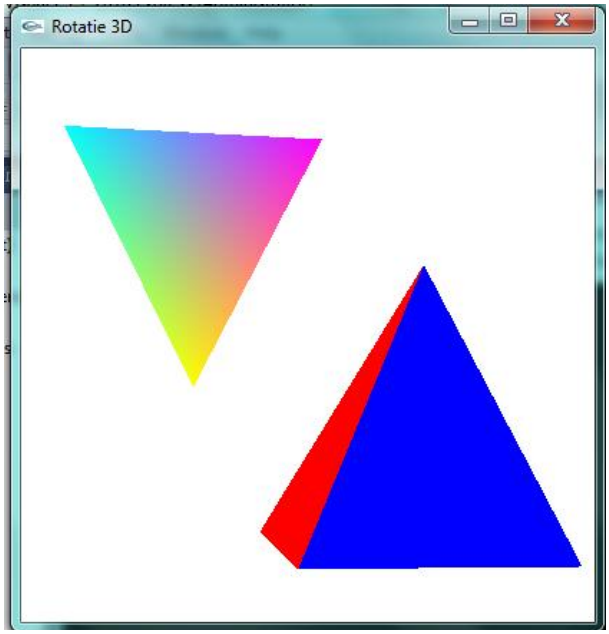
Cele patru triunghiuri au un nod comun (0.0,0.5,0.0) ’Hi fiecare triunghi are un nod de bazŁ cu fiecare dintre vecinii sŁi. Am plecat de la fa”aHinferioarŁ (un pŁtrat) deschis. Vom roti aceastŁ piramidŁ despre atât dupŁ axa x ,cât ’Hi dupŁ axa y, folosind (func”iaH glRotatef),

```
glRotatef(yang,1.0,1.0,0.0);
```

Rularea acest program fără alte modificări creează un efect ciudat.

```
void init()
 {
    glClearColor(1.0,1.0,1.0,1.0);
 glClearDepth(1.0);
 glEnable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
 }
```

Listarea completŁ a programului modificat este prezentatŁ mai jos,




```
//  rotatie simpla 3D demo

 #include <GL/glut.h>

 GLfloat yang = 0.0;
 GLfloat xang = 0.0;
```

```
void display()
 {
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.4,-0.25,0.0);
    glRotatef(yang,1.0,1.0,0.0);
    glBegin(GL_TRIANGLES);
     glColor3f(1.0,1.0,0.0);
     glVertex3f( 0.0, 0.5, 0.0);
     glVertex3f(-0.5,-0.5, 0.5);
     glVertex3f( 0.5,-0.5, 0.5);
     glColor3f(0.0,1.0,0.0);
     glVertex3f( 0.0, 0.5, 0.0);
     glVertex3f( 0.5,-0.5, 0.5);
     glVertex3f( 0.5,-0.5,-0.5);
     glColor3f(0.0,0.0,1.0);
     glVertex3f( 0.0, 0.5, 0.0);
     glVertex3f( 0.5,-0.5,-0.5);
     glVertex3f(-0.5,-0.5,-0.5);
     glColor3f(1.0,0.0,0.0);
     glVertex3f( 0.0, 0.5, 0.0);
     glVertex3f(-0.5,-0.5,-0.5);
     glVertex3f(-0.5,-0.5, 0.5);
    glEnd();
    glLoadIdentity();
    glTranslatef(-0.4,0.25,0.0);
    glRotatef(xang,1.0,0.0,0.0);
    glBegin(GL_POLYGON);
       glColor3f(1.0,1.0,0.0);
       glVertex2f(-0.0,  0.45);
       glColor3f(0.0,1.0,1.0);
       glVertex2f(-0.45, -0.5);
       glColor3f(1.0,0.0,1.0);
       glVertex2f( 0.45, -0.45);
    glEnd();
    yang = yang + 1.2;
    if (yang>360.0)
     yang = 0.0;
    xang = xang - 0.5;
    if (xang<0.0)
     xang = 360.0;
    glutSwapBuffers();
    glFlush();
 }

 void init()
 {
    glClearColor(1.0,1.0,1.0,1.0);
    glClearDepth(1.0);
    glEnable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
 }

 void main(int argc, char** argv)
 {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
```

```
    glutInitWindowSize(400,400);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Rotatie 3D");
    glutDisplayFunc(display);
    glutIdleFunc(display);
    init();
    glutMainLoop();
}
```