Texturarea in OpenGL (continuare)

3. Filtrarea texturilor

Tipul de filtrare care se aplică unei texturi este definit prin valorile a doi parametri, **GL_TEXTURE_MAG_FILTER** şi **GL_TEXTURE_MIN_FILTER**, setaţi prin apelul funcţiei **glTexParameter#()**. Filtrarea de mărire (magnification, **GL_TEXTURE_MAG_FILTER**) se aplică atunci când dimensiunea pre-imaginii pixelului este egală sau mai mică decât dimensiunea texelului. Filtrarea de micşorare (minification, **GL_TEXTURE_MIN_FILTER**) se aplică atunci când dimensiunea pre-imaginii pixelului este mai mare decât dimensiunea texelului.

Dacă nu este definită imagine mip-map a texturii, cei doi parametri de filtrare **GL_TEXTURE_MAG_FILTER** şi **GL_TEXTURE_MIN_FILTER** pot lua una din valorile **GL_NEAREST** sau **GL_LINEAR**. Valoarea **GL_NEAREST** înseamnă, de fapt, lipsa filtrării: se selectează texelul cel mai apropiat de centrul pre-imaginii. Valoarea **GL_LINEAR** asigură filtrarea texturii prin calculul mediei ponderate a patru texeli cei mai apropiaţi de centrul pre-imaginii pixelului. Dacă este implementată software, filtrarea texturii (**GL_LINEAR**) este executată mai lent decât eşantionare acesteia (**GL_NEAREST**).

Filtrarea de micşorare este accelerată prin tehnica de prefiltrare a texturilor folosind imagini de texturi mip-map. Secvenţa de imaguini ale texturii mip-map poate fi generată prin apelul funcţiei **glTexImage2D()** pentru fiecare nivel, începând cu nivelul 0 (rezoluţie maximă) până la ultimul nivel, cu rezoluţie 1x1. Datele fiecărei imagini (tablou bidimensional de texeli) se obţin din imaginea precedentă prin înlocuirea fiecărui grup de patru texeli cu un texel a cărui culoare este media culorilor celor patru texeli din imaginea precedentă.

Dacă s-a definit imaginea mip-map a texturii, atunci filtrarea de micşorare se poate realiza în mai multe moduri, depinzând de felul în care se selectează nivelul mip-map şi de tipul de filtrare în imaginea mip-map selectată. Filtrul de micşorare (**GL_TEXTURE_MIN_FILTER**) este definit printr-una din următoarele constate:

- **GL_NEAREST**: Nu se aplică nici o filtrare, se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului din imaginea de nivel 0 a texturii mip-map;
- **GL_LINEAR**: se selectează imaginea de nivel 0 a texturii mip-map şi se mediază ponderat patru texeli cei mai apropiaţi de centrul pre-imaginii pixelului;
- **GL_NEAREST_MIPMAP_NEAREST**: se selectează nivelul imaginii mip-map a texturii pentru care dimensiunea pre-imaginii pixelului este cea mai apropiată de dimensiunea texelului şi în această imagine se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului;
- **GL_LINEAR_MIPMAP_NEAREST**: se selectează nivelul imaginii mip-map a texturii pentru care dimensiunea pre-imaginii pixelului are dimensiunea texelului şi în această imagine se mediază ponderat patru texeli cei mai apropiaţi de centrul pre-imaginii pixelului;

- **GL_NEAREST_MIPMAP_LINEAR**: se selectează două imagini mip-map în care pre-imaginea pixelului are dimensiunea cea mai apropiată de imaginea texelului; în fiecare din aceste imagini se selectează câte un texel după criteriul **GL_NEAREST** (texelul cel mai apropiat de centrul pre-imaginii pixelului); aceşti texeli se mediază ponderat pentru obținerea valorii finale a culorii pixelului;
- **GL_LINEAR_MIPMAP_LINEAR**: se selectează două imagini mip-map în care pre-imaginea pixelului are dimensiunea cea mai apropiată de imaginea texelului; în fiecare din aceste imagini se selectează câte un texel după criteriul **GL_LINEAR** (media ponderală a patru texeli cei mai apropiați de centrul pre-imaginii pixelului) şi aceste valori se mediază ponderat pentru obținerea valorii finale a culorii pixelului.

Filtrarea **GL_LINEAR_MIPMAP_LINEAR**, care se mai numeşte şi filtrare triliniară, este cel mai eficient mod de eliminare a zgomotului de antialising al texturii, dar şi cel mai costisitor din punct de vedere al puterii de calcul. Chiar din simpla enumerare a operațiilor efectuate, se poate observa cantitatea de calcule extrem de mare necesară pentru fiecare pixel al imaginii generate. Filtrarea triliniară a texturilor generează imagini deosibit de realiste, dar este fie lentă, dacă este executată software, fie costisitoare, dacă este implementată hardware.

Aplicația 1

Să se citească dintr-un fişier o imagine BMP sau RGB. Imaginea texturii se citeşte dintr-un fişier, folosind funcția **auxRGBImageLoad()**, respectiv **auxDIBImageLoad,** care aparține unei biblioteci auxiliare din OpenGL, **glaux.lib**. Textura se va aplica pe un patrulater plan de coordonate: (0, 0, 1), (3, 0, 1), (3, 3, -1), (0, 3, -1). Se vor afişa imagini fără prefiltrare şi cu prefiltrare, utilizând tehnica mip-map. Prefiltrarea prin tehnica mip-map se poate efectua prin funcția **gluBuild2Dmipmaps().** Setarea tipului de filtrare se face în funcția **Init()** prin funcția **glTexParameterf()**.

Dacă se doreşte să nu se aplice nici o filtrare de micşorare, acest lucru se realizează prin apelul funcției:

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

Dacă se doreşte să se aplice modul de filtrare triliniară se va apela funcția:

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                                        GL_LINEAR_MIPMAP_LINEAR);
```

O parte a programului este dată mai jos, urmând să se facă completările respective.

```
AUX_RGBImageRec *image;

void Plane()
{
      glBegin(GL_QUADS);
      .....

      glEnd();
}
```

```
void init(void)
{
      glClearColor (...);

      //texturare
      image=auxRGBImageLoad("floor.rgb");
//    image=auxDIBImageLoad("tim.bmp");   //tim.bmp - pentru BMP
      glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
      gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image->sizeX,
            image->sizeY,GL_RGB, GL_UNSIGNED_BYTE, image->data);
      glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
             GL_LINEAR);
      glTexParameteri(...);

      glEnable(GL_TEXTURE_2D);
}

void display(void)
{
      ...
      glPushMatrix();
            glTranslatef(-1.5, -1.5, -5);
            Plane();
      glPopMatrix();
      ...
}

void reshape(int w, int h)
{
      glViewport(0, 0, (GLsizei) w, (GLsizei) h);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 0.1, 1000.0);
      glMatrixMode(GL_TEXTURE);
      glLoadIdentity();
      glScaled(8,8,8);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();

}
```

În funcția Reshape() se încarcă matricea din vârful stivei matricelor de texturare cu o matrice de scalare care multiplică coordonatele de texturare cu factorul 8. În acest program se foloseşte o singură textură, care este în permanență textura curentă. Toate funcțiile de definire a parametrilor şi de aplicare a texturii se referă la această unică textură şi nu a fost necesar atribuirea unui nume şi conectarea (bind) texturii pentru definire şi activare.
Eficacitatea tehnicii mip-map se poate vedea realizând afişarea mai multor imagini, fiecare situate la distanță dublă de precedenta. Se va observa că se păstrează rezoluția.

4. Generarea Coordonatelor de Texturare

În general, coordonatele de texturare se generează la crearea modelelor obiectelor tridimensionale. Pentru obiectele modelate prin fețe poligonale, fiecare vârf se defineşte prin coordonate, normală şi coordonate de texturare. Aceste date sunt transmise ca date ale primitivelor geometrice (în blocul **`glBegin()-glEnd()`**) şi prelucrate corespunzător.

Funcțiile de generare a coordonatelor de texturare sunt **glTexGen#()** cu mai multe variante în funcție de tipul argumentelor:

```
void glTexGend(GLenum coord, GLenum pname, GLdouble param);
void glTexGenf(GLenum coord, GLenum pname, GLfloat param);
void glTexGeni(GLenum coord, GLenum pname, GLint param);
```

Primul parametru, coord, specifică coordonate de texturare care se generează şi poate lua una din valorile constante: **`GL_S, GL_T, GL_R`** sau **`GL_Q`**, pentru coordonatele **`s, t, r q`** respectiv. Parametrul pname este numele simbolic (**`GL_TEXTURE_GEN_MODE`**) al funcției de generare a coordonatelor de texturare, iar parametrul **`param`** poate lua una din valorile simbolice **`GL_OBJECT_LINEAR, GL_EYE_LINEAR, GL_SPHERE_MAP.`**

Pentru parametrii **`GL_OBJECT_LINEAR`** şi GL_EYE_LINEAR funcția de generare a coordonatelor de texturare realizează o aplicație a texturii în două etape folosind ca suprafață intermediară un plan. În modul **`GL_OBJECT_LINEAR`** planul este specificat în sistemul de referință de modelare şi se obține o textură fixă pe obiect. În modul **`GL_EYE_LINEAR`** planul intermediar este specificat în sistemul de referință de observare şi se poate obține o textură dinamică care se mişcă pe obiect.

Parametrul **`GL_SPHERE_MAP`** defineşte o funcție de generare a coordonatelor de texturare printr-o aplicație în două etape folosind ca suprafață intermediară o suprafață sferică. Aplicația sferică a texturii dă posibilitatea de a reprezenta obiecte din materiale care simulează materiale reale (de exemplu lemn, marmură etc.)

Aplicația unei coordonate de texturare generate de OpenGL este efectivă numai dacă a fost validată prin apelul uneia dintre funcțiile **`glEnable(GL_TEXTURE_GEN_S)`** sau **`glEnable(GL_TEXTURE_GEN_T)`** pentru coordonata s, respectiv coordonata t.

Aplicația 2

Să se textureze suprafața unui ceainic şi a unui cub utilizând o aplicație sferică. Imaginea texturii se va citi dintr-un fişier (BMP sau RGB). Centrul ceainicului va fi plasat în punctul de coordonate (-2, 0, -8), rotit cu 30 grade față de axa x şi cu 20 față de y. Centrul cubului va fi plasat în punctul de coordonate (2, 0, -8) şi va fi rotit în acelaşi mod ca şi ceainicul. Pentru ceainic se validează generarea automată a coordonatelor de texturare, folosind aplicația sferică (**`GL_SPHERE_MAP`**). Pentru cub coordonatele de texturare se transmit prin

comenzile **glTexCoord2f()** şi este invalidată generarea automată a coordonatelor de texturare (**glDisable(GL_TEXTURE_GEN_S), glDisable(GL_TEXTURE_GEN_T)**).
Programul foloseşte o singură textură mip-map cu filtrare de mărire (**GL_LINEAR**) şi de micşorare (**GL_LINEAR_MIPMAP_LINEAR**).

Texturarea prin aplicație sferică creează efecte deosebit de interesante. Ea este folosită frecvent în ceea ce se numeşte „aplicație a mediului" prin care se redă un obiect perfect reflectiv, în care se reflectă toate obiectele din mediul înconjurător.

Unele părți ale programului sunt date mai jos.

```
AUX_RGBImageRec *image;

void Cube()
{
      //fata
      glBegin(GL_QUADS);
            glTexCoord2f(0, 0);     glVertex3f(-1, -1, 1);
            glTexCoord2f(1, 0);     glVertex3f(1, -1, 1);
            glTexCoord2f(1, 1);     glVertex3f(1, 1, 1);
            glTexCoord2f(0, 1);     glVertex3f(-1, 1, 1);
      glEnd();
      //spate
      ...
      //dreapta
      ...
      //stanga
      ...
      //jos
      ...
      //sus
      ...
}

void init(void)
{
      glClearColor (1, 1, 1, 1.0);
      glEnable(GL_DEPTH_TEST);

      //texturare
      ...
      glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
      gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image->sizeX, image->sizeY,
            GL_RGB, GL_UNSIGNED_BYTE, image->data);
      glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
      glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
      ...
      glEnable(GL_TEXTURE_2D);
}

void display(void)
{
```

```
	glClear(...);
	//texturarea ceainicului
	glEnable(GL_TEXTURE_GEN_S);
	glEnable(GL_TEXTURE_GEN_T);
	glPushMatrix();
	/* Ceainicul se va plasa în punctul de coordonate (-2, 0, -8),
rotit cu  30 grade după axa Ox şi 20 grade după axa Oy, de dimensiune 1.5
.*/
	glPopMatrix();
	glDisable(GL_TEXTURE_GEN_S);
	glDisable(GL_TEXTURE_GEN_T);
	//texturarea cubului
	/* Cubul se va plasa în punctul de coordonate (2, 0, -8), rotit cu
30 grade după axa Ox şi 20 grade după axa Oy.*/
...
}

void reshape(int w, int h)
{
	glViewport(0, 0, (GLsizei) w, (GLsizei) h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 0.1, 100);
	glMatrixMode(GL_TEXTURE);
	glLoadIdentity();
	glScaled(2,2,2);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();

}
```

Utilizând transformările de texturare să se realizeze texturarea diferită a celor doua obiecte. De exemplu, pe ceainic să se multiplice textura de 2 ori şi pe cub de 3 ori.

5. Aplicarea unei imagini true-color pe suprafața unui obiect utilizând o funcție proprie de citire a imaginii din fişier

Textura poate fi o imagine true-color preluată dintr-un fişier BitMap. Formatul BMP a fost dezvoltat de firma Microsoft pentru a stoca imagini raster într-un mod independent de dispozitivul de afişare. Formatul are variante atât pentru codificarea imaginilor monocromatice cât şi a celor color în care culorile pixelilor pot fi specificate prin triplete RGB sau prin indecşi într-o tabelă de culori.

Într-un fişier BMP informația este organizată sub forma unui antet (care descrie modul de structurare a informației într-un fişier) şi a unei zone de date. Codurile de culoare corespunzătoare pixelilor unei imagini sunt stocate în zona de date pe linii, de la stânga la dreapta. Originea unei imagini BitMap este plasată în colțul stânga-jos. Biții asociați fiecărui pixel sunt împachetați în octeți. Formatul BMP are mai multe variante, după cum informația referitoare la culoarea pixelilor este sau nu comprimată. Funcția prezentată şi

care va fi folosită în aplicația 3 parcurge un fișier în care informația nu este comprimată. O altă variantă a formatului foloseşte pentru comprimarea informației de culoare un algoritm de tip RLE (Run Length Encoding). În principiu, această schemă de compresie identifică în secvența de date şiruri rezultate prin repetiția unui acelaşi octet. În continuare fiecare apariție a unui astfel de şir va fi înlocuită cu valoarea octetului (care se repetă) şi o valoare ce indică numărul de repetiții.
Rutina **`ReadBMP()`** interpretează conținutul unui fişier BMP (ce codifică o imagine true-color) cu numele specificat în primul parametru. Prototipul funcției este:

```
PIXEL *ReadBMP(Char *nume, int *w, int *h);
```

În cazul în care datele din fişier respectă sintaxa formatului BMP funcția va întoarce în ultimii doi parametri numărul de linii respectiv numărul de coloane ale imaginii. Rezultatul întors este adresa de start a unui tablou ce conține codurile de culoare ai pixelilor ce alcătuiesc imaginea.

```
typedef struct{GLubyte r, g, b;} PIXEL;

PIXEL *imagineBMP;
GLint width, hight;

/*    citirea unei imagini tru-color dintr-un fisier cu numele specificat
      ca prim parametru al functiei.
      - nlin, ncol = pointer pentru numarul de linii respectiv de coloane
ale imaginii
      - img = tablou ce contine codurile de culoare ale pixelilor */

PIXEL *ReadBMP(char *nume, int *ncol, int *nlin)
{
      int i, j, w, h, data_offset, r, lglin, tcompr;
      long dofs;
      short nbpp;
      char buf[20];
      PIXEL *img;
      GLubyte t;
      FILE *f;
      if((f=fopen(nume, "rb"))==NULL)
      {
            fprintf(stderr, "ReadBMP --eroare deschidere fisier\n");
            exit(1);
      }
      fread(buf, 1,2, f);
      buf[2]='\0';
      fseek(f, 28L, SEEK_SET);
      fread(&nbpp, sizeof(short), 1, f);
      fseek(f, 30L, SEEK_SET);
      fread(&tcompr, sizeof(int), 1, f);
      if(strcmp(buf, "BM") !=0 || nbpp != 24 || tcompr != 0)
      {
            //se citeste un fisier BMP necomprimat
            fprintf(stderr, "ReadBMP -- format BMP incorect\n");
            exit(1);
      }
      fseek(f, 18L, SEEK_SET);
```

```
      fread(&w, sizeof(int), 1, f);             //citesc    numarul    de
coloane
      fread(&h, sizeof(int), 1, f);             //citesc numarul de linii
      img = (PIXEL *)malloc(h*w*sizeof(PIXEL));
      fseek(f, 10L, SEEK_SET);      //cauta deplasamentul zonei de date
      fread(&data_offset, sizeof(int), 1, f);   //citesc deplas. zonei de
                                    // date fata de inceputul fisierului
      lglin=w*3;
      if(r = lglin%4) lglin += 4 - r;
      for(dofs=data_offset, i=0;  i<h;  dofs += lglin, i++)
      {
            fseek(f, dofs, SEEK_SET);
            fread(&img[i*w], sizeof(PIXEL), w, f);
            for(j=0;  j<w; j++)                         //schimba red/blue
            {
                  t = img[i*w+j].r;
                  img[i*w+j].r = img[i*w+j].b;
                  img[i*w+j].b = t;
            }
      }
      *nlin = h;
      *ncol = w;
      fclose(f);
      return img;
}
```

În cazul unei imagini true-color, fiecărui pixel şi sunt asociaţi 3 octeţi de informaţie, câte unul pentru fiecare din cele trei culori fundamentale. Codificarea unei linii a imaginii trebuie să aibă un număr de octeţi multiplu de 4. În fişierul BitMap octeţii asociaţi unui pixel al imaginii conţin (în ordine) codificarea intensităţilor de albastru, verde respectiv roşu care intră în componenţa culorii pixelului. Pe de altă parte, formatul GL_RGB de codificare internă a nei imagini OpenGl consideră codurile de intensitate asociate componentelor roşie,verde şi albastră ale culorii unui pixel ca fiind stocate în memorie exact în ordinea R, G, B. De aceea, rutina ReadBMPTC() interschimbă valorile roşu şi albastru ce intră în componenţa culorii fiecărui pixel,
Afişarea imaginii citite poate fi realizată folosind secvenţa:

```
glRasterPos2i(0,0);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glDrawPixels(wbal, hbal, GL_UNSIGNED_BYTE, Img_bal);
```

În următorul tablel este prezentată parţial structura unui fişier BMP, în cazul codificării unei imagini true-color, necomprimate:

Deplasament faţă de început fişier	Lungime câmp (octeţi)	Conţinut câmp
0	2	„BM”
2	4	Număr de cuvinte de 32 biţi ce compun fişierul

6	4	Rezervat. (valoarea 0)
10	4	Deplasament față de începutul fişierului al începutului zonei de date
14	4	Număr de octeți header
18	4	Număr de coloane imagine
22	4	Număr de linii imagine
26	2	Valoare 1
28	2	Valoare 24 (în cazul imaginilor true-color)
30	4	Valoare 0 în cazul imaginilor necomprimate
...		Informații specifice imaginilor codificate în mod index (descriere paletă culori)
...		Zona de date

Aplicația 3

Să se citească o imagine true-color dintr-un fişier cu numele specificat, folosind rutina descrisă mai sus. Imaginea respectivă va constitui textura ce se va aplica pe o sferă. Se va utiliza generarea automată a coordonatelor de texturare. Cu butonul stâng al mouse-ului se va activa rotirea sferei în jurul axei z (cu un unghi de rotație de 1 grad), iar cu butonul drept se va opri rotația. În program se vor utiliza transformări de texturare pentru aplicarea texturii în locuri diferite pe sferă, rotită şi multiplicată de 2 şi 3 ori.

```
static GLuint texName;

void InitTextures(void)
{
      glGenTextures(1, &texName);
      glBindTexture(GL_TEXTURE_2D, texName);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
      glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width,
            hight, 0, GL_RGB, GL_UNSIGNED_BYTE, imagineBMP);
      glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
      glEnable(GL_TEXTURE_2D);
      glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
      glEnable(GL_TEXTURE_GEN_S);

      glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
      glEnable(GL_TEXTURE_GEN_T);
      glShadeModel(GL_SMOOTH);

}
```

```
void init(void)
{
      glClearColor (0.0, 0.0, 0.0, 0.0);
      InitTextures();
      glEnable(GL_DEPTH_TEST);
}

static int n=0;

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

            glRotatef(n, 0, 0, 1);
            glutSolidSphere(1, 30, 30);
            if(n>=360)  n=0;
                        else        n=n+1;
      glPopMatrix();
      glutSwapBuffers();
      glFlush();
}

void reshape(int w, int h)
{

      glViewport(0, 0, (GLsizei)w, (GLsizei)h);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(60, (GLdouble)w/(GLdouble)h, 1, 10);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glTranslated(0,0,-4);

}
...


int main(int argc, char** argv)
{
      imagineBMP=ReadBMP("glass.bmp", &width, &hight);
      glutInit(&argc, argv);
...}
```