LABORATOR 9

Texturi

O textura este o imagine aplicata pe un polygon. Prin atasarea vertexurilor (in spariul ecranului) la puncte de control numite **coordonate texturale u** si **v** (in spatiul texturilor), obtinem o interpolare pe ecran. Texturile sunt in mod obisnuit folosit pentru a adauga o infatisare unor suprafete ce ar fi dificil de produs daca s-ar folosi poligoane.

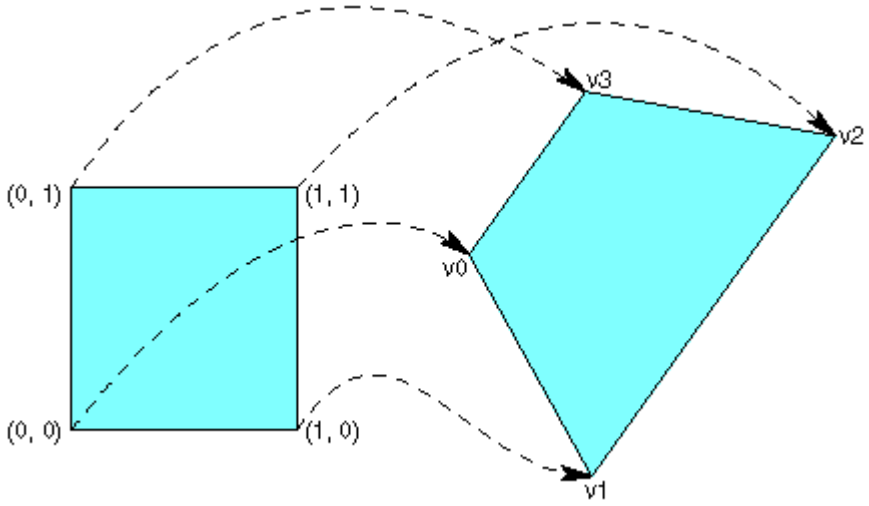


Fig. 1. Transformarile formeaza spatii texturale in spatial ecranului.

Exista un lucru foarte important de luat in seama cand se folosesc texturile in OpenGL: pentru o performanta cat mai buna, texturile trebuie sa fie imagini patrate si sa aiba dimensiunea putere de 2. Deci, dimenisiunile bune folosite pentru texturi sunt: 2, 4, 8, 16, 32, 64, 128, 256, 512, …

Numele texturilor

Pentru a incepe, ne trebuie in primul rand un nume pentru textura noastra. Acesta este in general un numar pe care OpenGL-ul il foloseste sa indexeze toate texturile diferite. Pentru a denumi o textura trebuie sa apelam functia *glGenTextures*.

```
#define NR_MAX_TEXTURI 2

GLuint texture [NR_MAX_TEXTURI];

glGenTextures(2,texture);/* aici se genereaza doua obiecte textura */
```

Acum ca am denumit textura, putem sa trecem de la o textura la alta folosind functia *glBindTexture*. De fapt aceasta allege cu ce textura se lucreaza. In acest moment, ar trebui sa observi ca sunt doua tipuri de texturi in OpenGL, 1D si 2D. Foarte probabil nu vei folosi 1D, asa ca vom prezenta texturile 2D. Deci hai sa selectam numele texturii pe care tocmai am creat-o.

// allege textura curenta

glBindTexture(GL_TEXTURE_2D, texture[0]);Texture Parameters

Daca vrem sa folosim o textura diferita va trebui sa folosim: **texture[1], texture[2], etc.** In exemplul nostrum, alegem doua texturi (**NR_MAX_TEXTURI 2**): numai **texture[a]** si **texture[1]** sunt disponibile aici.

Acum putem incepe lucrulla textura curenta. In primul rand, trebuie sa dam parametrii texturii. In al doilea rand, trebuie sa incarcam datele texturii. Inainte sa incepem, ar trebui sa declaram situatia cadrului textural. Cadrul exact pe care il vei decalra poate sa fie diferit de cel prezentat aici. *GL_MODULATE* preia culoarea si datele din textura si le multiplica cu datele din *glColor* si/sau sistemul de iluminare.

// alegeti *modulate* pentru a amesteca textura cu culoarea pentru umbre/nuante

glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

In continuare avem patru parametrii texturali pe care trebuie sa ii declaram. Aici putem crea efecte precum filtrarea texturala biliniara sau triliniara, dar si mipmapping-ul. Deasemenea putem spune daca textura se impatureste/indoaie la capete sau este dreapta. Repet pentru ca este cea mai obisnuita folosire. Cititi comentariile prntru detalii astura modului in care lucreaza fiecare.

// cand zona de texturare este mica, filtrul biliniar cel mai apropiat mipmap

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_NEAREST);

// cand zona de texturare este mare, filtrul bilinear originalul glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

// textura se indoaie/impatureste la capete (repeat) glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

Functia *glTexParameter()* este o parte extreme de importanta a texturarii in OpenGL, aceasta functie determina comportamentul si modul de afisare al texturilor cand sunt redate. In cele ce urmeaza avem o sumedenie de parametric si efectul lor asupra texturii redate. Fiecare textura poate avea propriile proprietati, care nu sunt globale. Proprietatile unei texturi nu vor afecta alte texturi.

Declarare	**Specifica ce texturi vizeaza**
GL_TEXTURE_1D	Texturi uni-dimensionale.
GL_TEXTURE_2D	Texturi bi-dimensionale.

Parametrul texturii	**Valorile acceptate**	**Descrierea**
GL_TEXTURE_MIN_FILTER	GL_NEAREST , GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR and GL_LINEAR_MIPMAP_LINEAR	Acesata textura este folosita cand un singur pixel al ecranului este atribuit mai multor texeli, asta inseamna ca textura va fi micsorata. Declararea standard este: GL_NEAREST_MIPMAP_LINEAR.
GL_TEXTURE_MAG_FILTER	GL_NEAREST or GL_LINEAR	Functia de marire a texturii este folosita cand pixelul este atribuit unei zone egale sau mai mici decat un texel, asta inseamna ca textura va fi marita. Declararea standard este: GL_LINEAR.

GL_TEXTURE_WRAP_S	GL_CLAMP or GL_REPEAT	Seteaza parametrii de suprafata pentru coordonatele texturii. Poate fi declarat GL_CLAMP sau GL_REPEAT. Declararea standard este: GL_REPEAT.
GL_TEXTURE_WRAP_T	GL_CLAMP or GL_REPEAT	Seteaza parametrii de suprafata pentru coordonatele texturii. Poate fi declarat GL_CLAMP sau GL_REPEAT. Declararea standard este: GL_REPEAT.
GL_TEXTURE_BORDER_COLOR	Any four values in the [0, 1] range	Seteaza culoarea marginii texturii, daca exista o margine. Declararea standard este: (0, 0, 0, 0).
GL_TEXTURE_PRIORITY	[0, 1]	Specifica prioritatea de baza a texturii, folosita sa impiedice OpenGL-ul sa scoata texturile din memoria video. Paoate avea valori in [0, 1] . Vezi *glPrioritizeTextures()* pentru mai multe informatii.

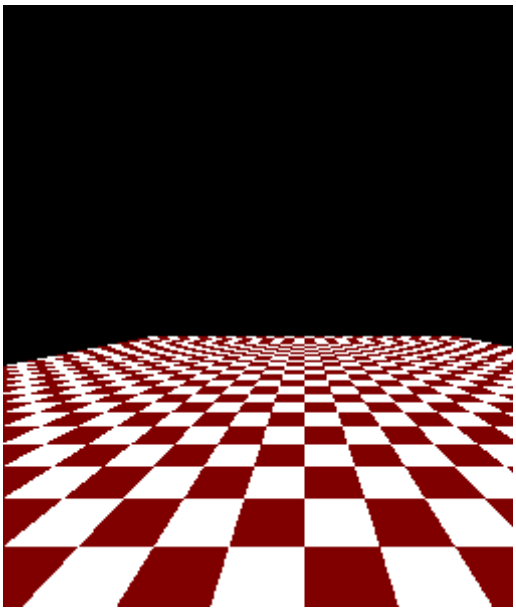
Parametrul	**Descrierea**
GL_CLAMP	Prinde coordonatele texturii on valorile [0,1].
GL_REPEAT	Ignora partea intreaga a coordonatelor texturii, numai partea fractionara este

	folosita, ceea ce creaza un sablon repetitive. O coordonata a texturii de 3.0 va face textura sa se stranga de trei ori cand se va fi redata.
GL_NEAREST	Intoarce valoarea elemenului textural care este cel mai apropiat de centrul pixelului texturat. Folositi acest parametru daca doriti sa aveti textura aspra cand este redata.
GL_LINEAR	Intoarce media celor patru elemente de textura care sunt mai apropiate de centrul pixelului texturat. ACeasta comanda poate include elemente texturale cu margine, depinzand de valorile *GL_TEXTURE_WRAP_S* si *GL_TEXTURE_WRAP_T*, si de mapping. Folositi acest parametru daca ati vrea textura sa fie in ceata.
GL_NEAREST_MIPMAP_NEAREST	Alege mipmap-ul care se potriveste cel mai bine marimii pixelului texturat si foloseste formularea *GL_NEAREST* (elemental textural cel mai apropiat de centrul pixelului).
GL_LINEAR_MIPMAP_NEAREST	Alege mipmap-ul care se potriveste cel mai bine marimii pixelului texturat si foloseste formularea *GL_LINEAR* (o medie a celor patru elemente care sunt cel mai apropiate de centrul pixelului).
GL_NEAREST_MIPMAP_LINEAR	Alege cele doua mipmap-uri care se potrivesc cel mai bine marimii pixelului texturat si foloseste formularea *GL_NEAREST* (elemental textural cel mai apropiat de centrul pixelului). Textura finala este media celor doua

	valori.
GL_LINEAR_MIPMAP_LINEAR	Alege cele doua mipmap-uri care se potrivesc cel mai bine marimii pixelului texturat si foloseste formularea *GL_LINEAR* o medie a celor patru elemente care sunt cel mai apropiate de centrul pixelului). Textura finala este media celor doua valori.

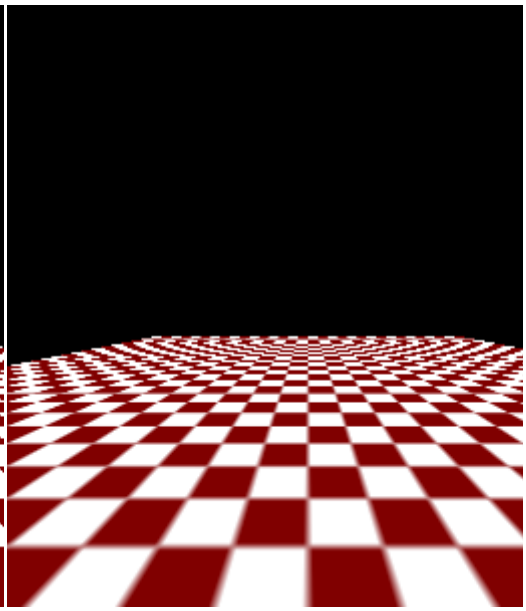
Exemple:

Min / Max Filter : GL_NEAREST

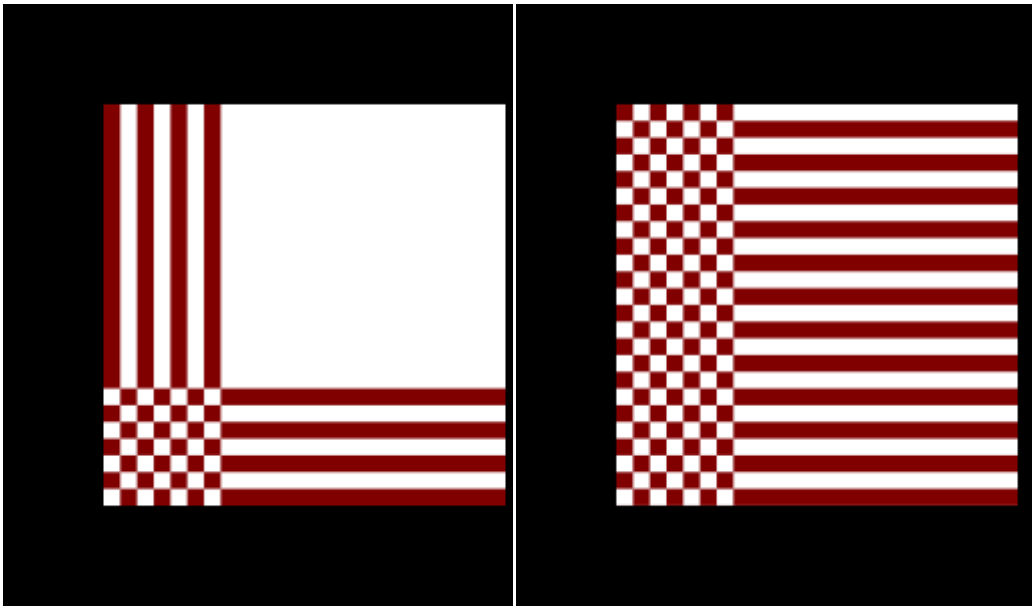


Wrap S : GL_CLAMP
Wrap T : GL_CLAMP

Min / Max Filter : GL_LINEAR

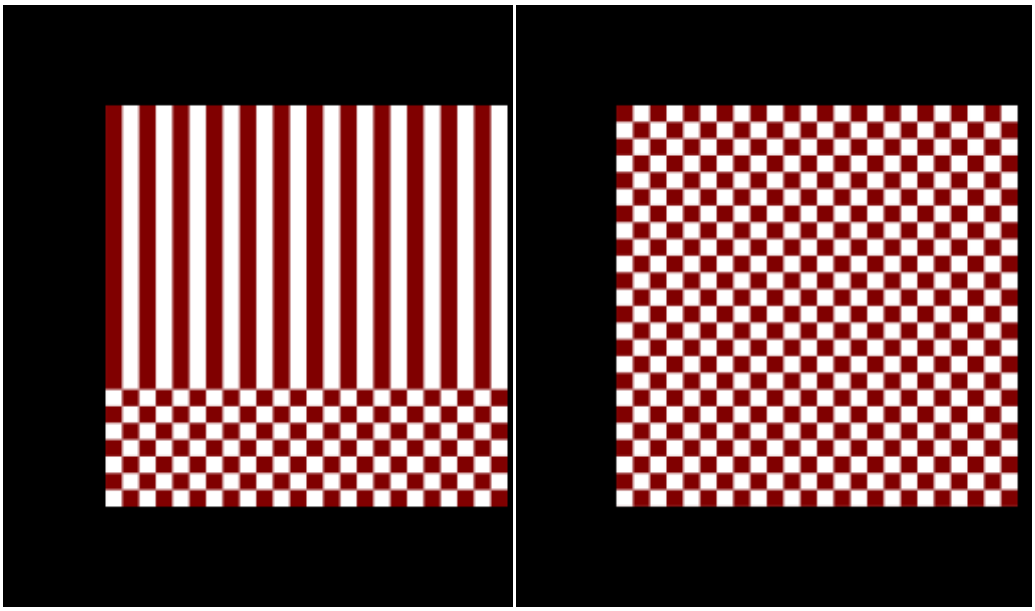


Wrap S : GL_CLAMP
Wrap T : GL_REPEAT



Wrap S : GL_REPEAT
Wrap T : GL_CLAMP

Wrap S : GL_REPEAT
Wrap T : GL_REPEAT



Texturarea imaginilor si mipmapurile

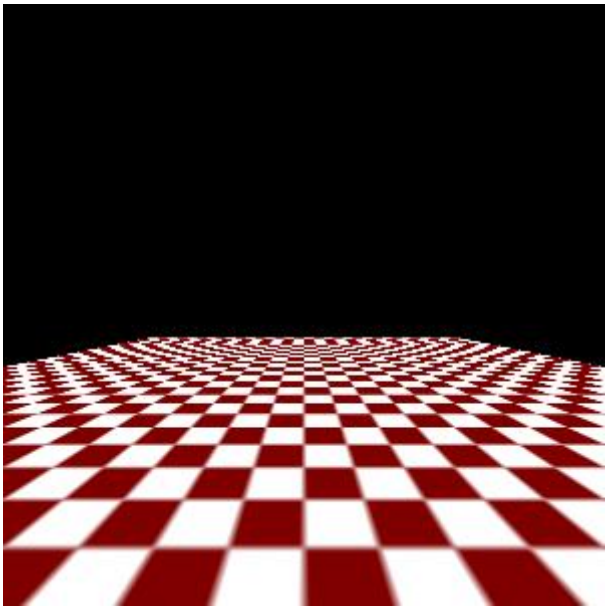
Cand o scena este redata in openGL obiectele texturate sunt desenate la distante variate din punctual de unde se priveste, unele apropiate iar unele departate. Cu cat obiectul texturat este mai departat de privitor, textura trebuie micsorata ca sa poate fi aplicata unui obiect mai mic. Problema care apare prin micsorarea texturii este ca pe masura ceobiectele se departeaza textura

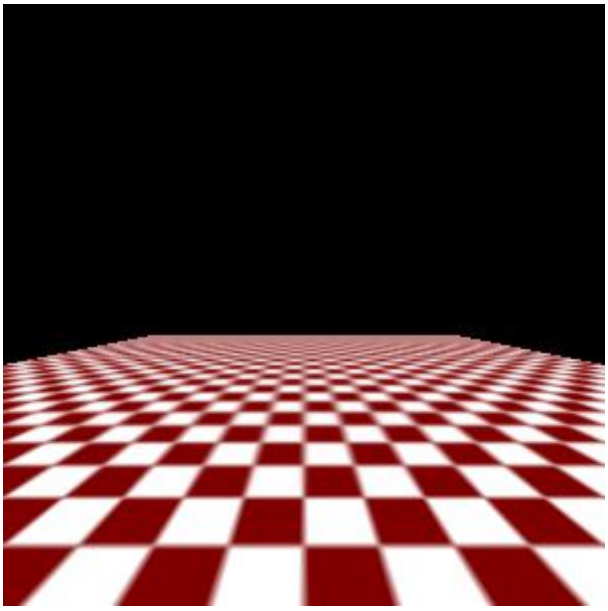
poate sa licare din cauza filtrarii. Solutia acestei probleme enervante este mipmapping-ul. Mipmapping-ul este procesul prin care un set de texturi filtrate de rezolutie descrescatoare sunt generate dintr-o singura textura de rezolutie mare si folosita sa imbunatateasca precizia in timpul texturarii. Fiecare mipmap este o textura distincta, arata ca originalul, dar este micsorata si filtrata pana la nivelul anterior al mipmap-ului. Mipmapping-ul permite OpenGL-ului sa aplice un nivel al detaliului mare la texturi cand redarea se face prin micsorare.

Generarea si folosirea mipmap-urilor

Fiecare nivel al mipmap-ului este generat prin luarea unei jumatati din marimea anterioara a mipmapu-ului si prin aplicarea unui filtru. Daca avem o imagine de 64x64, nivelurile inferioare ale detaluilui vor fi 32x32, 16x16, 8x8, 4x4, 2x2 si 1x1. Exista doua moduri de a calcula mipmap-ul si sa le folosim in OpenGL, ñde manaò sau prin functia *gluBuild2DMipmaps()*. Daca mipmap-urile sunt generate prin algoritmul creat de fiecare, functia *glTexImage2D()* Va fi folosita prin marirea fiecarul nivel al parametrilor pentru fiecare nivel succesiv al detalilui, cu nivelul 0 fiind la baza nivelelor de texturare. Destul de des, functia utilitare *gluBuild2DMipmaps()* va fi folosita pentru generarea unui set de mipmap-uri folosite la o textura data. *gluBuild2DMipmaps()* va avea grija de filtrarea si uploadarea momoriei mipmap-urilor. Functia utilitara *gluBuild2DMipmaps()* este aceeasi cu *glTexImage2D(),* cu exceptia faptului ca ea contruieste mipmap-uri.

Dupa ce am generat nivelurile diferite, cum pot sa fie folosite sa imbunatateasca calitatea obiectului texturat? Duca ce mipmap-urile folosite au fost calculate pentru acea textura si uploadate in memorie fara vre-un cod special, va trebui sa le folosim. OpenGL va alege in mod automat cel mai bun nivel al detaliului fara interventia programatorului. Mipmap-ul ales poate fi definit cu functia *glTexParameter()*. Dezavantajul mipmap-urilor este ca vor necesita mai multa memorie de texturare si va fi necesara o porgramare in plus pentru a le genera.





Cu mipmapping Fara mipmapping

Observati cum textura este deformata la distanta fara mipmap-uri, dar cu acestea este neteda si finisata si nu devine deformata la distanta.

Din moment ce am programat filtrul minim sa aleaga un mipmap, vom crea mipmap-uri chiar noi. Din fericire exsista deja o functie care face acest lucru automat. Tot ce trebuie sa facem este sa plasam informatiile imaginilor reale si functia face toata treaba in locul nostru.

Activarea si aplicarea texturilor

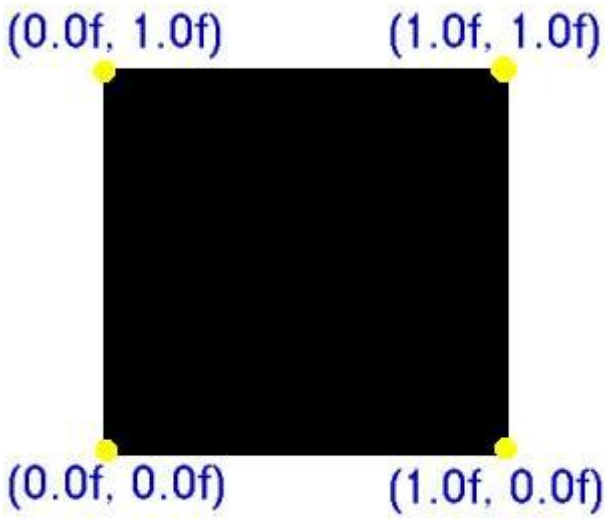
Dup ace am aflat cum se incarca o textura, pornim si oprim mapping-ul, apelam functia glEnable sau glDisable. Acest lucru il facem prin parametrul GL_TEXTURE_2D. Simplu nu?

```
glEnable( GL_TEXTURE_2D );
```

Se pot incarca diferite texturi si pot fi apoi selectate pe rand prin functia glBindTexture exact ca in LoadTextureRAW.

```
glBindTexture( GL_TEXTURE_2D, texture );
```

Ultima parte a texturarii este cea a coodronatelor. Desi esti obisuit sa la spui U si V, in OpenGL ele sunt denumite S si T. Pentru texturi, S si T si intre 0 si 1. Pentru texturile repetitive, marginile sunt 0 si 1 pentru fiecare textura (1 si 2, 2 si 3, sau chiar -1 si 0). Coordonatele se atribuie exact cum se atribuie culoarea dar cu functia glTexCoord. Orice culoare aplicata este multiplicata la numarul de texturi, asa ca in cazul in care textura este intunecata sau are o culoare neobisnuita, verificati culoarea aplicata.




```
glBegin( GL_QUADS );
glTexCoord2d(0.0,0.0); glVertex2d(0.0,0.0);
glTexCoord2d(1.0,0.0); glVertex2d(1.0,0.0);
glTexCoord2d(1.0,1.0); glVertex2d(1.0,1.0);
glTexCoord2d(0.0,1.0); glVertex2d(0.0,1.0);
glEnd();
```

Imaginea de mai sus prezinta sistemul de coordonate ale texturilor in OpenGL. In sectiunea de program de mai sus *glTexCoord2f* este foarte importanta deoarece ea va afecta in mod direct mapping-ul. Prin functia *glTexCoord2f (x,y)* OpenGL plaseaza coordonatele pe imagine. Daca imaginea texturata este de forma unui triunghi, imaginea va avea trei coordonate.

Aplicatia nr.1

```
// Aceasta aplicatie este o ilustrare de cartografiere a texturii.
// Sunt reprezentate mai multe triunghiuri, fiecare cu o cate o textură mapata pe
triunghi .
// Aceeași textură este utilizată pentru fiecare triunghi, dar mapările variaza destul de
mult, astfel se pare ca
// fiecare triunghi are o textură diferită.

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

// Se defineste un model de matrice 2 x 2 in carouri si culori de roșu și galben RGB.
#define red {0xff, 0x00, 0x00}
#define yellow {0xff, 0xff, 0x00}
#define magenta {0xff, 0, 0xff}
GLubyte texture[][3] = {
    red, yellow,
    yellow, red,
};

// Se fixeaza aparatul de fotografiat și aplica textura când fereastra este remodelata.
void reshape(int width, int height) {
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(80, GLfloat(width)/height, 1, 40);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(2, -1, 5, 0, 0, 0, 0, 1, 0);
  glEnable(GL_TEXTURE_2D);
  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
  glTexImage2D(GL_TEXTURE_2D,
               0,                    // nivelul  0
               3,                    // utilizeaza numai componentele R, G, si B
               2, 2,                 // textura are 2x2 texeli
               0,                    // fara cadru
               GL_RGB,               // texeli sunt in format RGB
               GL_UNSIGNED_BYTE,     // componentele de culoare sunt fara octeti asciati
               texture);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
}

// Deseneaza trei triunghiuri texturate. Fiecare triunghi utilizează aceeași textura,
// Dar mapările de textura pe coordonatele vertex sunt
// Diferite în fiecare triunghi.
void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_TRIANGLES);
    glTexCoord2f(0.5, 1.0);    glVertex2f(-3, 3);
    glTexCoord2f(0.0, 0.0);    glVertex2f(-3, 0);
    glTexCoord2f(1.0, 0.0);    glVertex2f(0, 0);

    glTexCoord2f(4, 8);        glVertex2f(3, 3);
    glTexCoord2f(0.0, 0.0);    glVertex2f(0, 0);
```

```
    glTexCoord2f(8, 0.0);      glVertex2f(3, 0);

    glTexCoord2f(5, 5);        glVertex2f(0, 0);
    glTexCoord2f(0.0, 0.0);    glVertex2f(-1.5, -3);
    glTexCoord2f(4, 0.0);      glVertex2f(1.5, -3);
  glEnd();
  glFlush();
}

// Initializeaza GLUT și intră în bucla principală.
int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(520, 390);
  glutCreateWindow("Triunghiuri Texturate");
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
}
```

Aplicatia nr.2

Visual Studio 2010ʒ Projectʒ Folder Creat de proiect se introduce fisierul .bmp

Fisierul care încarcă imaginea (de tip header)

fisierincarcare.h

```
#ifndef IMAGE_LOADER_H_INCLUDED
#define IMAGE_LOADER_H_INCLUDED

//Reprezentarea unei imagini
class Image {
	public:
		Image(char* ps, int w, int h);
		~Image();

		/* O matrice de forma (R1, G1, B1, R2, G2, B2, ...) indicând
* Culoarea fiecarui pixel din imagine. Componente de culoare variază de la 0 la 255.
* Matricea începe de la  pixelul din stânga-jos, apoi se mută până la sfârșitul
* rândului, apoi se mută pe la coloana următoare, și așa mai departe. Acesta este
* Formatul în care OpenGL plaseaza imagini.      */
		char* pixels;
		int width;
		int height;
};

//Citeste imaginea bmp din fisier.
Image* loadBMP(const char* filename);



#endif
```

fisierincarcare.cpp

```
#include <assert.h>
#include <fstream>

#include "imageloader.h"

using namespace std;

Image::Image(char* ps, int w, int h) : pixels(ps), width(w), height(h) {

}

Image::~Image() {
	delete[] pixels;
}

namespace {
	//Convertește un sir de patru caractere la un număr întreg
	int toInt(const char* bytes) {
		return (int)(((unsigned char)bytes[3] << 24) |
					((unsigned char)bytes[2] << 16) |
					((unsigned char)bytes[1] << 8) |
					(unsigned char)bytes[0]);
	}

	//Convertește un sir de patru caractere la un număr întreg
	short toShort(const char* bytes) {
		return (short)(((unsigned char)bytes[1] << 8) |
						(unsigned char)bytes[0]);
	}

	//Citește următorii octeți ca un întreg
	int readInt(ifstream &input) {
		char buffer[4];
		input.read(buffer, 4);
		return toInt(buffer);
	}

	//Citește următorii octeți
	short readShort(ifstream &input) {
		char buffer[2];
		input.read(buffer, 2);
		return toShort(buffer);
	}

	//Just like auto_ptr, but for arrays
	template<class T>
	class auto_array {
		private:
			T* array;
			mutable bool isReleased;
		public:
			explicit auto_array(T* array_ = NULL) :
```

```
                array(array_), isReleased(false) {
        }

        auto_array(const auto_array<T> &aarray) {
                array = aarray.array;
                isReleased = aarray.isReleased;
                aarray.isReleased = true;
        }

        ~auto_array() {
                if (!isReleased && array != NULL) {
                        delete[] array;
                }
        }

        T* get() const {
                return array;
        }

        T &operator*() const {
                return *array;
        }

        void operator=(const auto_array<T> &aarray) {
                if (!isReleased && array != NULL) {
                        delete[] array;
                }
                array = aarray.array;
                isReleased = aarray.isReleased;
                aarray.isReleased = true;
        }

        T* operator->() const {
                return array;
        }

        T* release() {
                isReleased = true;
                return array;
        }

        void reset(T* array_ = NULL) {
                if (!isReleased && array != NULL) {
                        delete[] array;
                }
                array = array_;
        }

        T* operator+(int i) {
                return array + i;
        }

        T &operator[](int i) {
                return array[i];
        }
};
}
```

```
Image* loadBMP(const char* filename) {
	ifstream input;
	input.open(filename, ifstream::binary);
	assert(!input.fail() || !"Nu a gasit fisierul");
	char buffer[2];
	input.read(buffer, 2);
	assert(buffer[0] == 'B' && buffer[1] == 'M' || !"Nu a gasit fisierul bitmap");
	input.ignore(8);
	int dataOffset = readInt(input);

	//Read the header
	int headerSize = readInt(input);
	int width;
	int height;
	switch(headerSize) {
		case 40:
			//V3
			width = readInt(input);
			height = readInt(input);
			input.ignore(2);
			assert(readShort(input) == 24 || !"Imaginea nu este de 24 bits per
pixel");
			assert(readShort(input) == 0 || !"Imagine comprimata");
			break;
		case 12:
			//OS/2 V1
			width = readShort(input);
			height = readShort(input);
			input.ignore(2);
			assert(readShort(input) == 24 || !"Imaginea nu este de 24 bits per
pixel");
			break;
		case 64:
			//OS/2 V2
			assert(!"Nu se poate incarca ");
			break;
		case 108:
			//Windows V4
			assert(!"Nu se poate incarca ");
			break;
		case 124:
			//Windows V5
			assert(!"Can't load Windows V5 bitmaps");
			break;
		default:
			assert(!"Unknown bitmap format");
	}

	//Citeste datele
	int bytesPerRow = ((width * 3 + 3) / 4) * 4 - (width * 3 % 4);
	int size = bytesPerRow * height;
	auto_array<char> pixels(new char[size]);
	input.seekg(dataOffset, ios_base::beg);
	input.read(pixels.get(), size);

	//Obtine datele in formatul corect
	auto_array<char> pixels2(new char[width * height * 3]);
	for(int y = 0; y < height; y++) {
```

```
			for(int x = 0; x < width; x++) {
				for(int c = 0; c < 3; c++) {
					pixels2[3 * (width * y + x) + c] =
						pixels[bytesPerRow * y + 3 * x + (2 - c)];
				}
			}
		}

		input.close();
		return new Image(pixels2.release(), width, height);
}
```

Programul principal

```
#include <iostream>
#include <stdlib.h>

#ifdef __APPLE__
#include <OpenGL/OpenGL.h>
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#include "imageloader.h"

using namespace std;

void handleKeypress(unsigned char key, int x, int y) {
		switch (key) {
			case 27: //Escape key
				exit(0);
		}
}

//Transforma imaginea într-o textură, și returnează ID-ul texturii
GLuint loadTexture(Image* image) {
		GLuint textureId;
		glGenTextures(1, &textureId); //Construieste camera pentru textura
		glBindTexture(GL_TEXTURE_2D, textureId); //Comunica OpenGL ce textura sa editeze
		//Harta imagine pentru textura
		glTexImage2D(GL_TEXTURE_2D,				// GL_TEXTURE_2D
					0,						//0 deocamdata
					GL_RGB,					//Formatul OpenGL utilizat
pentru imagine
					image->width, image->height,  //Latimea si inaltimea
					0,						//Cadrul imaginii
					GL_RGB, //GL_RGB,
					GL_UNSIGNED_BYTE, //GL_UNSIGNED_BYTE,
								//numere fara semn
					image->pixels);				//Datele pentru pixeli
		return textureId; //Returneaza id-ul texturii
}

GLuint _textureId; //id-ul texturii
```

```
void initRendering() {
	glEnable(GL_DEPTH_TEST);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	glEnable(GL_NORMALIZE);
	glEnable(GL_COLOR_MATERIAL);

	Image* image = loadBMP("vtr.bmp");
	_textureId = loadTexture(image);
	delete image;
}

void handleResize(int w, int h) {
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(45.0, (float)w / (float)h, 1.0, 200.0);
}

void drawScene() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

	glTranslatef(0.0f, 1.0f, -6.0f);

	GLfloat ambientLight[] = {0.2f, 0.2f, 0.2f, 1.0f};
	glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientLight);

	GLfloat directedLight[] = {0.7f, 0.7f, 0.7f, 1.0f};
	GLfloat directedLightPos[] = {-10.0f, 15.0f, 20.0f, 0.0f};
	glLightfv(GL_LIGHT0, GL_DIFFUSE, directedLight);
	glLightfv(GL_LIGHT0, GL_POSITION, directedLightPos);

	glEnable(GL_TEXTURE_2D);
	glBindTexture(GL_TEXTURE_2D, _textureId);

	//Bottom
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
	//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	glColor3f(1.0f, 0.2f, 0.2f);
	glBegin(GL_QUADS);

	glNormal3f(0.0, 1.0f, 0.0f);
	glTexCoord2f(0.0f, 0.0f);
	glVertex3f(-2.5f, -2.5f, 2.5f);
	glTexCoord2f(1.0f, 0.0f);
	glVertex3f(2.5f, -2.5f, 2.5f);
	glTexCoord2f(1.0f, 1.0f);
	glVertex3f(2.5f, -2.5f, -2.5f);
	glTexCoord2f(0.0f, 1.0f);
	glVertex3f(-2.5f, -2.5f, -2.5f);

	glEnd();
```

```
	//Jos
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	glColor3f(1.0f, 1.0f, 1.0f);
	glBegin(GL_TRIANGLES);

	glNormal3f(0.0f, 0.0f, 1.0f);
	glTexCoord2f(0.0f, 0.0f);
	glVertex3f(-2.5f, -2.5f, -2.5f);
	glTexCoord2f(5.0f, 5.0f);
	glVertex3f(0.0f, 2.5f, -2.5f);
	glTexCoord2f(10.0f, 0.0f);
	glVertex3f(2.5f, -2.5f, -2.5f);

	glEnd();

	//Stanga
	glDisable(GL_TEXTURE_2D);
	glColor3f(1.0f, 0.7f, 0.3f);
	glBegin(GL_QUADS);

	glNormal3f(1.0f, 0.0f, 0.0f);
	glVertex3f(-2.5f, -2.5f, 2.5f);
	glVertex3f(-2.5f, -2.5f, -2.5f);
	glVertex3f(-2.5f, 2.5f, -2.5f);
	glVertex3f(-2.5f, 2.5f, 2.5f);

	glEnd();

	glutSwapBuffers();
}

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

	glutCreateWindow("Textura");
	initRendering();

	glutDisplayFunc(drawScene);
	glutKeyboardFunc(handleKeypress);
	glutReshapeFunc(handleResize);

	glutMainLoop();
	return 0;
}
```