1. BIBLIOTECI GRAFICE

Pentru programarea aplicatiilor grafice complexe se pot utiliza mai multe biblioteci si interfete grafice, precum si sisteme de dezvoltare de programe (toolkit-uri), care permit proiectantului sa reutilizeze un numar mare de functii grafice deja implementate si sa-si concentreze eforturile asupra aplicatiei însasi. Dat fiind ca majoritatea acestor biblioteci sunt foarte iefine sau disponibile gratis pe Internet si pot fi folosite într-un numar mare de platforme hardware si software, cunoasterea si utilizarea lor este deosebit de importanta si utila.

Dintre bibliotecile grafice existente, biblioteca OpenGL, scrisa în limbajul C, este una dintre cele mai utilizate, datorita faptului ca implementeaza un mare numar de functii grafice de baza pentru crearea aplictiilor grafice tridimensionale, asigurând o interfata independenta de platforma hardware.

În redarea obiectelor tridimensionale, biblioteca OpenGL foloseste un numar redus de primitive geometrice (puncte, linii, poligoane), iar modele complexe ale obiectelor si scenelor tridimensionale se pot dezvolta particularizat pentru fiecare aplicatie, pe baza acestor primitive. Dat fiind ca OpenGL prevede un set puternic, dar de nivel scazut, de comenzi de redare a obiectelor, mai sunt folosite si alte biblioteci de nivel mai înalt care utilizeaza aceste comenzi si preiau o parte din sarcinile de programare grafica. Astfel de biblioteci sunt:

- Biblioteca de functii utilitare GLU (OpenGL Utility Library) permite definirea sistemelor de vizualizare, redarea suprafetelor curbe si alte functii grafice.
- Pentru fiecare sistem Windows exista o extensie a bibliotecii OpenGL care asigura interfata cu sistemul respectiv: pentru Microsoft Windows, extensia WGL, pentru sisteme care folosesc X Window, extensia GLX, pentru sisteme IMB OS/2, extensia PGL.
- Biblioteca de dezvoltare GLUT (OpenGL Utility Toolkit) este un sistem de dezvoltare independent de platforma, care ascunde dificultatile interfetelor de aplicatii Windows, punând la dispozitie functii pentru crearea si initializarea ferestrelor si pentru executia programelor grafice bazate pe biblioteca OpenGL.

Toate functiile bibliotecii OpenGL încep cu prefixul gl, functiile GLU încep cu prefixul glu, iar functiile GLUT încep cu prefixul glut.

Înainte de a prezenta caracteristicile bibliotecii OpenGL, se vor preciza conventiile de reprezentare a coordonatelor punctelor în spatiul tridimensional si al culorilor.

Sisteme de referinta tridimensionale. Pentru crearea si redarea scenelor tridimensionale este necesar ca obiectele sa fie pozitionate într-un sistem de referinta tridimensional. Exista mai multe posibilitati de a specifica pozitia unei multimi de puncte (vârfuri) prin care este reprezentat un obiect în spatiul tridimensional: coordonate cilindrice, coordonate sferice, coordonate carteziene. Dintre aceste sisteme de referinta, cel mai utilizat în aplicatiile grafice este sistemul de coordonate cartezian.

Sistemul de coordonate cartezian în care sunt definite toate obiectele scenei virtuale se numeste sistem de referinta universal (*world coordinate system*- WCS).

Un sistem de coordonate cartezian se defineste prin originea O si trei axe perpendiculare, Ox, Oy si Oz, orientate dupa regula mâinii drepte sau dupa regula mâinii stângi. Într-un sistem orientat dupa regula mâinii drepte, daca se roteste mâna dreapta în jurul axei z de la axa x pozitiva spre axa y pozitiva, orientarea degetului mare este în directia z pozitiv. Într-un sistem orientat dupa regula mâinii stângi, rotirea de la axa x pozitiva spre axa y pozitiva, cu orientarea degetului mare în directia z pozitiv, se obtine folosind mâna stânga. Diferite sisteme de grafica tridimensionala folosesc conventii diferite pentru definirea sistemelor de referinta, ceea ce conduce la confuzii, daca nu se precizeaza conventia folosita. În acest text, pentru sistemul de referinta universal se foloseste conventia de sistem de coordonate drept. În grafica tridimensionala se mai folosesc si alte sisteme de referinta, care permit descrierea operatiilor de transformari geometrice si care vor fi precizate pe parcurs.

Un punct P în spatiul tridimensional se reprezinta în sistemul de referinta Cartezian printr-un tripet de valori scalare x, y, z, care reprezinta componentele vectorului de pozitie **OP** pe cele trei axe de coordonate. Daca se noteza cu **i**, **j**, **k** versorii (vectorii unitate) ai celor trei axe de coordonate x, y,

z, atunci vectorul de pozitie al punctului P este **OP** = x**i** + y**j** + z**k**. În notatia matriceala, un punct în spatiul tridimensional se poate reprezenta printr-o matrice linie sau coloana:

$$\mathbf{P} = \begin{bmatrix} x & y & z \end{bmatrix} \text{ sau } \mathbf{P} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Ambele conventii sunt folosite în egala masura în sistemele grafice, ceea ce, din nou, poate provoca diferite confuzii, daca nu se precizeaza conventia folosita. Conventia de reprezentare sub forma de matrice linie a unui punct are avantajul ca exprima operatiile de concatenare a matricelor într-un mod natural, de la stânga la dreapta. Conventia de reprezentare matematica, standardul grafic PHIGS, biblioteca grafica OpenGL, ca si unele din lucrarile de referinta în domeniu, folosesc notatia de matrice coloana pentru un punct în spatiul tridimensional, care este adoptata si în lucrarea prezenta.

Reprezentarea culorilor în sistemele grafice. Imaginea care se obtine pe display este compusa dintr-un anumit numar de pixeli, dat de rezolutia display-ului. Fiecare pixel are o pozitie pe ecran, data de adresa lui în ferestra de afisare, si o culoare care poate fi reprezentata în mai multe modele: modelul RGB, modelul HSV, modelul HLS, si altele. Dintre aceste modele, în grafica se foloseste cel mai frecvent modelul RGB.

În modelul RGB, culoarea este reprezentata printr-un triplet de culori primare, rosu (*red*) verde (*green*), albastru (*blue*). Utilizarea preponderenta a modelului RGB în grafica se datoreaza în primul rând faptului ca monitoarele color folosesc acest model de producere a culorii. Orice culoare în modelul RGB se exprima printr-un triplet (r,g,b) si îi corespunde un punct în spatiul RGB al carui vector **C** este:

C = r**R** + g**G** + b**G**

unde **R**, **G**, **B** sunt versorii axelor rosu (*red*), verde (*green*), albastru (*blue*). În acest model culoarea negru este reprezentata prin tripletul (0,0,0), iar culoarea alb este reprezentata prin tripletul (1,1,1).

În sistemele grafice se mai foloseste o varianta a modelului RGB, modelul RGBA, unde cea de-a patra componenta (α) indica transparenta suprafetei. Valoarea 1 a acestei componente ($\alpha = 1$) înseamna suprafata opaca, iar valoarea minima ($\alpha = 0$) înseamna suprafata complet transparenta. Daca transparenta unei suprafete este diferita de zero, atunci culoarea care se atribuie pixelilor acestei suprafete se modifica în functie de culoarea existenta în bufferul de imagine.

1.1 CARACTERISTICILE BIBLIOTECII OPENGL

Biblioteca OpenGL defineste propriile tipuri de date, cele mai multe corespunzând tipurilor de date fundamentale ale limbajului C. De exemplu, în gl.h sunt definite urmatoarele tipuri:

```
typedef unsigned int GLenum;
typedef unsigned char GLboolean;
typedef signed char GLbyte;
typedef int GLint;
typedef int GLsizei;
typedef unsigned int GLuint;
typedef float GLfloat;
typedef float GLclampf;
typedef double GLdouble;
```

De asemenea, în fisierul header ale bibliotecii OpenGL (gl.h) sunt definite mai multe constante simbolice, care reprezinta diferite stari, variabile sau valori de selectie a optiunilor OpenGL. Aceste constante sunt toate scrise cu majuscule si sunt precedate de prefixul `GL_`. De exemplu, constantele simbolice care definesc valorile `TRUE` si `FALSE` si cele care selecteaza tipul unei primitive grafice sunt:

```
/* Boolean */
#define GL_TRUE                         1
#define GL_FALSE                        0
```

```
/* BeginMode */
#define GL_POINTS                       0x0000
#define GL_LINES                        0x0001
#define GL_TRIANGLES                    0x0004
#define GL_POLYGON                      0x0009
```

Pentru a întelege functionarea comenzilor OpenGL, se descriu în continuare cele mai importante dintre caracteristicile OpenGL.

Poarta de afisare OpenGL mai este numita *context de redare* (*rendering context*) si este asociata unei ferestre de afisare din sistemul Windows. Daca se programeaza folosind biblioteca GLUT, corelarea dintre fereastra de afisare si poarta OpenGL este asigurata de functii ale acestei biblioteci. Daca nu se foloseste biblioteca GLUT, atunci functiile bibliotecilor de extensie XGL, WGL sau PGL permit asocierea contextului de redare OpenGL cu o fereastra de afisare si accesul la aceasta.

În OpenGL un pixel este reprezentat printr-un descriptor care defineste mai multi parametri:

- numarul de biti/pixel pentru memorarea culorii
- numarul de biti/pixel pentru memorarea adâncimii
- numarul de buffere de imagine.

Bufferul de cadru (*frame buffer*) contine toate datele care definesc o imagine si consta din mai multe sectiuni logice: bufferul de imagine (sau bufferul de culoare), bufferul de adâncime (*Z-buffer*), bufferul sablon (*stencil*), bufferul de acumulare (*accumulation*).

Bufferul de imagine (*image buffer*, *color buffer*) în OpenGL poate contine una sau mai multe sectiuni, în fiecare fiind memorata culoarea pixelilor din poarta de afisare. Redarea imaginilor folosind un singur buffer de imagine este folosita pentru imagini statice, cel mai frecvent în proiectarea grafica (CAD). În generarea interactiva a imaginilor dinamice, un singur buffer de imagine produce efecte nedorite, care diminueaza mult calitatea imagii generate.

Orice cadru de imagine începe cu stergerea (de fapt, umplerea cu o culoare de fond) a bufferului de imagine. Dupa aceasta sunt generati pe rând pixelii care apartin tuturor elementelor imaginii (linii, puncte, suprafete) si intensitatile de culoare ale acestora sunt înscrise în bufferul de imagine. Pentru trecerea la cadrul urmator, trebuie din nou sters bufferul de imagine si reluata generarea elementelor componente, pentru noua imagine. Chiar daca ar fi posibila generarea si înscrierea în buffer a elementelor imaginii cu o viteza foarte mare (ceea ce este greu de realizat), tot ar exista un interval de timp în care bufferul este sters si acest lucru este perceput ca o pâlpâire a imaginii. În grafica interactiva timpul necesar pentru înscrierea datelor în buffer este (în cazul cel mai fericit) foarte apropiat de intervalul de schimbare a unui cadru a imaginii (update rate) si, daca acest proces are loc simultan cu extragerea datelor din buffer si afisarea lor pe display, atunci ecranul va prezenta un timp foarte scurt imaginea completa a fiecarui cadru, iar cea mai mare parte din timp ecranul va fi sters sau va contine imagini partiale ale cadrului. Tehnica universal folosita pentru redarea imaginilor dinamice (care se schimba de la un cadru la altul) este tehnica *dublului buffer de imagine*.

În aceasta tehnica exista doua buffere de imagine: bufferul din fata (*front*), din care este afisata imaginea pe ecran si bufferul din spate (*back*), în care se înscriu elementele de imagine generate. Când imaginea unui cadru a fost complet generata, ea poate fi afisata pe ecran printr-o simpla operatie de comutare între buffere: bufferul spate devine buffer fata, si din el urmeaza sa fie afisata imagine cadrului curent, iar bufferul fata devine buffer spate si în el urmeaza sa fie generata imaginea noului cadru. În OpenGL comutarea bufferelor este efectuata de functia `SwapBuffers()`. Biblioteca OpenGL ofera posibilitatea creerii imaginilor cu simplu sau dublu buffer, monoscopice si stereoscopice.

Bufferul de adâncime (*depth buffer*) memoreaza adâncimea fiecarui pixel si, prin aceasta, permite eliminarea suprafetelor ascunse. Bufferul de adâncime contine acelasi numar de locatii ca si un buffer de imagine, fiecare locatie corespunzând unui pixel, de la o anumita adresa. Valoarea memorata în locatia corespunzatoare unui pixel este distanta acestuia fata de punctul de observare (adâncimea pixelului). La generarea unui nou pixel cu aceeasi adresa, se compara adâncimea noului pixel cu adâncimea memorata în bufferul de adâncime, si noul pixel înlocuieste vechiul pixel (îl

“ascunde”) daca este mai apropiat de punctul de observare. Bufferul de adâncime se mai numeste si Z-buffer, de la coordonata z, care reprezinta adâncimea în sistemul de referinta ecran 3D.

Operatiile de baza. OpenGL deseneaza *primitive geometrice* (puncte, linii si poligoane) în diferite moduri selectabile. O primitiva este definita printr-unul sau mai multe vârfuri (*vertices*). Un vârf defineste un punct, capatul unei linii sau vârful unui poligon. Fiecare vârf are asociat un set de date: coordonate, culoare, normala, coordonate de textura.

Aceste date sunt prelucrate în ordine si în acelasi mod pentru toate primitivele geometrice. Modul în care este executata secventa de operatii pentru redarea primitivelor geometrice depinde de starea bibliotecii OpenGL, stare care este definita prin mai multe variabile de stare ale acesteia (parametri). Numarul de variabile de stare ale bibliotecii este destul de mare, descrierea lor poate fi gasita în manualul de referinta (*OpenGL Reference Manual*), iar pe parcursul expunerii vor fi prezentate numai cele mai importante dintre acestea.

La initializare, fiecare variabila de stare este setata la o valoare implicita. O stare o data setata îsi mentine valoarea neschimbata pâna la o noua setare. Variabilele de stare au denumiri date sub forma de constante simbolice care pot fi folosite pentru aflarea valorilor acestora. Câteva exemple de stari definite prin constante simbolice în fisierul gl.h sunt:

```
#define GL_CURRENT_COLOR                 0x0B00
#define GL_CURRENT_NORMAL                0x0B02
#define GL_MODELVIEW_MATRIX              0x0BA6
#define GL_PROJECTION_MATRIX             0x0BA7
```

Variabilele de stare OpenGL sunt de doua categorii: variabile de tip binar si variabile definite prin diferite structuri de date.

Variabile de tip binar pot avea una din doua stari: starea activa (*enabled*) sau starea inactiva (*disabled*). Setarea la starea activa se realizeaza prin apelul functiei

```
void glEnable(GLenum param);
```

unde `param` este numele simbolic al parametrului (variabilei de stare).

Setarea la starea inactiva se realizeaza prin apelul functiei:

```
void glDisable(GLenum param);
```

De exemplu, apelul functiei `glEnable(GL_DEPTH_TEST)` activeaza testul de adâncime si actualizarea corespunzatoare a bufferului de adâncime (*depth buffer*), iar apelul functiei `glDisable(GL_DEPTH_TEST)` dezactiveaza testul de adâncime.

În orice loc într-un program OpenGL, se poate afla valoarea unui parametru binar prin apelul functiei: `GLboolean glIsEnabled(GLenum param)` care returneaza `GL_TRUE` daca parametrul `param` este în starea activa si `GL_FALSE` daca parametrul `param` este în starea inactiva.

Valoarea unei variabile de stare care nu este de tip binar se seteaza prin apelul unei functii specifice variabilei respective, care are ca argumente valorile necesare pentru actualizare. De exemplu, variabila de stare culoare curenta, denumita `GL_CURRENT_COLOR`, se seteaza prin functia:

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

la o valoare data prin trei componente: rosu (*red*), verde (*green*), albastru (*blue*).

Fiind o biblioteca dezvoltata în limbajul C, fara posibilitatea de supraîncarcare a functiilor, selectia unei functii apelate cu diferite tipuri de argumente de apel este realizata prin modificarea (printr-un sufix) a numelui functiei. De exemplu, functia de setare a culorii curente are mai multe variante, dupa tipul si numarul argumentelor: `glColor3f()`, `glColor4d()`, etc. În continuare, în aceasta lucrare se noteaza generic cu # sufixul dintr-o familie de functii (de exemplu, `glColor#()`).

Primitive geometrice. Functiile OpenGL executa secventa de operatii grafice asupra fiecarei primitive geometrice, definita prin tipul acesteia si o lista de vârfuri. Coordonatele unui vârf al unei primitive sunt transmise catre OpenGL prin apelul unei functii `glVertex#()`. Aceasta are mai multe variante, dupa numarul si tipul argumentelor. Iata, de exemplu, numai câteva din prototipurile functiilor `glVertex#()`:

```
void glVertex2d(GLdouble x, GLdouble y);
```

```
void glVertex3d(GLdouble x, GLdouble y, GLdouble z);
void glVertex3f(GLfloat x, GLfloat y, GLfloat z);
void glVertex4d(GLdouble x, GLdouble y,GLdouble z, GLdouble w);
```

Vârfurile pot fi specificate în plan, în spatiu sau în coordonate omogene, folosind apelul functiei corespunzatoare.

O primitiva geometrica se defineste printr-o lista de vârfuri (care dau descrierea geometrica a primitivei) si printr-unul din tipurile prestabilite, care indica topologia, adica modul în care sunt conectate vârfurile între ele. Fiecare vârf este specificat prin intermediul unei functii `glVertex`, iar lista de vârfuri este delimitata între functiile `glBegin(GLenum mode)` si `glEnd()`. Aceeasi lista de vârfuri (v_0, v_1, v_2,…v_{n-1}) poate fi tratata ca puncte izolate, linii, poligon, etc, în functie de tipul primitivei, care este transmis prin argumentul `mode` al functiei `glBegin()` (`GL_POINTS`, `GL_LINES`, `GL_POLYGON`, etc).

De exemplu, desenarea unei primitive geometrice (un patrulater) în spatiul tridimensional se realizeaza în OpenGL prin secventa de apeluri de functii:

```
glBegin(GL_POLYGON);
   glVertex3d(-1.0, 1.0, 0.0);
   glVertex3d( 1.0, 1.0, 0.0);
   glVertex3d( 1.0,-1.0, 0.0);
   glVertex3d(-1.0,-1.0, 0.0);
glEnd();
```

Argumentele functiilor `glVertex3d()` reprezinta coordonatele în spatiul tridimensional ale vârfurilor patrulaterului. De fapt, "desenarea" unei suprafete (primitiva geometrica) înseamna reprezentarea pe display a proiectiei pe un plan de proiectie a suprafetei din spatiul tridimensional. Obtinerea imaginii pe display este rezultatul unei secvente de operatii grafice, care vor fi prezentate în lucrarile urmatoare.

Primitivele de tip suprafata (triunghiuri, patrulatere, poligoane) pot fi desenate în modul "cadru de sârma" (*wireframe*), sau în modul "plin" (*fill*), prin setarea variabilei de stare `GL_POLYGON_MODE` folosind functia

```
void glPolygonMode(GLenum face, GLenum mode);
```

unde argumentul `mode` poate lua una din valorile:

- `GL_POINT` : se deseneaza numai vârfurile primitivei, ca puncte în spatiu, indiferent de tipul acesteia.
- `GL_LINE`: muchiile poligoanelor se deseneaza ca segmente de dreapta.
- `GL_FILL`: se deseneaza poligonul plin.

Argumentul `face` se refera la tipul primitivei geometrice (din punct de vedere al orientarii), careia i se aplica modul de redare `mode`. Din punct de vedere al orientarii, OpenGL admite primitive orientate direct si primitive orientate invers. Argumentul `face` poate lua una din valorile: `GL_FRONT`, `GL_BACK` sau `GL_FRONT_AND_BACK`, pentru a se specifica primitive orientate direct, primitive orientate invers si, respectiv, ambele tipuri de primitive.

În mod implicit, sunt considerate orientate direct suprafetele ale caror vârfuri sunt parcurse în ordinea inversa acelor de ceas. Acesta setare se poate modifica prin functia `glFrontFace(GLenum mode)` unde `mode` poate lua valoarea `GL_CCW` pentru orientare în sens invers acelor de ceas (*counterclockwise*) sau `GL_CW` pentru orientare în sensul acelor de ceasornic (*clockwise*).

Reprezentarea culorilor în OpenGL. În biblioteca OpenGL sunt definite doua modele de culori: modelul de culori RGBA si modelul de culori indexate. În modelul RGBA sunt memorate componentele de culoare R, G, B si transparenta A pentru fiecare primitiva geometrica sau pixel al imaginii. În modelul de culori indexate, culoarea primitivelor geometrice sau a pixelilor este reprezentata printr-un index într-o tabela de culori (*color map*), care are memorate pentru fiecare intrare (index) componentele corespunzatoare R,G,B,A ale culorii. În modul de culori indexate nu se pot efectua unele dintre prelucrarile grafice importante (cum sunt umbrirea, anti-aliasing, ceata). Modelul de culori indexate este folosit în principal în aplicatii de proiectare grafica (CAD), în care

este necesar un numar mic de culori si nu se folosesc umbrirea, ceata, etc. În aplicatiile de realitate virtuala nu se poate folosi modelul de culori indexate si de aceea în continuare nu vor mai fi prezentate comenzile sau optiunile care se refera la acest model si toate descrierile considera numai modelul RGBA.

Culoarea care se atribuie unui pixel dintr-o primitiva geometrica depinde de mai multe conditii, putând fi o culoare constanta a primitivei, o culoare calculata prin interpolare între culorile vârfurilor primitivei, sau o culoare calculata în functie de iluminare, anti-aliasing si texturare. Presupunând pentru moment culoarea constanta a unei primitive, aceasta se obtine prin setarea unei variabile de stare a bibliotecii, variabila de culoare curenta (`GL_CURRENT_COLOR`). Culoarea curenta se seteaza folosind una din functiile `glColor#()`, care are mai multe variante, în functie de tipul si numarul argumentelor. De exemplu, doua din prototipurile acestei functii definite în fisierul gl.h sunt:

```
void glColor3f(GLfloat r, GLfloat g, GLfloat b);
void glColor4d(GLdouble r, GLdouble g, GLdouble b, GLdouble a);
```

Culoarea se poate specifica prin trei sau patru valori, care corespund componentelor rosu (r), verde (g), albastru (b), respectiv transparenta (a) ca a patra componenta pentru functiile cu 4 argumente.

1.2 DEZVOLTAREA APLICATIILOR GRAFICE OPENGL

Aplicatiile grafice bazate pe biblioteca OpenGL se pot dezvolta în mai multe moduri, în functie de cerintele programului.

În sistemul de operare Windows, biblioteca OpenGL poate fi folosita într-un program dezvoltat la nivelul interfetei de programare Win32 API. Mai avansata este însa includerea bibliotecii OpenGL în aplicatiile cadru oferite de biblioteca MFC (aplicatii SDI sau MDI), în care exista numeroase posibilitati de comunicatie cu utilizatorul.

Pe lânga aceste posibilitati, biblioteca grafica GLUT permite dezvoltarea aplicatiilor grafice independent de platforma de calcul.

În acest laborator se va folosi cu precadere dezvoltarea programelor grafice OpenGL în aplicatii cadru MFC (SDI) din mediul de dezvoltare MSVC 6.0, dar se vor experimenta si aplicatii GLUT.

1.2.1 DEZVOLTAREA APLICATIILOR GRAFICE OPENGL FOLOSIND BIBLIOTECA MFC

Un exemplu de program grafic OpenGL într-o aplicatie MFC - SDI este proiectul *Cube*. Acest proiect prezinta un obiect tridimensional (cub) care, la comanda *File-Play*, se roteste în jurul axelor sale. Handlerul acestei comenzi creeaza un timer (cu un interval de timp de 15 msec) iar în functia de tratare a mesajului `WM_TIMER` se modifica pozitia cubului si se apeleaza functiile OpenGL de desenare a cubului. Studiul acestui exemplu permite întelegerea aspectelor de baza ale programelor grafice OpenGL în aplicatii MFC si crearea cu usurinta a altor programe.

În continuare se vor prezenta pasii care trebuie sa fie executati pentru crearea unui program propriu, în care se pot experimenta diferite aspecte de grafica 3D.

Pasul 1. Se creeaza (folosind MFC App Wizard) un proiect SDI cu numele dorit (fiecare student poate folosi numele propriu; în explicatiile care urmeaza se va folosi ca nume al proiectului Lab3d). În aceasta faza se pot accepta toate optiunile implicite de creare a proiectului iar adaugirile care se vor efectua vor urmari în cea mai mare parte structura proiectului Cube (cu mici diferente, care sa permita dezvoltarea în continuare a proiectului).

În clasa `CLab3dView` se adauga variabilele membre:

```
CRect       m_oldRect;
CClientDC   *m_pDC;
GLfloat     m_fFovy;
GLfloat     m_fNearPlane, m_fFarPlane;
```

Prima variabila (`m_oldRect`) este folosita pentru memorarea dimensiunilor portii de afisare OpenGL (egala cu dimensiunile ferestrei vedere); variabila `m_pDC` este pointerul la contextul de dispozitiv al ferestrei vedere si se initializeaza cu `NULL` în constructorul clasei `CLab3dView`. Variabilele `m_fFovy`, `m_fNearPlane, m_fFarPlane` definesc piramida de vizibilitate a imaginii. Aceste variabile se initializeaza în constructor:

```
CLab3dView::CLab3dView(){
   m_pDC = NULL;
   m_fFovy = 45.0f;
   m_fNearPlane = 1.0f;
   m_fFarPlane = 100.0f;
}
```

Pasul 2. Se adauga bibliotecile OpenGL si GLU în proiect. Pentru aceasta se foloseste comanda *Project - Settings - Link*; în caseta combinata (combo-box) *Category* se selecteaza optiunea *General*, iar în caseta de editare *Object/Library modules* se introduc numele bibliotecilor *opengl32.lib* si *glu32.lib*. Bineînteles, bibliotecile respective (*opengl32.lib* si *glu32.lib*) trebuie sa se gaseasca în directorul MSVC\VC98\lib al mediului MSVC, iar bibliotecile DLL corespunzatoare (*opengl32.dll* si *glu32.dll*) trebuie sa existe în directorul sistemului de operare (*Windows\system*). De asemenea, în fisierul header `stdafx.h` se adauga fisierele header ale bibliotecilor OpenGL si GLU:

```
#include "gl\gl.h"
#include "gl\glu.h"
```

Aceste fisiere trebuie sa existe în directorul *MSVC\VC98\include\gl*. Toate fisierele aferente bibliotecii OpenGL se introduc în directoarele necesare în mod automat la instalarea mediului de dezvoltare MSVC 6.0, deci tot ceea ce este de facut este sa se verifice ca instalarea s-a efectuat corect.

Pasul 3. Se specifica stilul de prezentare al ferestrei vedere adaugând instructiunea:

```
cs.style |= WS_CLIPSIBLINGS | WS_CLIPCHILDREN;
```

în functia `BOOL CLab3dView::PreCreateWindow(CREATESTRUCT& cs)`.

Se asociaza contextul de redare OpenGL (*rendering context*) cu contextul de dispozitiv (*device context*) al ferestrei vedere a programului. Pentru aceasta se vor adauga functii de tratare a mesajelor `WM_CREATE` si `WM_DESTROY` în clasa vedere (CLab3dView).

Functia de tratare a mesajului `WM_CREATE` (`CLab3dView:: OnCreate()`) este apelata de aplicatia cadru la crearea ferestrei vedere si în aceasta functie se apeleaza functia `Init()` care face initializarea bibliotecii OpenGL. Completati functiile `OnCreate()` si `Init()` cu codul urmator:

```
int CLab3dView::OnCreate(LPCREATESTRUCT lpCreateStruct){
   if (CView::OnCreate(lpCreateStruct) == -1)
      return -1;
   Init(); // initializare OpenGL
   return 0;
}
void CLab3dView::Init(){
   // Selectie format pixel
   m_pDC = new CClientDC(this);
   ASSERT(m_pDC != NULL);
   if (!bSetupPixelFormat())return;
   // Asociere context de redare cu context disp. vedere
   HGLRC hrc = wglCreateContext(m_pDC->GetSafeHdc());
   wglMakeCurrent(m_pDC->GetSafeHdc(), hrc);
   GetClientRect(&m_oldRect);
   // Initializare stare OpenGL
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f);  // culoare stergere
   glClearDepth(1.0f);                    //adancimea maxima Z buffer
   glEnable(GL_DEPTH_TEST);               // validare Z buffer
}
```

În functia `Init()` se fac mai multe operatii de initializare. În primul rând se seteaza un anumit format de pixel, în functie de capacitatile sistemului grafic, prin apelul functiei `bSetupPixelFormat()`, care poate fi preluata din proiectul Cube. Pentru crearea unui context de redare OpenGL si asocierea acestuia cu contextul de dispozitiv al ferestrei de afisare se folosesc functiile extensiei `WGL` a bibliotecii OpenGL, extensie specifica sistemului de operare Windows. Crearea unui context de redare OpenGL (identificat prin handle `hrc`, de tip `HGLRC`) se face cu functia `wglCreateContext()`, iar asocierea acestui context cu contextul de dispozitiv al ferestrei vedere (`m_pDC`) se face cu functia `wglMakeCurrent()`. Dupa aceasta, în variabila `m_oldRect` se memoreaza dimensiunea ferestrei de afisare obtinuta cu functia `GetClientRect()`.

În ultima parte a functiei `Init()` se face initializarea variabilelor de stare OpenGL. Cea mai mare parte a variabilelor de stare OpenGL au valori implicite convenabile dezvoltarii majoritatii programelor grafice si numai putine dintre ele trebuie sa fie initializate explicit.În exemplul dat se seteaza culoare de stergere a buferului de culoare (`glClearColor`), se seteaza valoarea de stergere a buferului de adâncime si se valideaza testul de adâncime (`GL_DEPTH_TEST`). Diferite alte initializari ale starii OpenGL se pot adauga în aceasta functie.

Functia de tratare a mesajului `WM_DESTROY` (`CLab3dView::OnDestroy()`) este apelata de aplicatia cadru la distrugerea ferestrei vedere si în aceasta functie se elibereaza contextul de redare OpenGL si contextul de dispozitv creat:

```
void CLab3dView::OnDestroy() {
   //KillTimer(1);   // Numai daca a fost creat un timer
   HGLRC   hrc = ::wglGetCurrentContext();
   ::wglMakeCurrent(NULL,  NULL);
   if (hrc) ::wglDeleteContext(hrc);
   if (m_pDC)  delete m_pDC;
   CView::OnDestroy();
}
```

Pasul 4. Se adauga functia de tratare a mesajului `WM_SIZE` în clasa vedere, astfel încât, atunci când se modifica dimensiunea ferestrei vedere a aplicatiei, sa se modifice în mod corespuntator poarta de afisare OpenGL. Codul functiei `CLab3dView::OnSize()` se completeaza astfel (în mod asemanator cu cel din proiectul Cube):

```
void CLab3dView::OnSize(UINT nType, int cx, int cy) {
   CView::OnSize(nType, cx, cy);
   if(cy > 0){
      glViewport(0, 0, cx, cy);
      m_oldRect.right = cx;
      m_oldRect.bottom = cy;

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(45.0f,(GLdouble)cx/cy,m_fNearPlane,m_fFarPlane);
      glMatrixMode(GL_MODELVIEW);
   }
}
```

Aceasta functie este apelata de aplicatia cadru ori de câte ori se modifica dimensiunea ferestrei vedere a aplicatiei. În acesta functie se redefineste transformarea fereastra-poarta, cu functia `glViewport()`, având ca argumente noile dimensiuni `cx` si `cy` ale ferestrei si aceste dimensiuni se memoreaza în variabila `m_oldRect`.

În instructiunile urmatoare se defineste transformarea de perspectiva (cu functia `gluPerspective()`), care tine seama de unghiul de vizibilitate pe verticala (`m_fFovy`), de raportul dimensiunilor ferestrei de afisare (`cx/cy`) si de pozitia planelor de vizibilitate apropiata si departata (`m_fNearPlane` si `m_fFarPlane`) care au fost initializate în constructorul clasei vedere.

Pasul 5. Tot ceea se deseneaza într-un cadru de imagine se prevede într-o functie membra a clasei vedere, pe care o vom numi `DrawScene()`, iar pentru început aceasta poate fi chiar functia

`DrawScene()` din proiectul Cube. Aceasta functie se va apela ori de câte ori trebuie sa fie redesenata imaginea continuta în fereastra vedere, adica atunci când se schimba obiectul desenat, forma sau pozitia acestuia. De asemenea apelul functiei `DrawScene()` trebuie sa fie introdus si în functia `CLab3dView::OnDraw()`, astfel încât ea sa fie executata ori de câte ori se redeseneaza fereastra vedere (la modificarea dimensiunilor ferestrei, la maximizarea ferestrei dupa ce fusese minimizata, la descoperirea ferestrei dupa ce fusese acoperita de o alta fereastra, etc):

```
void CLab3dView::OnDraw(CDC* pDC){
   CLab3dDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   // TODO: add draw code for native data here
   DrawScene();
}
```

La compilarea si executarea proiectului se va obtine imaginea cubului în pozitia initiala, la fel ca la executia programului Cube. Se poate adauga cu usurinta miscarea de rotatie a cubului. Pentru aceasta se adauga o comanda de meniu (de exmplu *View -Play*), la actionarea careia se complementeaza variabila booleana `m_play` si, în functie de valoarea acesteia, se porneste sau se opreste un timer (cu durata de 30 ms), la fel ca în proiectul Cube:

```
void CLab3dView::OnViewPlay() {
   // TODO: Add your command handler code here
   m_play = m_play ? FALSE : TRUE;
   if (m_play) SetTimer(1, 30, NULL);
   else  KillTimer(1);
}
```

În clasa vedere (`CLab3dView`) se adauga functia de tratare a mesajului de timer `WM_TIMER`, astfel încât la fiecare interval de timer se executa functia :

```
void CLab3dView::OnTimer(UINT nIDEvent) {
   DrawScene();
   CView::OnTimer(nIDEvent);
   // Eat spurious WM_TIMER messages
   MSG msg;
   while(::PeekMessage(&msg, m_hWnd, WM_TIMER, WM_TIMER, PM_REMOVE));
}
```

Pasul 6. Programul dezvoltat pâna în acest punct asigura initializarea bibliotecii OpenGL si o verificare minimala a functionarii acesteia. În lucrarile urmatoare se vor studia diferite aspecte ale generarii imaginilor tridimensionale si se vor adauga noi comenzi si facilitati grafice.

O prima comanda care se va intoduce este necesara pentru selectarea afisarii unuia dintre mai multe obiecte tridimensionale sau grupe de astfel de obiecte (scene tridimensionale) posibile. Pentru aceasta se defineste un tip enumerare care sa diferentieze obiectele care se vor afisa; de exemplu:

```
enum SCENE_TYPE{
   CUBE = 1,
   RECTANGLE = 2, ….
};
```

În clasa `CLab3dView` se introduce o variabila membra: `SCENE_TYPE m_scene` care se initializeaza în constructor (de exemplu, cu valoarea `CUBE`).

Se introduce comanda de meniu `Scene` cu articolele `Cube` si `Rectangle`. Functia de tratare a mesajului de comanda (cu identificatorul `IDC_SCENE_CUBE`, respectiv `IDC_SCENE_RECTANGLE`) va seta la valoarea corespunzatoare variabila `m_scene`:

```
void CLab3dView::OnSceneCube() {
   m_scene = CUBE;
   DrawScene();              // redesenarea ferestrei imaginii
}
```

Pentru aceasta, functia de desenare `DrawScene()` preluata din proiectul Cube se va modifica astfel încât sa selecteze obiectul (scena) desenat în functie de variabila `m_scene` si sa permita modificarea pozitiei obiectelor în scena:

```
static BOOL     bBusy = FALSE;            //variabile pentru animatie
static GLfloat  wAngleY = 0.0f;
static GLfloat  wAngleX = 0.0f;
static GLfloat  wAngleZ = 0.0f;

void DrawCube(){
   glBegin(GL_QUAD_STRIP);
      glColor3f(1.0f, 0.0f, 1.0f);
      glVertex3f(-0.5f, 0.5f, 0.5f);
   ……..……
   glEnd();
}
void DrawRectangle(){
   glBegin(GL_POLYGON);
      glVertex3d(-1.0, 1.0, 0.0);
      glVertex3d( 1.0, 1.0, 0.0);
      glVertex3d( 1.0,-1.0, 0.0);
      glVertex3d(-1.0,-1.0, 0.0);
   glEnd();
}
void CLab3dView::DrawScene(){

   if(bBusy) return;    // test daca s-a terminat imaginea precedenta
   bBusy = TRUE;
   // Stergere imagine
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glPushMatrix();
      // glTranslatef(0.0f, 0.0f, -5);
      // Transformarea de observare
      glRotatef(-angleZv, 0.0,0.0,1.0);
      glRotatef(-angleXv, 1.0,0.0,0.0);
      glRotatef(-angleYv, 0.0,1.0,0.0);
      glTranslatef(-Xv, -Yv, -Zv);        // se initializeaza Zv = 5;
      // Transformarea de modelare (pozitionare) a obiectelor scenei
      glTranslatef(Xs,Ys,Zs);
      glRotatef(angleYs, 0.0,1.0,0.0);
      glRotatef(angleXs, 1.0,0.0,0.0);
      glRotatef(angleZs, 0.0,0.0,1.0);
      // Animatie (play)
      glRotatef(wAngleX, 1.0f, 0.0f, 0.0f);
      glRotatef(wAngleY, 0.0f, 1.0f, 0.0f);
      glRotatef(wAngleZ, 0.0f, 0.0f, 1.0f);

      glColor3f(1.0,0.0,0.0);             // culoare curenta
      switch (m_scene){
      case CUBE:
         DrawCube();
         break;
       case RECTANGLE:
         DrawRectangle();
         break;
      }  // end switch

   glPopMatrix();
   glFinish();
   SwapBuffers(wglGetCurrentDC());        // comutare buffer
   bBusy = FALSE;                         // s-a terminat o imagine
```

```
    if (m_play) {
       wAngleX += 1.0f;                    // actualizare pozitie play
       wAngleY += 10.0f;
       wAngleZ += 5.0f;
    }
}
```

Patratul este specificat în planul z = 0, are centrul în originea sistemului de referinta si latura egala cu 2 si are culoarea rosie.

În acest foarte simplu exemplu se poate urmari bucla de operatii OpenGL necesare pentru desenarea imaginilor:

- Stergerea buferului de imagine si a buferului de adâncime (`glClear()`);
- Transformarile de pozitionare (observare, modelare, animatie) stabilesc pozitia observatorului si a obiectelor în scena; aceste transformari vor fi detaliate în lucrarile urmatoare; în aceasta faza este suficient sa aveti grija ca toate variabilele de pozitie (`Xs`, `angleXs`, etc) sa fie intializate cu 0, cu exceptia variabilei `Zv` care trebuie setata la valoarea 5 la initializare (în constructorul clasei), pentru ca sa avem o imagine vizibila la executia programului.
- Desenarea primitivelor geometrice; coordonatele vârfurilor sunt transmise prin secventa de functii `glVertex()`, încadrata de functiile `glBegin()`, `glEnd()`. Culoarea primitivei se poate stabili în mai multe moduri, modul cel mai simplu fiind prin setarea culorii curente a bibliotecii OpenGL (cu functia `glColor3f()`).
- Comutarea buferelor (`SwapBuffers`), dupa ce toate operatiile OpenGL precedente au fost terminate, ceea ce se verifica prin functia `glFinish()`.

Pe lânga aceste operatii mai apar si altele, legate de manevrarea stivelor de matrice OpenGL (`glTranslate()`, `glPushMatrix()`, `glPopMatrix()`), care vor fi descrise ulterior.

1.2.2 DEZVOLTAREA PROGRAMELOR GRAFICE FOLOSIND BIBLIOTECA GLUT

Biblioteca GLUT permite crearea si administarea ferestrelor de afisare si a evenimentelor de intrare în aplicatii grafice OpenGL, în mod independent de platforma de calcul. Header-ul glut.h trebuie sa fie inclus în fisierele aplicatiei, iar biblioteca glut.lib (sau glut32.lib) trebuie legata (linkata) cu programul de aplicatie. În lucrarea de fata s-a folosit versiunea glut3.6, care poate fi preluata din Internet (www.sgi.com/pub/opengl/GLUT).

Sub GLUT, orice aplicatie se structureaza folosind mai multe functii *callback*. O functie *callback* este o functie care apartine programului aplicatiei si este apelata de un alt program, în acest caz sistemul de operare, la aparitia anumitor evenimente. În GLUT sunt predefinite câteva tipuri de functii callback pentru intializarea programului, redimensionarea ferestrei de afisare, desenarea fesrestrei si controlul dispozitivelor de intrare (tastatura si mouse). Aceste functii sunt scrise în aplicatie si pointerii lor sunt transmisi la înregistrare sistemului Windows, care le apeleaza (prin pointerul primit) în momentele necesare ale executiei.

Functii de control al ferestrei de afisare. Sunt disponibile cinci functii pentru controlul ferestrei de afisare a programului. În programele dezvoltate sub GLUT, pentru corelarea dintre poarta de afisare si fereastra de afisare se folosesc functiile si `glutInitDisplayMode()` si `glutInit()`. Functia `void glutInit(int* argc, char** argv)` initializeaza biblioteca GLUT folosind argumentele din linia de comanda; ea trebuie sa fie apelata înaintea oricaror alte functii GLUT sau OpenGL. Functia `void glutInitDisplayMode(unsigned int mode)` specifica caracteristicile de afisare a culorilor si a bufferului de adâncime si numarul de buffere de imagine. Parametrul `mode` se obtine prin SAU logic între valorile fiecarei optiuni.

Functia: `void glutInitWindowPosition(int x, int y)` specifica pozitia pe ecran a coltului stânga sus al ferestrei de afisare. Pentru definirea dimensiunii initiale a ferestrei de afisare se apeleaza functia `void glutInitWindowSize(int width, int heigth)`.

Crearea fereastrei în care se afiseaza contextul de redare (poarta) OpenGL are loc la apelul functiei `int glutCreateWindow(char* string)`.

Functii callback. Functiile callback se definesc în program si se înregistreaza în sistem prin intermediul unor functii GLUT. Ele sunt apelate de sistemul de operare atunci când este necesar, în functie de evenimentele aparute. Apelul `glutDisplayFunc(Display)` înregistreaza functia callback `Display()` care calculeaza si afiseaza imaginea. Argumentul functiei este un pointer la o functie fara argumente care nu returneaza nici o valoare. Functia `Display` (a aplicatiei) este apelata oridecâte ori este necesara desenarea ferestrei: la initializare, la modificarea dimensiunilor ferestrei, sau la apelul explicit al functiei `gluPostRedisplay()`.

Functia `void glutReshapeFunc(void(*Reshape)(int w, int h))` înregistreaza functia callback `Reshape()` care este apelata oridecâte ori se modifica dimensiunea ferestrei de afisare. Argumentul este un pointer la functia cu numele `Reshape` cu doua argumente de tip întreg si care nu returneaza nici o valoare. În aceasta functie, programul de aplicatie trebuie sa refaca transformarea fereastra-poarta, dat fiind ca fereastra de afisare si-a modificat dimensiunile.

Functia `glutKeyboardFunc(void(*Keyboard)(unsigned int key, int x, int y)` înregistreaza functia callback `Keyboard()` care este apelata atunci când se actioneaza o tasta. Parametrul `key` este codul tastei, iar `x` si `y` sunt coordonatele (relativ la fereastra de afisare) a mouse-ului în momentul actionarii tastei.

Functia `glutMouseFunc(void(*Mouse)(unsigned int button, int state, int x, int y)` înregistreaza functia callback `Mouse` care este apelata atunci când este apasat sau eliberat un buton al mouse-ului. Parametrul `button` este codul butonului (poate avea una din constantele `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON` sau `GLUT_RIGHT_BUTTON`). Parametrul `state` indica apasarea (`GLUT_DOWN`) sau eliberarea (`GLUT_UP`) al unui buton al mouse-ului. Parametrii x si y sunt coordonatele relativ la fereastra de afisare a mouse-ului în momentul evenimentului. Functia `glutMotionFunc(void(*Motion)(int x, int y)` înregistreaza functia callback `Motion` care este apelata la miscarea mouse-ului.

Executia unui program folosind toolkit-ul GLUT se lanseaza prin apelul functiei `glutMainLoop()`, dupa ce au fost efectuate toate initializarile si înregistrarile functiilor callback. Aceasta bucla de executie poate fi oprita prin închiderea ferestrei aplicatiei.

Generarea obiectelor tridimensionale. Multe programe folosesc modele simple de obiecte tridimensionale pentru a ilustra diferite aspecte ale prelucrarilor grafice. GLUT contine câteva functii care redau astfel de obiecte tridimensionale în modul wireframe sau cu suprafete pline (*filled*). Fiecare obiect este reprezentat într-un sistem de referinta local, dimensiunea lui poate fi transmisa ca argument al functiei, iar pozitionarea si orientarea în scena se face în programul de aplicatie. Exemple de astfel de functii:

```
void glutWireCube(GLdouble size);
void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
```

Programele GLUT au un mod specific de organizare, care provine din felul în care sunt definite si apelate functii callback. Acest mod va fi prezentat la primul exemplu de program OpenGL-GLUT si va fi reluat apoi si în alte exemple.

Pentru crearea unui program grafic bazat pe biblioteca GLUT în sistemele Windows se poate folosi tot mediul MSVC 6.0. Se creeaza un proiect de tipul *Win32 Console Application* si se selecteaza optiunea "*An empty project*". Programul dorit se scrie într-unul sau mai multe fisiere care se insereaza în proiectul creat. De exemplu, programul GLUT care deseneaza un patrat rosu în spatiul tridimensional, la fel ca programul precedent SDI, va contine fisierul sursa:

```
// Program HelloGlut.cpp
#include <GL/glut.h>
// Pozitionare obiecte scena
float Xs = 0.0f, Ys = 0.0f, Zs = 0.0f;
float angleXs = 0.0f, angleYs = 0.0f, angleZs = 0.0f;
// Pozitionare observator
float Xv = 0.0f, Yv = 0.0f, Zv = 5.0f;
float angleXv = 0.0f, angleYv = 0.0f, angleZv = 0.0f;
// Mouse
bool pressed = false;
int mouse_x;
int mouse_y;
// Functia de initializare a starii OpenGL
void Init(){
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f);  // culoare stergere
   glClearDepth(1.0f);                    //adancimea maxima Z buffer
   glEnable(GL_DEPTH_TEST);               // validare Z buffer
}
void DrawRectangle(){
   glBegin(GL_POLYGON);
      glVertex3d(-1.0, 1.0, 0.0);
      glVertex3d( 1.0, 1.0, 0.0);
      glVertex3d( 1.0,-1.0, 0.0);
      glVertex3d(-1.0,-1.0, 0.0);
   glEnd();
}
// Functia callback de desenare
void Display(void){
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glPushMatrix();
      // Transformarea de observare
      glRotatef(-angleZv, 0.0,0.0,1.0);
      glRotatef(-angleXv, 1.0,0.0,0.0);
      glRotatef(-angleYv, 0.0,1.0,0.0);
      glTranslatef(-Xv, -Yv, -Zv);        // se initializeaza Zv = 5;

      // Transformare de modelare (pozitionare) obiecte din scena
      glTranslatef(Xs,Ys,Zs);
      glRotatef(angleYs, 0.0,1.0,0.0);
      glRotatef(angleXs, 1.0,0.0,0.0);
      glRotatef(angleZs, 0.0,0.0,1.0);
      glColor3d(1.0, 0.0, 0.0);           // culoare curenta
      DrawRectangle()
   glPopMatrix();
   glFinish();
   glutSwapBuffers();                     // comutare buffer
}
// Functia CALLBACK de redimensionare a ferestrei
void Reshape(int w, int h){
    h = (h == 0) ? 1 : h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);  //selectie stiva PROJECTION
    glLoadIdentity();
    gluPerspective(45.0f, (GLdouble)w/h, 1.0f, 100.0f);
    glMatrixMode(GL_MODELVIEW);  // selectie stiva  MODELVIEW
    glLoadIdentity();
}
// Functia callback de tastatura
void Keyboard(unsigned char key, int x, int y){
   switch (key){
   // tastele z si Z modifica pozitia obiectelor
   case 'z':
```

```
      Zs -= 0.5;
      glutPostRedisplay();
      break;
   case 'Z':
      Zs += 0.5;
      glutPostRedisplay();
      break;

   }
}
// Functia callback de tratare evenimente de apasare butoane mouse
void Mouse(int button, int state, int x, int y){
   switch(button){
   case GLUT_LEFT_BUTTON:
      if (state == GLUT_DOWN){
         pressed = true;
         mouse_x = x;
         mouse_y = y;
      }
      else pressed = false;
      break;
   }
}
// Functia callback de tratare evenimente de miscare mouse
void Motion(int x, int y){
   if (pressed){
      float w = 500;
      float h = 500;
      angleYs += 180.0f*(x - mouse_x)/w;
      angleXs += 180.0f*(y - mouse_y)/h;
      mouse_x = x;
      mouse_y = y;
      glutPostRedisplay();
   }
}
// Functia principala a programului
int main(int argc, char** argv){
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize(500,500);
   glutInitWindowPosition(100,100);
   glutCreateWindow("OpenGL");
   Init();
   glutDisplayFunc(Display);
   glutReshapeFunc(Reshape);
   glutKeyboardFunc(Keyboard);      // Inregistrare func. Keyboard
   glutMouseFunc(Mouse);            // Inregistrare func. Mouse
   glutMotionFunc(Motion);          // Inregistrare func. Motion
   glutMainLoop();
   return 0;
}
```

Pentru compilarea si executia corecta a programului, trebuie adaugate bibliotecile OpenGL, GLU si GLUT. Pentru aceasta se foloseste comanda *Project-Settings-Link*; în caseta combinata *Category* se selecteaza optiunea *General*, iar în caseta de editare *Object/Library Modules* se adauga biblitecile *opengl32.lib*, *glu32.lib*, *glut32.lib*.

În acest program se pot remarca functiile strict necesare dezvoltarii unui program folosind biblioteca GLUT si echivalenta acestora cu functiile din proiectul MFC-SDI.

Functia de initializare `Init()` este echivalenta partii de setare a starii OpenGL din functia `Init()` definita în clasa vedere (CLab3dView); ea seteaza culoarea de stergere (prin apelul functiei

`glClearColor`), seteaza valoarea de stergere a buferului de adâncime si valideaza testul de adâncime (`GL_DEPTH_TEST`).

În functia `main()` se fac initializarile bibliotecilor GLUT si OpenGL, se înregistreaza functiile callback si se apeleaza functia de executie în bucla (`glutMainLoop()`).

Functia callback de desenare (denumita `Display()`) este echivalenta functiei `DrawScene()` din proiectul precedent; în exemplul prezentat ea contine doar partea de desenare a unui patrat.

Functia callback de redimensionare (denumita `Reshape()`) este echivalenta functiei `OnSize()` din proiectul precedent. Ea este invocata de aplicatia cadru GLUT atunci când se schimba dimensiunile ferestrei de afisare a imaginii si defineste transformarea fereastra-poarta (`glViewport()`) si transformarea de proiectie perspectiva (`gluPerspective()`, cu aceeasi parametri ca în proiectul precedent.

Functia callback de tratare a evenimentelor de tastatura (`Keyboard()`) permite modificarea pozitiei obiectelor în scena (coordonata Zs) la actionarea tastelor z, Z. Functiile callback de tratare a evenimentelor de mouse (`Mouse()` si `Motion()`) permit rotirea obiectului dupa axele x si y.

La executia programului se afisa un patrat a carui pozitie în spatiul tridimensional poate fi modificata folosind mouse-ul si tastatura.

Teme - Execitii

1 **Studiul culorilor în OpenGL**. În cele doua proiecte dezvoltate modificati culoarea de stergere (prin modificarea argumentelor functiei `glClearColor()`) si culoarea patratului desenat (prin modificarea argumentelor functiei `glColor3f()`).

2 **Modurile de desenare a suprafetelor.** Selectarea modului de desenare "plina" se specifica prin functia `glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)` care se poate introduce în functia de initializare. Aceasta este setarea implicita a bibliotecii OpenGL. Daca se modifica argumentul `GL_FILL` în `GL_LINE`, suprafetele sunt desenate sub forma "*cadru de sârma*" (*wireframe*), iar la valoarea `GL_POINT` se deseneaza numai vârfurile suprafetelor.

3 **Interpolarea culorilor la desenarea primitivelor geometrice.** Pentru interpolarea culorilor se specifica câte o culoare diferita pentru fiecare vârf al primitivei. Pentru obiectul `CUBE` desenat, exista deja acest mod de specificare. Pentru obiectul `RECTANGLE`, modificati functia `DrawRectangle()` astfel:

```
void DrawRectangle(){
   glBegin(GL_POLYGON);
      glColor3d(1.0, 0.0, 0.0);     // rosu
      glVertex3d(-1.0, 1.0, 0.0);
      glColor3d(0.0, 1.0, 0.0);     // verde
      glVertex3d( 1.0, 1.0, 0.0);
      glColor3d(0.0, 0.0, 1.0);     // albastru
      glVertex3d( 1.0,-1.0, 0.0);
      glColor3d(1.0, 1.0, 1.0);     // alb
      glVertex3d(-1.0,-1.0, 0.0);
   glEnd();
}
```

Selectarea modului de desenare cu interpolarea culorilor se specifica prin functia `glShadeModel(GL_SMOOTH)`, care se poate introduce în functia de initializare. Aceasta este setarea implicita a bibliotecii OpenGL. Daca se modifica argumentul din `GL_SMOOTH` în `GL_FLAT`, nu se mai efectueaza interpolarea culorilor.

4 În proiectul Lab3d introduceti o comanda de meniu pentru selectia culorilor (de exemplu, *Color*) cu articolele *Clear*, *Current*, etc. La fiecare comanda de selectie se va lansa un dialog de tipul

`CColorDialog`, prin care se selecteaza o culoare (de tipul `COLORREF`). Acest tip contine componenta R în octetul cel mai putin semnificativ, componenta G în al doilea octet, iar componenta B în al treilea octet. Aceste valori pot fi usor transformate în argumente ale functiilor OpenGL de setare a culorilor.

De exemplu, pentru setarea culorii de stergere a buferului de imagine argumentele functiei `glClearColor()` trebuie sa fie de tipul `GLclampf`, adica valori flotante în gama [0, 1]. Functia de tratare a mesajului de comanda a articolului `COLOR-CLEAR` poate arata în modul urmator:

```
void CLab3dView::OnColorClear() {
   // TODO: Add your command handler code here
   CColorDialog dlg;
   if (dlg.DoModal()==IDOK){
      COLORREF cr = dlg.GetColor();
      float red = (cr & 0xFF)/255.0;
      float green = ((cr & 0xFF00)>>8)/255.0;
      float blue = ((cr & 0xFF0000)>>16)/255.0;
      glClearColor(red,green, blue, 1.0);
      DrawScene();
   }
}
```

În proiectul Lab3d mai introduceti si comenzile necesare astfel încât obiectele scenei sa fie rotite la miscarea mouse-ului apasat pe suprafata ferestrei de redare, la fel ca în proiectul GLUT.

5 Introduceti comenzi de interfata care sa permita selectarea modurilor de functionare ale bibliotecii OpenGL privind redarea (plina sau wireframe) a primitivelor si interpolarea culorilor. Comenzile se pot introduce prin tastatura sau în meniu.

6 În proiectul GLUT înlocuiti în functia de afisare `Display()` functia de desenare a patratului (`DrawRectangle()`) cu desenarea unei sfere (sau a altor obiecte GLUT) folosind functiile de desenare corespunzatoare (`glutSolideSphere()`, `glutSolideCone()`, etc).

7 În proiecte bazate pe MFC se pot folosi si unele functii din biblioteca GLUT. Pentru aceasta se include fisierul `glut.h` în fisierul în care se apeleaza functii GLUT (cel mai obisnuit în fisierul *Lab3dView.cpp*) si se adauga biblioteca GLUT în proiect prin comanda *Project-Settings-Link*; în caseta combinata *Category* se selecteaza optiunea *General*, iar în caseta de editare *Object/Library Modules* se adauga bibliteca *glut32.lib*. Dupa adaugarea bibliotecii GLUT, se poate apela una din functiile de desenare obiecte GLUT de mai sus. În comanda de meniu `Scene` se adauga articolele corespunzatoare, iar în functia `DrawScene()` se trateaza cazul de desenare respectiv. De exemplu:

```
void CLab3dView::DrawScene(){
.....
   switch (m_scene){
   .....
   case SPHERE:
      glutSolidSphere(1.0, 64, 64);
   break;
   case CONE:
      glutSolidCone (1.0, 3.0, 64, 64);
      break;
   case TORUS:
      glutSolidTorus (0.275, 0.85, 64, 64);
      break;
   case TEAPOT:
      glutSolidTeapot(1.0);
      break;
   }
}
```

2. 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.

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 – B-rep*), este forma "clasica" 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.

2.1.1 REPREZENTAREA POLIGOANELOR

Un poligon este o regiune din plan marginita de o colectie finita de segmente de dreapta care formeaza un circuit închis simplu. Fie n puncte în plan, notate $v_0, v_1, \ldots, v_{n-1}$ si n segmente de dreapta $e_0 = v_0v_1$, $e_1 = v_1v_2$, $\ldots$. $e_{n-1} = v_{n-1}v_0$, care conecteaza perechi de puncte succesive în ordine ciclica, deci inclusiv conexiunea între ultimul punct si primul punct din lista. Aceste segmente marginesc un poligon, daca si numai daca:

(a) Intersectia fiecarei perechi de segmente adiacente în ordinea ciclica este un singur punct, continut de ambele segemente: $e_i \cap e_{i+1} = v_{i+1}$, pentru oricare i = 0,…, n–1.

(b) Segmente neadiacente nu se intersecteaza: $e_i \cap e_j = \emptyset$, pentru orice $j \neq i+1$.

Segmentele care marginesc un poligon (linia poligonala) formeaza un circuit închis (*ciclu*), deoarece segmentele sunt conectate capat la capat si ultimul segment conecteaza ultimul punct cu primul punct; ciclul este simplu deoarece segmentele neadiacente nu se intersecteaza.

Punctele v_i se numesc vârfurile poligonului (*vertices*); segmentele e_i se numesc muchii (sau laturi) ale poligonului. De remarcat ca un poligon contine n vârfuri si n muchii si ca muchiile sunt orientate, astfel încât formeaza un ciclu (circuit închis). O astfel de orientare a segmentelor se numeste orientare consistenta. În general, se foloseste ordinea de parcurgere în sensul invers acelor de ceasornic: daca se parcurg muchiile în sensul lor de definitie, interiorul poligonului este vazut întotdeauna în partea stânga (fig. 1).

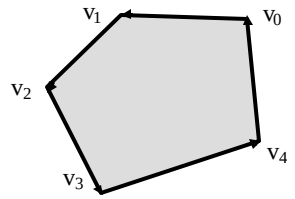

Fig. 1 Segmentele liniei poligonale sunt orientate si nu se autointersecteaza.

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 ?P în alte puncte decât vârfurile a si b, de început si de sfârsit ale segmentului.

Doua diagonale ale unui poligon sunt neîncrucisate (*noncrossing*) daca intersectia lor este o submultime a capetelor lor (punctele de început si de sfârsit ale segmentelor). Daca se adauga atâtea diagonale neîncrucisate câte sunt posibile într-un poligon, atunci poligonul este împartit în triunghiuri. O astfel de partitionare a unui poligon în triunghiuri se numeste *triangularizarea* poligonului. Diagonalele se pot adauga în orice ordine, atât timp cât sunt neîncrucisate. Demonstratia teoremei conform careia orice poligon admite o triangularizare se bazeaza pe teorema lui Meister, care stabileste ca orice poligon cu n ? 4 vârfuri admite cel putin o diagonala.

Teorema triangularizarii se bazeaza si pe lema numarului de diagonale: Orice triangularizare a unui poligon P cu n vârfuri utilizeaza n – 3 diagonale si consta din n – 2 triunghiuri. Aceste teoreme se demonstreaza prin inductie. În fig. 2.2 este prezentata triangularizarea unui poligon convex cu opt laturi; se insereaza 8 – 3 = 5 diagonale neîncrucisate si rezulta 8 – 2 = 6 triunghiuri.

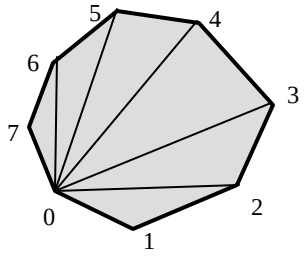

Fig. 2 Triangularizarea unui poligon convex.

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.

Din punct de vedere al reprezentarii în program a poligoanelor, cea mai simpla forma este reprezentarea printr-o lista liniara de vârfuri, fiecare vârf fiind specificat printr-o structura (sau clasa, în programarea orientata pe obiecte) care memoreaza (cel putin) coordonatele vârfului. Alte date referitoare la vârfurile poligoanelor necesare în modelarea si redarea obiectelor (normala, culoare, coordonate de texturare, etc.) vor fi descrise în capitolele care urmeaza.

Este posibila reprezentarea unui poligon si prin lista segmentelor sale, dar acesta reprezentare necesita un volum mai mare de date si este folosita în implementarea anumitor algoritmi de prelucrare a poligoanelor (reuniune, divizare, etc.) si mai putin în reprezentarea modelului unui obiect. Lista liniara de vârfuri poate fi implementata ca vector sau ca lista simplu sau dublu înlantuita.

2.1.2 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 bidimensional: 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.3).

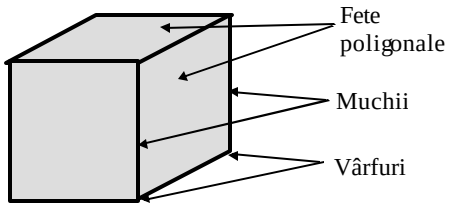

Fig.3 Reprezentarea prin suprafata de frontiera a unui poliedru.

Suprafata de frontiera a unui poliedru este o colectie finita de fete poligonale care se intersecteaza corect. Intersectia corecta a fetelor înseamna ca, pentru fiecare pereche de fete ale obiectului, fetele sunt disjuncte, au în comun un singur vârf, sau au în comun doua vârfuri si muchia care le uneste.

Din punct de vedere matematic, nu este imediat evident ca un solid poate fi reprezentat univoc prin suprafata care îl margineste. De aceea, este necesar sa fie stabilite conditiile în care aceasta reprezentare este permisa. Aceste conditii, numite conditii de *constructie corecta*, se definesc pentru suprafete de frontiera triangularizate. Triangularizarea unei suprafete poliedrale se obtine prin triangularizarea fiecarei fete poligonale, astfel încât suprafata rezultata consta din vârfuri care sunt înconjurate de triunghiuri, fiecare pereche de triunghiuri fiind adiacente de-a lungul unei muchii. Laturile triunghiurilor adiacente unui vârf formeaza un circuit de segmente, numit link-ul vârfului (fig. 4).

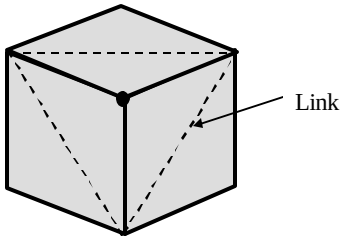

Fig. 4 Triangularizarea suprafetei de frontiera. Link-ul unui vârf al suprafetei.

O suprafata de frontiera construita corect îndeplineste urmatoarele conditii:

- Linkul fiecarui vârf al suprafetei triangularizate este complet, adica formeaza un circuit închis, nu neaparat planar.
- Triunghiurile suprafetei triangularizate sunt orientate consistent.

Închiderea link-ului fiecarui vârf asigura proprietatile suprafatei de a fi *închisa* si *conectata.* Proprietate de închidere înseamna ca suprafata nu are un sfârsit. Daca o suprafata nu este închisa sau nu este conectata, prin triangularizarea suprafetei nu se obtin link-uri închise [Kal89].

O suprafata de frontiera închisa, conectata si consistent orientata, împarte spatiul în doua parti: o parte interioara suprafetei, care este o regiune limitata, si o parte exterioara suprafetei, care este o regiune nelimitata.

Orientarea consistenta se verifica prin directia normalelor la fetele obiectului: daca normalele fetelor sunt îndreptate catre aceeasi regiune a spatiului (fie toate îndreptate spre interior, fie toate îndreptate spre exterior), atunci suprafata are o orientare consistenta. Acest mod de verificare se refera la obiectele tridimensionale fara cavitati, dar se poate extinde cu usurinta si la obiecte care prezinta cavitati.

Teoretic, orientarea consistenta se verifica pentru fetele triangularizate ale suprafetei de frontiera, dar, prin extindere, se pot folosi normalele la fetele poligonale, deoarece toate triunghiurile obtinute prin triangularizarea unui poligon care reprezinta o fata a unui obiect au aceeasi orientare (fig. 5).

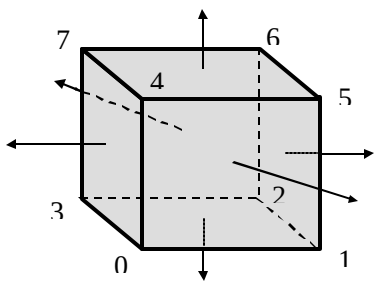

Fig. 5 Orientarea consistenta a fetelor cubului.

Orientarea consistenta a fetelor poligonale ale obiectelor este o conditie de verificare a constructiei corecte a suprafetei de frontiera si, în acelati timp, este folosita în operatiile de eliminare a suprafetelor ascunse în cursul redarii obiectelor tridimensionale.

Poligoanele care compun suprafata de frontiera care aproximeaza au poliedru au orientare consistenta daca normalele lor sunt toate îndreptate în acceasi regiune a spatiului: fie în exteriorul fie în interiorul poliedrului (în mod obisnuit spre exterior). Dat fiind ca sensul normalei este dat de ordinea de parcurgere a vârfurilor, aceasta conditie înseamna ca lista vârfurilor fiecarei fete poligonale trebuie sa fie parcursa în aceelasi sens (în mod obisnuit în sensul direct trigonometric –sensul invers acelor de ceas). De exemplu, pentru cubul din fig. 5, orientarea consistenta este asigurata daca poligoanele sunt specificate prin urmatoarele liste de vârfuri: (3, 2, 1, 0), (4, 5, 6, 7), (0, 1, 5, 4), (2, 3, 7, 6), (1, 2, 6, 5), (0, 4, 7, 3).

Modelul poligonal al unui obiect se poate genera prin mai multe metode, în functie de tipul obiectului si de aplicatia grafica în care este folosit modelul respectiv. Se poate folosi una din urmatoarele metode de modelare poligonala:

- Generarea modelului din descrierea matematica a obiectului.
- Generarea modelului obiectului prin baleiere spatiala.
- Generarea modelului pornind de la o multime de puncte care apartin suprafetei de frontiera a obiectului.

2.1.3 GENERAREA MODELULUI DIN DESCRIEREA MATEMATICA

Se poate genera reteaua de poligoane de aproximare a obiectelor care au o descriere matematica cunoscuta. De exemplu, ecuatiile unor suprafete quadrice:

- Elipsoid:

$x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 = 0$

unde a, b c sunt semiaxele elipselor.

- Hiperboloid:

$x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 = 0$ si

$x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 = 0$

- Paraboloid eliptic:

$x^2/a^2 + y^2/b^2 = z$

Surprafata se intersecteaza mai întâi cu un numar n de plane perpendiculare pe axa Oz, de ecuatii $z = n \cdot z$, pentru $n = -k, -k+1, .., -1, 0, 1, 2, \ldots, k-1, k$. Se obtin n elipse (paralele) si pe fiecare elipsa se esantioneaza m puncte echidistante (pe meridiane), obtinându-se $(n-1) \cdot m$ poligoane care aproximeaza suprafata de frontiera a obiectului respectiv.

Aceste suprafete se pot obtine si prin rotatia unei curbe în jurul unei axe de rotatie. De exemplu, suprafata elipsoidului se obtine prin rotatia în jurul axei z a elipsei:

$x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 = 0$

$y = 0$.

Prin rotatia unei curbe în jurul unei axe se pot obtine obiecte tridimensionale mai variate, în functie de forma curbei care se roteste. De exemplu, un tor se obtine prin rotatia unui cerc în jurul unei axe paralele cu planul cercului. Suprafetele astfel obtinute se numesc suprafete de rotatie.

2.1.4 GENERAREA MODELULUI PRIN BALEIERE SPATIALA

Se pot genera obiecte tridimensionale prin deplasarea (*sweeping*) unei suprafete generatoare de-a lungul unei curbe oarecare. Daca se variaza forma si orientarea suprafetei generatoare în cursul deplasarii, se pot obtine obiecte variate, în functie de forma curbei si de orientarea, forma si variatia formei suprafetei generatoare. Prin aceasta metoda se pot obtine atât formele regulate descrise mai sus

(elipsoid, hiperboloid, paraboloid eliptic, tor) cât si alte obiecte numite solide ductibile sau extrudate (*ducted solids*) sau cilindri generalizati (*generalized cylinders*) (fig. 6).

Pentru definirea deplasarii unei suprafete de-a lungul unei curbe, este necesar sa se defineasca intervalul curbei pe care are loc deplasarea si modul în care se divide intervalul parcurs. Împartirea intervalului în distante egale nu da rezultate bune, deoarece punctele obtinute nu vor fi egal distribuite pe suprafata obiectului. De aceea este necesara divizarea intervalului în functie de curbura curbei. Daca curbura este pronuntata, se aleg subdiviziuni mai mici ale intervalului, iar pentru curburi mai reduse se aleg subdiviziuni mai mari ale intervalului.

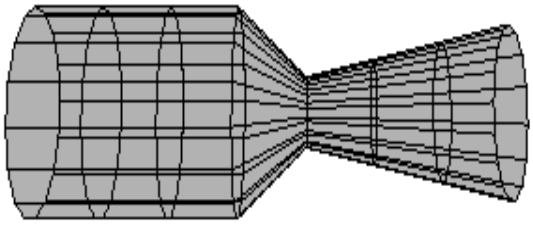

Fig. 6 Obiect poligonal modelat prin deplasarea unui cerc cu diametru variabil

Teme - Exercitii

8 **Modelul unui dreptunghi.** Obiectele redate pâna acum au fost realizate prin apeluri directe ale functiilor OpenGL, ceea ce este posibil numai pentru obiecte simple si cu forma bine precizata, în care se cunosc coordonatele vârfurilor suprafetelor componente.

Un model mai general al unui obiect tridimensional se compune din structuri de date care contin valori ale proprietatilor modelului (coordonatele vârfurilor, culori, texturi, etc.) iar la redarea obiectului, aceste structuri sunt parcurse, si valorile stocate sunt folosite ca argumente ale functiilor grafice de redare. Cel mai simplu model al unui poligon consta din lista vârfurilor sale.

Înlocuiti functia `DrawRectangle()` din proiectele realizate cu o functie care extrage coordonatele dintr-un model al dreptunghiului (o lista de vârfuri). De exemplu :

```
GLfloat rectangle[4][3] = {
   -1.0f,  1.0f, 0.0f,
    1.0f,  1.0f, 0.0f,
    1.0f, -1.0f, 0.0f,
   -1.0f, -1.0f, 0.0f
};
void DrawRectangle(){
   glBegin(GL_POLYGON);
      for (int i=0;i<4;i++){
         glColor3f(rectangle[i][0],rectangle[i][1],rectangle[i][2]);
         glVertex3f(rectangle[i][0],rectangle[i][1],rectangle[i][2]);
      }
   glEnd();
}
```

Se va obtine imaginea unui patrat cu interpolarea culorilor. Culorile vârfurilor patratului vor fi verde (0,1,0), galben (1,1,0), rosu (1,0,0) si negru (0,0,0), deoarece argumentele functiei `glColor3f()` sunt în mod implicit trunchiate la intervalul [0,1].

9 **Modelul unui poligon regulat în planul z = 0.** Sa se genereze modelul unui poligon regulat cu un numar oarecare de vârfuri într-un plan paralel cu unul din planele sistemului de referinta. Se va crea mai întâi modelul poligonului, prin calcularea si completarea listei vârfurilor sale. Ca exemplu, generati modelul unui poligon regulat cu 10 vârfuri în planul z = 0, în modul urmator:

```
// Generarea modelului poligonului regulat
int n = 10;
GLfloat coords[10*3];
for (i=0;i<n;i++){
   float angle = 2*3.14*i/n;
   coords[i*3+0] = cos (angle);
   coords[i*3+1] = sin(angle);
```

```
   coords[i*3+2] = 0;
}
```

Functia de redare a poligonului va parcurge lista vârfurilor si va apela functiile OpenGL corespunzatoare. Culoarea fiecarui vârf se poate seta egala cu coordonatele acestuia (trunchiate implicit în intervalul [0,1]).

```
void DrawPolygon(){
   glBegin(GL_POLYGON);
      for (int i=0;i<n;i++){
         glColor3f(coords[i*3],coords[i*3+1],coords[i*3+2]);
         glVertex3f(coords[i*3],coords[i*3+1],coords[i*3+2]);
      }
   glEnd();
}
```

10 Modelul unui poligon regulat într-un plan oarecare. Generalizati algoritmul de generare a modelului unui poligon regulat. În proiectul Lab3D introduceti o comanda de meniu *Scene-Polygon*, la actionarea careia se va lansa un dialog care sa permita stabilirea parametrilor poligonului: numarul de vârfuri si ecuatia planului în care se va genera poligonul regulat. Dupa stabilirea parametrilor poligonului, se memoreaza numarul de vârfuri (`n`) si se genereaza vectorul de coordonate (`float* coords`) de dimensiunea necesara în heap (cu operatorul `new`). Într-un proiect GLUT se pot introduce parametrii în linia de comanda.

11 Modelul cu fete separate al unui cub. În mod asemanator cu modelarea unui poligon, se va modifica functia de redare a cubului (`DrawCube()`), astfel încât aceasta sa parcurga modelul unui cub constituit din coordonatele vârfurilor celor sase fete ale cubului:

```
double cube[6][4][3] = {
    -1,-1,-1,  1,-1,-1,   1,-1, 1,   -1,-1, 1,   // fata 0
    -1, 1, 1,  1, 1, 1,   1, 1,-1,   -1, 1,-1,   // fata 1
    -1,-1, 1,  1,-1, 1,   1, 1, 1,   -1, 1, 1,   // fata 2
     1,-1,-1, -1,-1,-1,  -1, 1,-1,    1, 1,-1,   // fata 3
     1,-1, 1,  1,-1,-1,   1, 1,-1,    1, 1, 1,   // fata 4
    -1,-1, 1, -1, 1, 1,  -1, 1,-1,   -1,-1,-1    // fata 5
};
```

Culoarea fiecarei fete (sau vârf) al cubului se poate seta folosind chiar coordonatele vârfului respectiv (valorile acestora sunt trunchiate în mod implicit în intervalul [0,1] de functiile OpenGL pentru reprezentarea culorii).

12 Modelul cu fete indexate al unui cub. Definiti modelul unui cub folosind un vector de 8 vârfuri, iar fiecare fata specificata printr-o lista de indecsi, fiecare index reprezentând pozitia în vectorul de vârfuri a vârfului respectiv:

```
double cubeCoords[8][3]={
    -1,-1, 1,
     1,-1, 1,
     1,-1,-1,
    -1,-1,-1,
    -1, 1, 1,
     1, 1, 1,
     1, 1,-1,
    -1, 1,-1
};
int cubeIndex[6][4]={
   3, 2, 1, 0,
   4, 5, 6, 7,
   0, 1, 5, 4,
   2, 3, 7, 6,
   1, 2, 6, 5,
```

```
    0, 4, 7, 3
};
```

Modificati în mod corespunzator functia de redare a cubului. Aceasta forma de modelare a obiectelor, cunoscuta sub numele de "model cu fete indexate" (*indexed face set*) este cea mai frecvent folosita în limbajele de modelare (VRML) si în bibliotecile grafice de nivel înalt (Java 3D).

13 **Orientarea suprafetelor.** Studiati modul de redare a suprafetelor în functie de orientarea lor. În OpenGL, în mod implicit este dezactivata testarea orientarii suprafetelor si se redau toate suprafetele, indiferent de orientarea lor. Functia `glEnable (GL_CULL_FACE)` valideaza testarea orientarii fetelor si eliminarea acelora care au orientarea precizata prin functia `glCullFace(GLenum mode)`. Argumentul `mode` poate lua una din valorile `GL_FRONT` (orientare spre fata) sau `GL_BACK` (orientare spre spate); respectiv, vor fi eliminate suprafetele cu orientare `FRONT` sau cu orientare `BACK`.

Tipul de orientare `FRONT` al suprafetelor este este definit prin functia `glFrontFace(GLenum mode)`. Valoarea `GL_CCW` a argumentului `mode` defineste orientarea `FRONT` aceea în care parcurgerea listei de vârfuri a primitivei este în sens invers acelor de ceas (*Counter ClockWise*) si este setarea implicita a bibliotecii OpenGL; valoarea `GL_CW` (clockwise, în sensul acelor de ceas) a argumentului `mode` defineste orientarea `FRONT` aceea în care parcurgerea listei de vârfuri a primitivei este în sensul acelor de ceas (*ClockWise*).

Experimentati diferite situatii de eliminare a suprafetelor dupa orientarea lor, pastrând setarea implicita `glFrontFace(GL_CCW)`. De exemplu, daca ati introdus ordinea vârfurilor patratului (`RECTANGLE`) cea specificata în lucrare (în sensul acelor de ceas), atunci patratul este de tip `BACK` si, la validarea eliminarii suprafetelor cu orientare `BACK` (prin apelul functiei `glCullFace (GL_BACK)`), veti observa ca patratul nu se mai deseneaza. Pentru poligonul regulat generat cu functia descrisa mai sus, orientarea este de tip `FRONT` (în sens invers acelor de ceas) si va fi eliminat daca s-a apelat functia `glCullFace(GL_FRONT)`.

Se vor introduce comenzi de tastatura sau în meniu pentru selectarea diferitelor situatii de eliminare a suprafetelor în functie de orientare.

14 **Orientarea consistenta a fetelor poligonale.** Studiati si experimentati orientarea consistenta a fetelor poligonale ale suprafetei de frontiera care aproximeza un poliedru. De exemplu, pentru modelul cu fete indexate al cubului, ordinea vârfurilor data la punctul 2.5 asigura orientarea fiecarei în sensul invers acelor de ceas (care, în mod implicit înseamna orientare de tip `FRONT`). În aceasta situatie, prin eliminarea fetelor cu orientare `BACK` se vor desena (în mod corect) toate fetele vizibile (îndreptate catre observator). Daca schimbati orientarea uneia dintre fete sau stabiliti eliminarea fetelor `FRONT`, imaginea obtinuta este (în unele pozitii) incorecta.

15 Creati modelul cu fete indexate al unei piramide cu baza un poligon regulat (cu un numar oarecare de laturi) în planul y = 0, iar vârful pe axa Oy. Se va avea în vedere orientarea consistenta a fetelor poligonale. Asigurati triangularizarea poligonului de la baza piramidei fie explicit, prin introducerea diagonalelor necesare, fie prin argumentul corespunzator al functiei `glBegin()`. În proiectul Lab3D introduceti o comanda de meniu (*Scene-Pyramid*) si un dialog care sa permita introducerea parametrilor piramidei. În proiectul GLUT se pot introduce parametrii în linia de comanda.

16 Creati modele pentru urmatoarele forme quadrice: elipsoid, hiperboloid, paraboloid eliptic. În proiectul Lab3D introduceti comenzi de meniu (*Scene-Ellipsoide, etc*) si un dialog care sa permita introducerea parametrilor obiectului. În proiectul GLUT se pot introduce parametrii în linia de comanda.

17 Creati modelul unui obiect generat prin deplasarea unui poligon regulat de dimensiune variabila si cu numar variabil de vârfuri de-a lungul uneia din axele sistemului de referinta. Legea de variatie a dimensiunii poligonului poate fi stabilita printr-o secventa de valori.