Lucrarea nr. 5/6

Elemente de grafică în OpenGL

1. *Scopul lucrării:*

Lucrarea de faţă îşi propune prezentarea conceptelor şi tehnicilor de bază OpenGL. În această lucrare se prezintă funcţiile OpenGL puse la dispoziţia programatorilor pentru reprezentarea atributelor grafice ale imaginilor (iluminare, surse de lumină, materiale, texturi) şi modul de construire al aplicaţiilor grafice utilizând aceste funcţii.

2. *Noţiuni teoretice şi practice:*

2.1. Aplicaţii grafice folosind OpenGL sub sistemul de operare Windows

OpenGL şi MFC

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

1. **Folosind Microsoft Visual Studio, se creează un proiect de tip aplicaţie MFC AppWizard.**
2. **Se adaugă bibliotecile OPENGL32.lib GLU32.lib GLAUX.LIB la lista de biblioteci a editorului de legături.**
3. **Se adaugă în fişierul stdafx.h header-ele:**

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

2.2. Iluminare. Surse de lumină. Materiale.

În lumea reală, culoarea pe care o percepem pentru un obiect nu este determinată doar de culoarea propriu-zisă a obiectului, ci şi de lumina din mediul ce îl înconjoară (intensitate, culoare, direcţie). Pentru a avea efectul de iluminare cu ajutorul calculatorului avem nevoie de următoarele elemente:

- precizarea sistemului de surse luminoase (poziţie, tip, intensitate).
- precizarea proprietăţilor optice ale suprafeţelor (lucitoare, mată, transparentă).
- stabilirea poziţiilor relative ale suprafeţelor faţă de sursele luminoase.

Surse de lumină

OpenGL oferă programatorilor trei tipuri de surse luminoase:

- **Lumina ambiantă** – nu are o direcţie precizată din care este emisă şi este reflectată uniform şi în toate direcţiile de către suprafeţele obiectelor. Nu creează imagini realiste.
- **Lumina difuză** – are o direcţie bine precizată şi este reflectată uniform şi în toate direcţiile de către suprafeţele obiectelor. Creează efecte de tipul tonurilor de culoare.
- **Lumina speculară** – are direcţie bine precizată şi este reflectată după o direcţie particulară dată de legea reflecţiei (unghiul de incidenţă este egal cu unghiul de reflecţie).

Standardul OpenGL implementează cel mult GL_MAX_LIGHTS (opt în cazul implementării Windows) surse de lumină pentru o scenă, fiecare din acestea putând emite oricare din cele trei tipuri de lumină. Pentru stabilirea proprietăţilor unei surse de lumină (poziţia, intensitatea ambiantă/difuză/specular, coeficienţii de atenuare, direcţia, unghiul specular de deschidere, exponentul specular) se foloseşte una din variantele funcţiei:

```
void glLight(GLenum light, GLenum pname, const GLfloat* params);
```

unde:

- **`light`** este identificatorul sursei de lumină (**GL_LIGHT0 – GL_LIGHT7**).
- **`pname`** este denumirea proprietăţii ce este stabilită (**GL_POSITION, GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_SPOT_DIRECTION, GL_SPOT_CUTOFF, GL_SPOT_EXPONENT, GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, GL_QUADRATIC_ATTENUATION**).
- **params** (al cărui tip depinde de varianta funcţiei folosite) reprezintă valoarea proprietăţii.

Exemplul 1: *Context de randare pentru iluminare*

```
void StabilireContextRandare()
{
	// Intensitatea si pozitia

	GLfloat intensitate[]={1.0f,0.2f,0.2f,1.0f};
	GLfloat pozitie[]={-150.f,150.0f,-10.0f,0.0f};

	// Activare iluminare
	glEnable(GL_LIGHTING);

	// Stabilire proprietati si activare sursa de lumina 0
	glLightfv(GL_LIGHT0,GL_AMBIENT_AND_DIFFUSE,intensitate);
	glLightfv(GL_LIGHT0,GL_POSITION,pozitie);
	glEnable(GL_LIGHT0);
}
```

Pentru crearea efectului de iluminare trebuie combinate toate tipurile de lumină specificate pentru toate sursele luminoase ale scenei. Pentru activarea explicită a modului de lucru iluminare trebuie iniţializată variabila de mediu **GL_LIGHTING**.

```
glEnable(GL_LIGHTING);
```

Pentru configurarea modelului de iluminare se foloseşte funcţia:

```
glLightModelfv (GLenum pname, const GLfloat *params);
```

în care argumentul **`pname`** poate fi:

- GL_LIGHT_MODEL_AMBIENT – stabileşte intensitatea luminii ambiante pentru întreaga scenă.
- GL_LIGHT_MODEL_TWO_SIDE – indică dacă se iluminează sau nu ambele feţe ale unui poligon (implicit doar front-face-ul este iluminat).
- GL_LIGHT_MODEL_LOCAL_VIEWER – modifică modul de calcul al unghiurilor de reflecţie specular㠖.

Materiale

Noţiunea de **material** se referă la totalitatea proprietăţilor optice ale suprafeţei unui obiect.

Cea mai frecvent utilizată proprietate este, evident, culoarea suprafeţei. Culoarea unei suprafeţe precizează cantitatea de lumină reflectată de suprafaţa respectivă (ex. o suprafaţă albastră va reflecta doar lumina albastră incidentă, absorbind celelalte componente).

Alte proprietăţi ale materialului se referă la strălucirea, transparenţa sau emisia de lumină a suprafeţei. De exemplu, suprafeţele lucioase reflectă lumina specular㠖 absorbind o mare parte din lumina ambiantă şi difuză incidentă.

Pentru stabilirea proprietăţilor suprafeţei (materialului) se apelează funcţia **glMaterial** înainte de definirea suprafeţei.

```
glMaterialfv(GLint face, GLint pname, GLfloat[] params);
```

unde

- **face** indică partea poligonului pentru care se stabileşte materialul (GL_FRONT_AND_BACK, GL_FRONT sau GL_ BACK).
- **params** indică valoarea proprietăţii

```
glfloat cull[] = {1.0f, 0.0f, 0.0f,1.0f};
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,cull);
glBegin(GL_POLYGON);
//……..
glEnd();
```

În cele mai multe cazuri, componentele ambiante şi difuze sunt la fel, iar componenta specular㠖 apare doar în cazul materialelor strălucitoare. Pentru a putea stabili culoarea suprafeţelor utilizând funcţia glColor, trebuie activat modul “color tracking” pentru specificarea proprietăţilor suprafeţei:

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GLenum face, GLenum mode);
```

unde

- **face** indică partea poligonului.
- **mode** indică care proprietate a materialului va fi precizată prin glColor şi poate fi unul din următoarele: GL_EMISSION, GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR sau GL_AMBIENT_AND_DIFFUSE..

Exemplul prezentat anterior poate fi rescris astfel:

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
```

```
   glColor3f(1.0f,0.0f,0.0f,1.0f);
   glBegin(GL_POLYGON);
   //……..
   glEnd();
```

Exemplul 2: *Construire mediu de lucru OpenGL în care adăugăm o sursă de lumină difuză, specular şi ambiantă cu modul color tracking şi activarea iluminării (Figura 1).* ***Pentru a mări realismul implementării vom folosi o proiecţie perspectivă cu unghiul de deschidere de*** 45^o .

```
//variabile globale
GLint DELAY  = 0;

void CL2_1Dlg::OnBnClickedButton1()
{
      // Stabilirea ferestrei de lucru OpenGL
      auxInitDisplayMode(AUX_DOUBLE| AUX_RGBA);
      auxInitPosition(50, 50, 500, 500);
      auxInitWindow("Soare-Pamant-Luna");

      ConfigurareContextRandare();

      // Înregistrarea functiilor callback
      auxKeyFunc(AUX_UP, MaiRepede);
      auxKeyFunc(AUX_DOWN, MaiIncet);
      auxReshapeFunc(Redimensionare);
      auxIdleFunc(Desenare);
      auxMainLoop(Desenare);
}

// La redimensionarea ferestrei de lucru OpenGL, stabileste volumul de
// vedere si viewportul
void CALLBACK Redimensionare(GLsizei w, GLsizei h)
{
      if(h == 0) h = 1;

      glViewport(0, 0, w, h);

      // Stabilirea volumului de vedere folosind o proiectie perspectiva
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(45.0f, (GLfloat)w/(GLfloat)h, 1.0, 425.0);

// Initializeaza sistemul de coordonate al lumii reale (modelview) cu
//pozitia initiala
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
}

void ConfigurareContextRandare(void)
{
      GLfloat reflectieSpec[]={1.0f,1.0f,1.0f,1.0f};
      GLfloat glAmbient[]={0.2f,0.0f,0.0f,1.0f};

      // Valorile si coordonatele luminii
      GLfloat ambient[]={0.3f,0.3f,0.3f,1.0f};
      GLfloat difuza[]={0.7f,0.7f,0.7f,1.0f};
      GLfloat speculara[]={1.0f,1.0f,1.0f,1.0f};
      GLfloat pozitie[]={-50.f,50.0f,100.0f,1.0f};
```

```
	glEnable(GL_DEPTH_TEST);
	glFrontFace(GL_CCW);
	glEnable(GL_CULL_FACE);

	// Activare iluminare
	glEnable(GL_LIGHTING);
	glLightModelfv(GL_LIGHT_MODEL_AMBIENT,glAmbient);

	// Stabilire si activare sursa de lumina 0
	glLightfv(GL_LIGHT0,GL_AMBIENT,ambient);
	glLightfv(GL_LIGHT0,GL_DIFFUSE,difuza);
	// Adauga lumina speculara alba
	glLightfv(GL_LIGHT0,GL_SPECULAR,speculara);
	glLightfv(GL_LIGHT0,GL_POSITION,pozitie);
	glEnable(GL_LIGHT0);

	// Activare "color tracking"
	glEnable(GL_COLOR_MATERIAL);
	glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);

	// Materialul va reflecta toata lumina speculara
	glMaterialfv(GL_FRONT,GL_SPECULAR,reflectieSpec);
	glMateriali(GL_FRONT,GL_SHININESS,128);

	// Fundal albastru
	glClearColor(0.5f,0.5f,1.0f,1.0f);
}

void CALLBACK Desenare(void)
{
	// Unghiurile de rotatie ale Pamântului si Lunii
	static float fMoonRot = 0.0f;
	static float fEarthRot = 0.0f;
	GLfloat lightPos[] = {0.0f, 0.0f, 0.0f};

	Sleep(DELAY);

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	// Salveaza matricea modelului
	glMatrixMode(GL_MODELVIEW);
	glPushMatrix();

	// Stabileste pozitia sursei de lumina înainte de rotatie
	glLightfv(GL_LIGHT0,GL_POSITION,lightPos);

	// Translateaza întreaga scena in volumul de vedere
	glTranslatef(0.0f, 0.0f, -300.0f);

	// Deseneaza Soarele
	glColor3f(1.0f, 1.0f, 0.0f);
	auxSolidSphere(15.0f);

	// Deplaseaza sursa de lumina dupa desenarea Soarelui
	glLightfv(GL_LIGHT0,GL_POSITION,lightPos);

	// Roteste sistemul de coordonate pentru miscarea
	// de rotatie a Pamântului in jurul Soarelui
	glRotatef(fEarthRot, 0.0f, 1.0f, 0.0f);
```

```
	// Deseneaza Pamântul (bineînteles albastru;)
	glColor3f(0.0f, 0.0f, 1.0f);
	glTranslatef(105.0f,0.0f,0.0f);
	auxSolidSphere(15.0f);

	// Roteste Luna in jurul Pamântului
	glColor3f(0.8f,1.0f,0.8f);
	glRotatef(fMoonRot,0.0f, 1.0f, 0.0f);
	glTranslatef(30.0f, 0.0f, 0.0f);

	// Deseneaza Luna
	auxSolidSphere(6.0f);

	// Restaureaza matricea modelului de pe stiva
	glPopMatrix();

	// Calculeaza urmatoarea pozitie a Lunii
	fMoonRot+= 15.0f;
	if(fMoonRot > 360.0f)
		fMoonRot = 0.0f;

	// Calculeaza urmatoarea pozitie a Pamântului
	fEarthRot += 5.0f;
	if(fEarthRot > 360.0f)
		fEarthRot = 0.0f;
	glFlush();
	auxSwapBuffers();
}

void CALLBACK MaiRepede(void)
{
	if((DELAY-=50)<0) DELAY=0;
}

void CALLBACK MaiIncet(void)
{
	DELAY += 50;
}
```

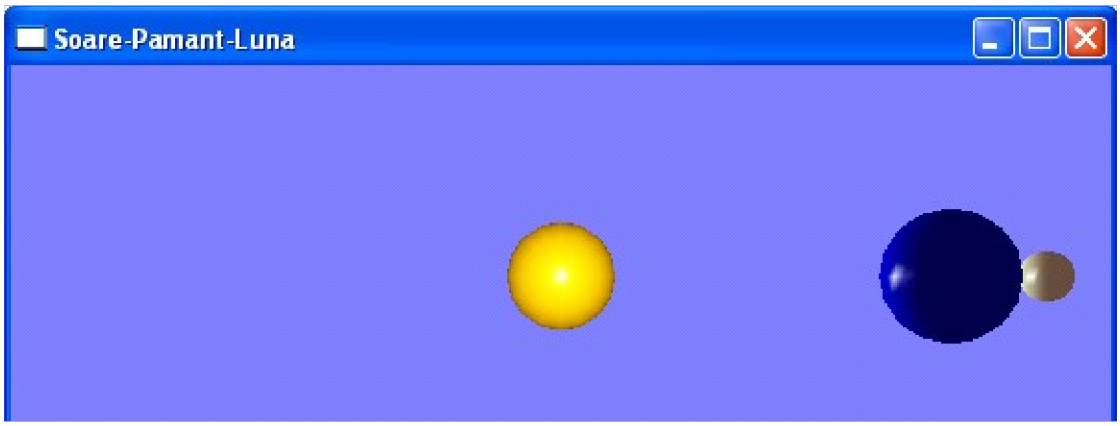

Figura 1 - Simularea sistemului Soare – Pământ – Lună folosind OpenGL

Efectele iluminării

Prin combinarea diverselor tipuri de surse luminoase şi proprietăţi ale suprafeţelor se obţin efecte realiste de iluminare. Lumina (şi proprietăţile) ambiantă şi difuză produc efecte suficiente pentru modelarea suprafeţelor mate de tipul lemnului, materialelor textile, solului, etc.

Pentru modelarea suprafeţelor lucioase (metal, faianţă, oglindă, etc.) sunt necesare lumini şi proprietăţi speculare. Cu cât valoarea factorului de multiplicare (specular exponent) este mai mare cu atât suprafaţa va fi mai lucitoare şi mai pronunţată (pentru toate implementările OpenGL acest parametru ia valori în intervalul 1 - 128).

Sursele de lumină de tip reflector (**spotlight**) modelează lumini direcţionate ce produc efecte de suprailuminare asemănătoare cu cele din lumea reală. Parametrul GL_SPOT_CUTOFF specifică unghiul conului de lumină emanată de reflector, astfel încât obiectele situate în afara conului de lumină nu vor fi iluminate de această sursă de lumină. În exemplul de mai jos (exemplul 3) este desenată o sferă iluminată de o sursă de lumină de tip reflector. Poziţia sursei de lumină este schimbată de rotirea scenei.

Exemplul 3: *Reflector (Figura 2)*

```
// Variabile globale
//Intensitatile si pozitia luminii
GLfloat pos[]={0.0f,0.0f,75.0f};
GLfloat spotDir[]={0.0f,0.0f,-1.0f};

void CALLBACK ModificaDimensiune(GLsizei w, GLsizei h)//Construirea
volumului de vedere
{
	GLfloat nRange = 100.0f;
	if (h==0) h = 1;

	// Stabilirea viewportului la dimensiunea ferestrei
	glViewport(0, 0, w, h);

	// Initializeaza matricea de proiectie cu matricea identitate
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	// Stabileste volumul de vedere folosind o proiectie ortografica
	if (w<=h)
		glOrtho(-nRange,nRange,-nRange*h/w,nRange*h/w,-nRange,nRange);
	else
		glOrtho(-nRange*w/h,nRange*w/h,-nRange,nRange,-nRange,nRange);

	// Initializarea matricii modelului
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}

// Initializarea contextului de randare
void CreareContextRandare()
{
	GLfloat specular[]={1.0f,1.0f,1.0f,1.0f};
	GLfloat specref[]={1.0f,1.0f,1.0f,1.0f};
	GLfloat glAmbient[]={0.5f,0.5f,0.5f,1.0f};

	glEnable(GL_DEPTH_TEST);
	glFrontFace(GL_CCW);
	glEnable(GL_CULL_FACE);
```

```
	// Activeaza modul de lucru iluminare
	glEnable(GL_LIGHTING);

	// Lumina ambianta globala
	glLightModelfv(GL_LIGHT_MODEL_AMBIENT,glAmbient);

	// Stabileste proprietatile sursei de lumina 0
	glLightfv(GL_LIGHT0,GL_DIFFUSE,glAmbient);
	glLightfv(GL_LIGHT0,GL_SPECULAR,specular);
	glLightfv(GL_LIGHT0,GL_POSITION,pos);
	glLightfv(GL_LIGHT0,GL_SPOT_DIRECTION,spotDir);

	// Specifica proprietatile reflectorului
	glLightf(GL_LIGHT0,GL_SPOT_CUTOFF,60.0f);
	glLightf(GL_LIGHT0,GL_SPOT_EXPONENT,100.0f);

	// Activeaza sursa de lumina 0
	glEnable(GL_LIGHT0);

	// Activeaza modul color tracking
	glEnable(GL_COLOR_MATERIAL);
	glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

	// Materialul obiectelor va avea reflectivitate speculara si o
	// stralucire pronuntata
	glMaterialfv(GL_FRONT,GL_SPECULAR,specref);
	glMateriali(GL_FRONT,GL_SHININESS,128);

	// Black background
	glClearColor(0.0f,0.0f,0.0f,1.0f);
}

// Desenarea scenei
void CALLBACK DeseneazaScena(void)
{
	GLfloat xRot=30.0f, yRot=45.0f;

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	// Salveaza matricea modelului
	glPushMatrix();
    // Roteste sistemul de coordonate
	glRotatef(yRot, 0.0f, 1.0f, 0.0f);
	glRotatef(xRot, 1.0f, 0.0f, 0.0f);

	// Stabileste noua pozitie si directie a
	// reflectorului in sistemul transformat
	glLightfv(GL_LIGHT0,GL_POSITION,pos);
	glLightfv(GL_LIGHT0,GL_SPOT_DIRECTION,spotDir);

	// Deseneaza un con in centrul reflectorului
	glColor3f(1.0f,0.0f,0.0f);
	glTranslatef(pos[0],pos[1],pos[2]);
	auxSolidCone(4.0f,6.0f);

	// Deseneaza o sfera mica pentru a simula un bec
	// Salveaza var. de stare pentru iluminarea
	glPushAttrib(GL_LIGHTING_BIT);

	// Dezactiveaza iluminarea pe perioada desenarii sferei mici
	glDisable(GL_LIGHTING);
```

```
	glColor3f(1.0f,1.0f,0.0f);
	auxSolidSphere(3.0f);

	// Restaureaza var. de stare pentru iluminare
	glPopAttrib();
	glPopMatrix();

	// Stabileste culoarea sferei si o deseneaza
	glColor3f(0.0f,0.0f,1.0f);
	auxSolidSphere(30.0f);
	glFlush();
}

void CL2_1Dlg::OnBnClickedButton1()
{
	// Stabilirea ferestrei de lucru OpenGL
	auxInitDisplayMode(AUX_SINGLE | AUX_RGBA);
	auxInitPosition(50, 50, 500, 500);
	auxInitWindow("Reflector în OpenGL");

	CreareContextRandare();

	// Înregistrarea functiilor callback
	auxReshapeFunc(ModificaDimensiune);
	auxMainLoop(DeseneazaScena);
}
```

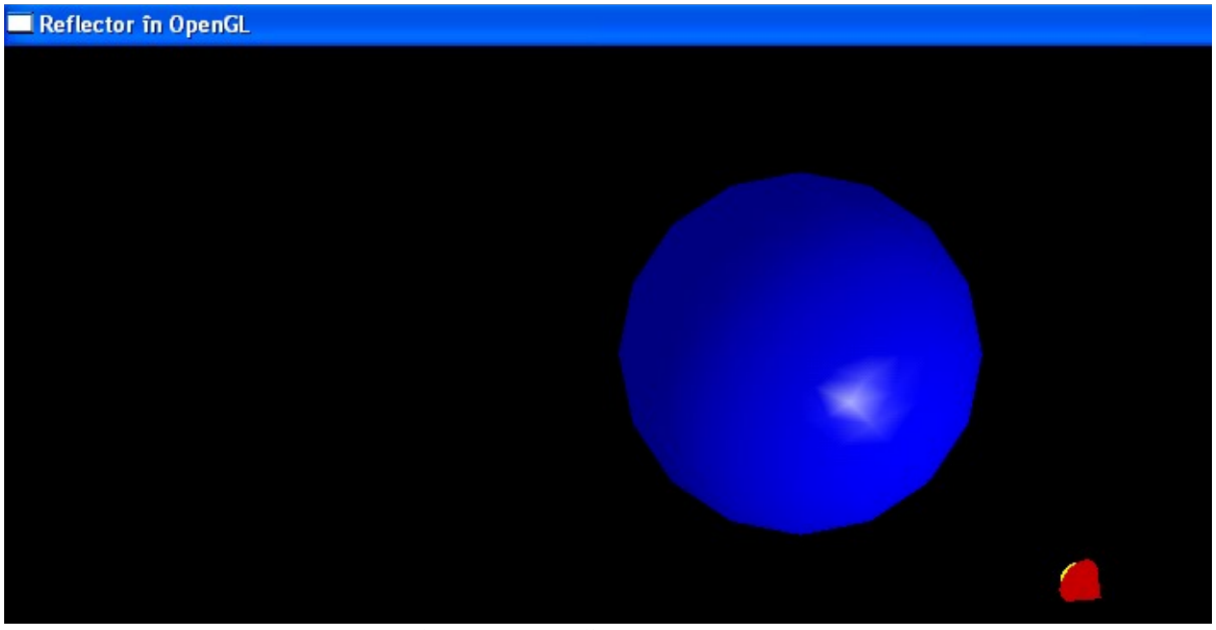

Figura 2 – Reflector în OpenGL

Un efect deosebit de important în realizarea scenelor realiste îl reprezintă umbrirea. Biblioteca OpenGL nu pune la dispoziţia programatorilor nici un fel de mecanisme pentru realizarea umbrelor. Acestea trebuie simulate prin program, de obicei prin proiecţia obiectului pe suprafaţa pe care dorim sa-i apară umbra. Zona obţinută prin proiecţie va fi desenată cu o culoare mai închisă.

Texturare

Texturarea este procedeul prin care se măreşte realismul scenelor tridimensionale prin ‘’aplicarea’’ unei imagini (textură) pe suprafaţa unui poligon.

Exemple de utilizare a texturilor pot fi găsite în toate jocurile pe calculator la simularea zidurilor, podelelor, tavanelor, monştrilor, etc.. Texturarea este un procedeu mare consumator de timp, dar există pe piaţă plăci grafice care implementează texturarea direct la nivel hardware. OpenGL suportă texturi uni-, bi- şi tridimensionale.

Lucrul cu texturi în OpenGL presupune următorii paşi:

- Crearea texturilor şi stocarea lor în memorie (memoria sistem sau memoria plăcii grafice dacă aceasta permite).
- Activarea modului texturare şi configurarea modului de combinare al culorilor texturilor şi culorilor pixelilor.
- Stabilirea coordonatelor de texturare.

Crearea texturilor

Definirea texturilor unidimensionale (reprezentate printr-un tablou de culori) se face utilizând funcţia:

```
void glTexImage1D (GLenum target, GLint level, GLint
components, GLsizei width, GLint border, GLenum format,
GLenum type, const Glvoid* pixels);
```

în care:

- **`target`** indică tipul texturii ce urmează să fie definită şi trebuie să ia valoarea GL_TEXTURE_1D.
- **`level`** reprezintă nivelul de detaliu al texturii (de obicei 0).
- **`components`** este numărul de culori pentru fiecare pixel (pentru culori RGB\RGBA ia valoare 3\4).
- **`width`** specifică dimensiunea texturii (fără pixelii de graniţă) şi trebuie să fie o putere a lui 2.
- **`border`** specifică numărul pixelilor de graniţă (şi poate fi 0, 1, 2).
- **`format`** indică tipul valorilor culorilor (valori valide sunt GL_COLOR_INNDEX, GL_LUMINANCE, GL_RGB sau GL_RGBA).
- **`type`** indică tipul de dată utilizat la specificarea componentelor culorii.
- **`pixels`** este tablou de culori ce defineşte textura propriu – zisă.

Pentru definirea texturilor bidimensionale (reprezentate printr-un tablou bidimensional de culori) se foloseşte funcţia:

```
void glTextImage2D (GLenum target, GLint level,
     GLint components, GLsizei width, GLsizei height,
     GLint border,GLenum format, GLenum type,
     Const GLvoid* pixels);
```

care primeşte un singur argument în plus faţă de funcţia `glTexImage1D` şi anume **`height`** ce specifică înălţimea texturii (o putere a lui 2).

Configurarea modului de texturare

Petru validarea modului de lucru texturare trebuie activate una din variabilele de stare GL_TEXTURE_1D sau GL_TEXTURE_2D. în cazul prezenţei texturilor, culoarea finală a unui pixel depinde de culoarea din imaginea de textură şi de culoarea pixelului primitivei geometrice. Modul în care se combină acestea se stabileşte prin apelul funcţiei:

```
void glTexEnv (GLenum target, GLenum pname, TYPE param);
```

unde:

- **target** specifică mediul de aplicare al texturii şi trebuie să fie GL_TEXTURE_ENV.
- **pname** este numele parametrului care urmează să fie definit conform tabelului de mai jos (Tabelul 1).

Tab. 1 – Parametrii funcţiei glTexEnv

Nume parametru	Descriere	Valori (param)
GL_TEXTURE_ENV_MODE	Tipul texturării	GL_DECAL GL_BLEND GL_MODULATE
GL_TEXTURE_ENV_COLOR	Culoarea ce va fi folosită la combinarea culorilor	Un pointer la o culoare RGBA

Stabilirea coordonatelor de texturare

Coordonatele vârfurilor primitivelor geometrice în planul texturii se stabilesc cu ajutorul functiei `glTexCoord` care poate primi 1, 2, 3 sau 4 argumente de diferite tipuri. De exemplu, varianta:

```
void glTexCoord2f (GLfloat s, GLfloat t);
```

stabileşte coordonata orizontală (s) şi verticală (t) în imaginea texturii.

Sistemul de coordonare al texturării este notat cu s, t, r şi acestea corespund coordonatelor x, y, z din spaţiul 3D. Funcţia:

```
GLTexParameter (GLenum target, GLenum pname, param);
```

stabileşte modurile de tratare a cazurilor în care coordonatele texturilor depăşesc intervalul [0, 1]. Argumentele:

- **target** indică tipul texturii (GL_TEXTURE_1D sau GL_TEXTURE_2D).
- **pname** indică parametrul care urmează a fi iniţializat cu valoarea **param** (al cărei tip depinde de varianta funcţiei).

  În tabelul 2 prezentăm valorile disponibile pentru argumentele **pname** şi **param**.

Tab. 2 – Parametrii funcţiei **glTexParameter**

Nume parametru	Valori (parametru)
GL_TEXTURE_MAG_FILTER • Metoda (filtrul) utilizată la mărirea texturii	GL_NEAREST Gl_LINEAR GL_NEAREST_MIPMAP_NEAREST GL_NEAREST_MIPMAP_LINEAR GL_LINEAR_MIPMAP_NEAREST GL_LINEAR_MIPMAP_LINEAR
GL_TEXTURE_MIN_FILTER	GL_NEAREST

• Metoda (filtrul) utilizată la micşorarea texturii	GL_LINEAR GL_NEAREST_MIPMAP_NEAREST GL_NEAREST_MIPMAP_LINEAR GL_LINEAR_MIPMAP_NEAREST GL_LINEAR_MIPMAP_LINEAR
GL_TEXTURE_WRAP_S • Modul de tratarea al coordonatei S în afara intervalului [0,1]	GL_REPEAT GL_CLAMP
GL_TEXTURE_WRAP_T • Modul de tratare al coordonatei T în afara intervalului [0,1]	GL_REPEAT GL_CLAMP
GL_TEXTURE_BORDER • Culoarea de graniţă ce va fi utilizată pentru texturile fără graniţă	o culoare RBGA

În exemplul 5 prezentăm etapele ce trebuie parcurse la texturarea obiectelor. Vom simula un curcubeu folosind o textură unidimensională. Pentru accelerarea vitezei de execuţie, textura va fi stocată într-o **listă de afişare** compilată (*display list*) cu ajutorul funcţiilor **glNewList/glEndList**.

Lista de afişare este o secvenţă compilată de comenzi şi funcţii OpenGL (împreună cu rezultatele lor) stocată în contextul de randare.

Pentru a executa ulterior o astfel de listă se foloseşte funcţia **glCallList(GLint id)**. O astfel de execuţie este mult mai rapidă decât refacerea tuturor calculelor stocate în ea.

Exemplul 5: *Curcubeu în OpenGL. (Figura 4)*

```
//Pentru a folosi textura 1D normala activati apelul CreareTextura1D();
//in functia atasata butonului

//variabile globale
#define M_PI          3.14157
#define RAZA_INTERIOR 50
#define RAZA_EXTERIOR 60
#define PAS           0.03125
GLfloat nRange = 100.0f;

void CALLBACK ModificaDimensiune(GLsizei w, GLsizei h)
{
      if (h==0) h = 1;
      // Stabilirea viewportului la dimensiunea ferestrei
      glViewport(0, 0, w, h);
      // Initializeaza matricea de proiectie cu matricea identitate
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();

      // Stabileste volumul de vedere folosind o proiectie ortografica
      if (w<=h)
            glOrtho(-nRange,nRange,-nRange*h/w,nRange*h/w,-nRange,nRange);
      else
            glOrtho(-nRange*w/h,nRange*w/h,-nRange,nRange,-nRange,nRange);

      // Initializarea matricii modelului
      glMatrixMode(GL_MODELVIEW);
```

```
	glLoadIdentity();
}

GLint RainbowTexture;
void CreareTextura1D()
{
	static unsigned char curcubeu[8][3] =
	{
	{ 0x3f, 0x00, 0x3f }, /*violet inchis*/
	{ 0x7f, 0x00, 0x7f }, /*violet*/
	{ 0xbf, 0x00, 0xbf }, /*indigo*/
	{ 0x00, 0x00, 0xff }, /*albastru*/
	{ 0x00, 0xff, 0x00 }, /*verde*/
	{ 0xff, 0xff, 0x00 }, /*galben*/
	{ 0xff, 0x7f, 0x00 }, /*portocaliu*/
	{ 0xff, 0x00, 0x00 }  /*rosu*/
	};

	glNewList(RainbowTexture=glGenLists(1),GL_COMPILE);
	glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER,GL_LINEAR);
	glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER,GL_LINEAR);
	glTexImage1D(GL_TEXTURE_1D, 0, 3, 8, 0,   GL_RGB, GL_UNSIGNED_BYTE,
	curcubeu);
	glEndList();
}

void CALLBACK DesenareCurcubeu(void)
{
	GLfloat th;

	glClear(GL_COLOR_BUFFER_BIT);

	// Activarea texturarea 1D
	glDisable(GL_TEXTURE_2D);
	glEnable(GL_TEXTURE_1D);

	// Stabilirea modului de combinare a culorilor
	glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

	// Utilizarea texturii
	glCallList(RainbowTexture);

	// Definirea curcubeului (arc de cerc) si a coordonatelor texturii
	glBegin(GL_QUAD_STRIP);
	for (th = 0.0; th <= M_PI; th += (GLfloat)(PAS * M_PI))
	{
		// Stabilirea cordonatei texturii in varful
		// ce urmeaza sa fie definit
		glTexCoord1f(0.0);
		glVertex3f((GLfloat)cos(th)*RAZA_INTERIOR,
				(GLfloat)sin(th)*RAZA_INTERIOR, 0.0f);

		glTexCoord1f(1.0);
		glVertex3f((GLfloat)cos(th)*RAZA_EXTERIOR,
				(GLfloat)sin(th)*RAZA_EXTERIOR, 0.0f);
	}
	glEnd();
	glFlush();
}

void RedesenareFereastra()
```

```
{
	RECT rect;
	GetClientRect(auxGetHWND(), &rect);
	SendMessage(auxGetHWND(), WM_SIZE, 0, MAKELONG(rect.right-
rect.left,rect.bottom-rect.top));
	DesenareCurcubeu();
}

void CALLBACK MaiAproape(void)
{
	if((nRange-=100)<100) nRange=100.0f;
	RedesenareFereastra();
}

void CALLBACK MaiDeparte(void)
{
	nRange += 100;
	RedesenareFereastra();
}

void CL2_1Dlg::OnBnClickedButton1()
{
	// Stabilirea ferestrei de lucru OpenGL
	auxInitDisplayMode(AUX_SINGLE | AUX_RGBA);
	auxInitPosition(50, 50, 500, 500);
	auxInitWindow("Curcubeu");

	 CreareTextura1D();

	// Înregistrarea functiilor callback
	auxKeyFunc(AUX_UP, MaiAproape);
	auxKeyFunc(AUX_DOWN, MaiDeparte);
	auxReshapeFunc(ModificaDimensiune);
	auxMainLoop(DesenareCurcubeu);
}
```

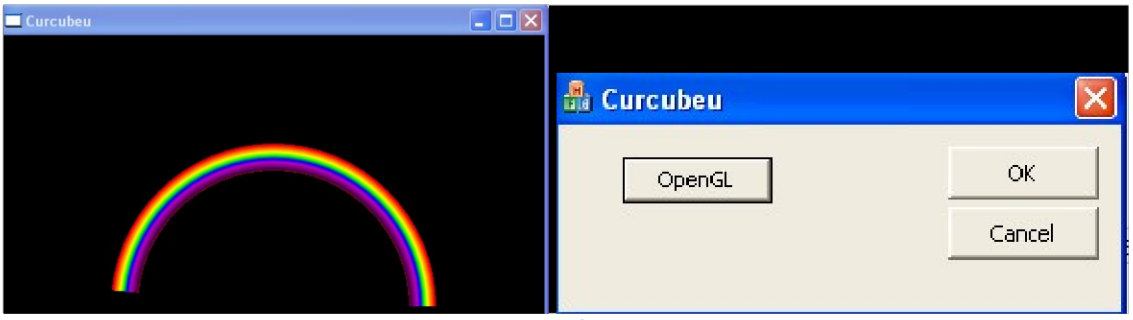

Figura 4 – Curcubeu în OpenGL

Deşi în exemplul de mai sus (exemplul 5) coordonatele texturii au fost simplu de calculat, în cazurile reale, calcularea coordonatelor texturii este o activitate complexă. Această activitate poate fi automatizată cu ajutorul funcţiei **`glTexGen`** pentru definirea coordonatelor s şi t în funcţie de coordonatele x şi z din scenă:

```
void  glTexGen (GLenum coord , GLenum pname, TYPE *params)
```

în care:

- **`coord`** precizează coordonata texturii ca va fi generată şi poate fi GL_S, GL_T, GL_R sau GL_Q.

- **pname** specifică vectorul care va fi definit (GL_OBJECT_PLANE, GL_TEXTURE_GEN_MODE sau GL_EYE_PLANE). Dacă **pname** ia valoarea GL_TEXTURE_GEN_MODE, **param** ia una din următoarele valori:
  - GL_OBJECT_LINEAR coordonatele texturii vor fi calculate folosind coordonatele obiectului (vârfurilor).
  - GL_EYE_LINEAR coordonatele texturii vor fi calculate folosind coordonatele din sistemul privitorului (adică coordonatele obiectului multiplicate cu matricea modelului).
  - GL_SPHERE_MAP coordonatele texturii vor fi generate în jurul poziţiei de vedere.

Dacă **pname** ia valoarea GL_OBJECT_PLANE sau GL_EYE_PLANE, argumentul **param** este un tablou de 4 elemente care este folosit ca un multiplicator la calcularea coordonatelor obiect sau privitor. Daca se foloseşte modul GL_OBJECT_LINEAR coordonatele texturii sunt generate după formula:

coord = v[0]*X + v[1]*Y + v[2]*Z + v[3]*W

unde v este tabloul trimis ca parametru. Pentru calcularea automata a coordonatelor cu funcţia descrisă mai sus este nevoie şi de activarea variabilelor de stare GL_TEXTURE_GEN_S, GL_TEXTURE_GEN_T, GL_TEXTURE_GEN_R sau GL_TEXTURE_GEN_Q cu ajutorul funcţiei glEnable.

Texturi multi-nivel

Texturile prezentate până acum aveau asociată o singură imagine de texturare. OpenGL suportă texturi cu imagini de texturare multiple, numite texturi ***mipmapped.*** O textură mipmapped va selecta imaginea de texturare cea mai apropiată de rezoluţia poligonului. Efectele vizuale sun astfel mult îmbunătăţite şi viteza de generare a poligoanlor cu astfel de texturi se îmbunătăţeşte la rândul ei. Totuşi, crearea texturilor mipmapped ia ceva mai mult timp deoarece se va crea o imagine de texturare distinctă pentru fiecare nivel de detaliu (rezoluţie). Acest lucru se face prin transmiterea de valori diferite pentru parametrul **level** al funcţiilor **glTexImage1D** sau **glTexImage2D**. Imaginea de texturare stabilită pentru valoarea 0 a acestui parametru se numeşte ***imagine de texturare primară*** şi este cea cu rezoluţia cea mai ridicată. Imaginea pentru nivelul 1 este jumătate din cea primară, ş.a.m.d. Când se desenează poligoane folosind texturi multinivel, parametrul GL_TEXTURE_MIN_FILTER trebuie să fie iniţializat cu una din următoarele valori (vezi şi funcţia glTexParameter): GL_NEAREST_MIPMAP_* sau GL_LINEAR_MIPMAP_*.

OpenGL pune la dispoziţia programatorilor funcţiile auxiliare:

```
void gluBuild1DMipmaps(GLenum target, GLint levels,
GLsizei width, GLint border,  GLenum format, GLenum
type, const GLvoid *primaryImage);
void gluBuild2DMipmaps(GLenum target, GLint levels,
GLsizei width, GLsizei height, GLint border,  GLenum
format, GLenum type, const GLvoid *primaryImage);
```

pentru generarea automată a imaginilor de texturare pentru nivelele >=1 pe baza imaginii de texturare primară.

Transformări geometrice ale texturilor

Operaţiile de texturare în OpenGL sunt implementate tot cu ajutorul matricelor 4*4. pe lângă matricea modelului şi a proiecţiei, a treia matrice de transformare OpenGL este ***matricea de texturare***. Selectarea matricii texturării ca matrice de lucru se face cu apelul glMatrixMode(GL_TEXTURE). Aplicarea transformărilor geometrice unei texturi produc efecte de deformare (glScale), de rotire (glRotate) sau translaţie (glTranslate) a texturii.

Exemplul 6: *Ceainic texturat cu o textură (textura este randată ca dungi pe suprafaţa ceainicului) având coordonatele generate automat. (Figura 5)*

```
//variabilele mele - globale
#define stripeImageWidth 32
GLubyte stripeImage[4*stripeImageWidth];

void makeStripeImage(void)
{
   int j;

   for (j = 0; j < stripeImageWidth; j++) {
      stripeImage[4*j] = (Glubyte) ((j<=4) ? 255 : 0);
      stripeImage[4*j+1] = (Glubyte) ((j>4) ? 255 : 0);
      stripeImage[4*j+2] = (Glubyte) 0;
      stripeImage[4*j+3] = (Glubyte) 255;
   }
}

// Planele utilizate la generarea coordonatelor texturii
static Glfloat xequalzero[] = {1.0, 0.0, 0.0, 0.0};
static Glfloat slanted[] = {1.0, 1.0, 1.0, 0.0};
static Glfloat *currentCoeff;
static Glenum currentPlane;
static Glint currentGenMode;

void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_SMOOTH);

   makeStripeImage();
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

   glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S,GL_REPEAT);
   glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
   glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
   glTexImage1D(GL_TEXTURE_1D, 0, 4,
                stripeImageWidth, 0, GL_RGBA,
                GL_UNSIGNED_BYTE, stripeImage);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

   currentCoeff = xequalzero;
   currentGenMode = GL_OBJECT_LINEAR;
   currentPlane = GL_OBJECT_PLANE;
   glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
   glTexGenfv(GL_S, currentPlane, currentCoeff);

   glEnable(GL_TEXTURE_GEN_S);
   glEnable(GL_TEXTURE_1D);
   glEnable(GL_CULL_FACE);
   glEnable(GL_LIGHTING);
```

```
   glEnable(GL_LIGHT0);
   glEnable(GL_AUTO_NORMAL);
   glEnable(GL_NORMALIZE);
   glFrontFace(GL_CW);
   glCullFace(GL_BACK);
   glMaterialf (GL_FRONT, GL_SHININESS, 64.0);
}

void CALLBACK display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glPushMatrix ();
   glRotatef(45.0, 0.0, 0.0, 1.0);
   auxSolidTeapot(2.0);
   glPopMatrix ();
   glFlush();
   auxSwapBuffers();
}

void CALLBACK reshape(int w, int h)
{
   glViewport(0, 0, (Glsizei) w, (Glsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h)
      glOrtho (-3.5, 3.5, -3.5*(Glfloat)h/(Glfloat)w,
               3.5*(Glfloat)h/(Glfloat)w, -3.5, 3.5);
   else
      glOrtho (-3.5*(Glfloat)w/(Glfloat)h,
               3.5*(Glfloat)w/(Glfloat)h,
               -3.5, 3.5, -3.5, 3.5);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void CALLBACK keyboard_1()
{
    currentCoeff = slanted;
    glTexGenfv(GL_S, currentPlane, currentCoeff);
    display();
}

void CALLBACK keyboard_2()
{
    currentCoeff = xequalzero;
    glTexGenfv(GL_S, currentPlane, currentCoeff);
    display();
}

void CALLBACK keyboard_3()
{
   currentGenMode = GL_EYE_LINEAR;
   currentPlane = GL_EYE_PLANE;
   glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
   glTexGenfv(GL_S, currentPlane, currentCoeff);
   display();
}

void CALLBACK keyboard_4()
{
```

```
    currentGenMode = GL_OBJECT_LINEAR;
    currentPlane = GL_OBJECT_PLANE;
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
    glTexGenfv(GL_S, currentPlane, currentCoeff);
    display();
}

void CL2_1Dlg::OnBnClickedButton1()
{
   // Stabilirea ferestrei de lucru OpenGL
   auxInitDisplayMode(AUX_DOUBLE|AUX_RGBA);
   auxInitPosition(100, 100, 256, 256);
   auxInitWindow("Ceainic texturat");

   // initializare
   init();
   // Înregistrarea functiilor callback
   auxReshapeFunc(reshape);
   auxKeyFunc(AUX_1, keyboard_1);
   auxKeyFunc(AUX_2, keyboard_2);
   auxKeyFunc(AUX_3, keyboard_3);
   auxKeyFunc(AUX_4, keyboard_4);
   auxMainLoop(display);
}
```

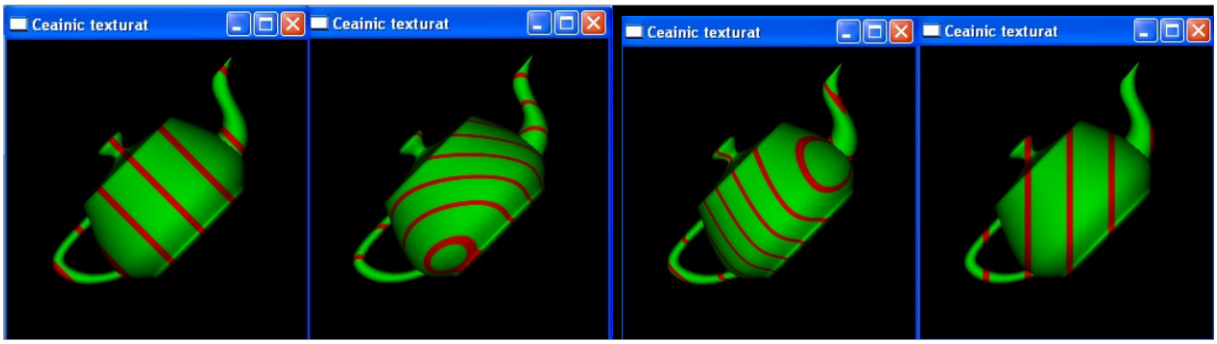

Figura 11 – Texturi în OpenGl

Exerciţii:

Notă:

Se atrage atenţia asupra faptului că toate cunoştinţele şi deprinderile practice căpătate în acest laborator vor fi utilizate şi-n derularea următoarelor laboratoare.

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/opengl/openglstart_9uw5.asp
www.opengl.org