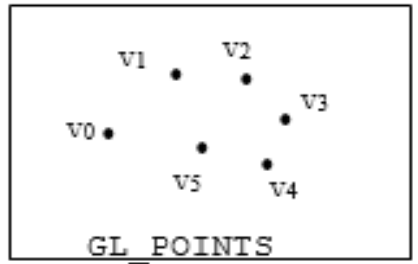
Grafica pe calculator
12.11.2023

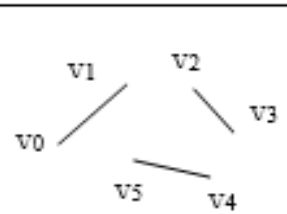
OpenGL

Tipurile de primitive geometrice

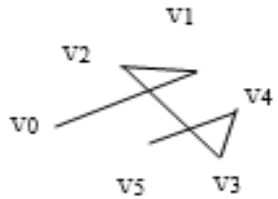
Proprietate	Primitivă
GL_POINTS	Desenează n puncte
GL_LINES	Desenează segmentele de dreaptă izolate (v_0,v_1), (v_2,v_3),… ş.a.m.d. Dacă n este impar ultimul vârf este ignorat
GL_LINE_STRIP	Desenează linia poligonală formată din segmentele (v_0,v_1), (v_1,v_2), .. . …(v_{n-2}, v_{n-1})
GL_LINE_LOOP	La fel ca primitiva GL_LINE_STRIP, dar se mai desenează segmentul (v_n, v_0) care închide o buclă.
GL_TRIANGLES	Desenează o serie de triunghiuri folosind vârfurile (v_0,v_1,v_2), (v_3,v_4,v_5), ş.a.m.d. Dacă n nu este multiplu de 3, atunci ultimele 1 sau 2 vârfuri sunt ignorate.
GL_TRIANGLE_STRIP	Desenează o serie de triunghiuri folosind vârfurile (v_0,v_1,v_2), (v_2,v_1,v_3),..ş.a.m.d. Ordinea este aleasă astfel ca triunghiurile să aibă aceeaşi orientare, deci să poată forma o suprafaţă închisă.
GL_TRIANGLE_FAN	Desenează triunghiurile (v_0,v_1,v_2), (v_0,v_2,v_3), ş.a.m.d.
GL_QUADS	Desenează o serie de patrulatere (v_0,v_1,v_2,v_3), (v_4,v_5,v_6,v_7), ş.a.m.d. Dacă n nu este multiplu de 4, ultimele 1, 2 sau 3 vârfuri sunt ignorate.
GL_QUADS_STRIP	Desenează o serie de patrulatere (v_0,v_1,v_3,v_2), (v_3,v_2,v_5,v_4), ş.a.m.d Dacă $n < 4$, nu se desenază nimic. Dacă n este impar, ultimul vârf este ignorat.
GL_POLYGON	Desenează un poligon cu n vârfuri, $(v_0,v_1,…v_{n-1})$. Dacă poligonul nu este convex, rezultatul este impredictibil.



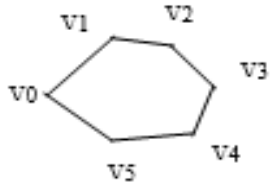
GL_POINTS



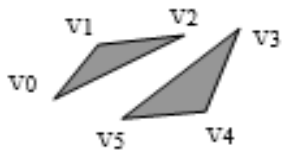
GL_LINES



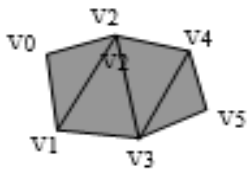
GL_LINE_STRIP



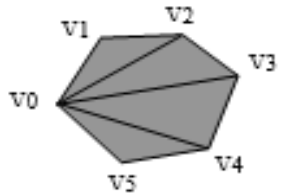
GL_LINE_LOOP



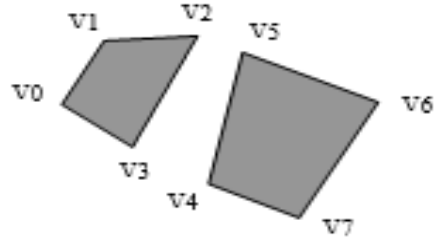
GL_TRIANGLES



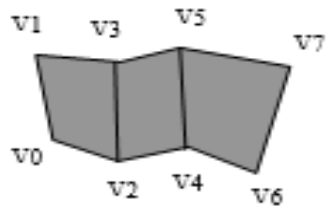
GL_TRIANGLE_STRIP



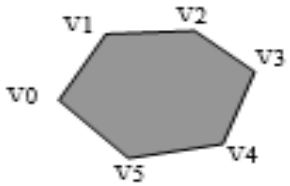
GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP



GL_POLYGON

GLUT include o serie de rutine pentru generarea de obiecte geometrice 3D ușor de recunoscut.

The routines generate normals appropriate for lighting but do not generate texture coordinates (except for the teapot).

- 11.1 glutSolidSphere, glutWireSphere
- 11.2 glutSolidCube, glutWireCube
- 11.3 glutSolidCone, glutWireCone
- 11.4 glutSolidTorus, glutWireTorus
- 11.5 glutSolidDodecahedron, glutWireDodecahedron
- 11.6 glutSolidOctahedron, glutWireOctahedron
- 11.7 glutSolidTetrahedron, glutWireTetrahedron
- 11.8 glutSolidIcosahedron, glutWireIcosahedron
- 11.9 glutSolidTeapot, glutWireTeapot

Afişarea obiectelor 3D predefinite

GLUT conţine funcţii pentru afişarea următoarelor obiecte 3D:

con	icosaedru	teapot
cub	octaedru	tetraedru
dodecaedru	sfera	tor

Aceste obiecte pot fi afişate prin familii de curbe sau ca obiecte solide.

Exemplu: funcţii de desenare cub, sferă si tor prin două familii de curbe şi ca solide.

Desenare cub de latură size prin două familii de curbe

void **glutWireCube**(GLdouble *size*);

Desenare cub solid de latură size

void **glutSolidCube**(GLdouble *size*);

Desenare sferă prin două familii de curbe

void **glutWireSphere**(GLdouble *radius*, GLint *slices*, GLint *stacks*);

Desenare sferă solidă

void **glutSolidSphere**(GLdouble *radius*, GLint *slices*, GLint *stacks*);

Desenare tor prin două familii de curbe

void **glutWireTorus**(GLdouble *innerRadius*, GLdouble *outerRadius*, GLint *nsides*, GLint *rings*);

Desenare tor solid

void **glutSolidTorus**(GLdouble *innerRadius*, GLdouble *outerRadius*, GLint *nsides*, GLint *rings*);

Desenare Teapot

void **glutWireTeapot**(GLdouble *size*);

void **glutSolidTeapot**(GLdouble *size*);

Alte funcţii sunt:

void glutWireIcosahedron(void);
void glutSolidIcosahedron(void);

void glutWireOctahedron(void);
void glutSolidOctahedron(void);

void glutWireTetrahedron(void);
void glutSolidTetrahedron(void);

void glutWireDodecahedron(GLdouble radius);
void glutSolidDodecahedron(GLdouble radius);

void glutWireCone(GLdouble radius, GLdouble height, GLint slices,GLint stacks);
void glutSolidCone(GLdouble radius, GLdouble height, GLint slices,GLint stacks);

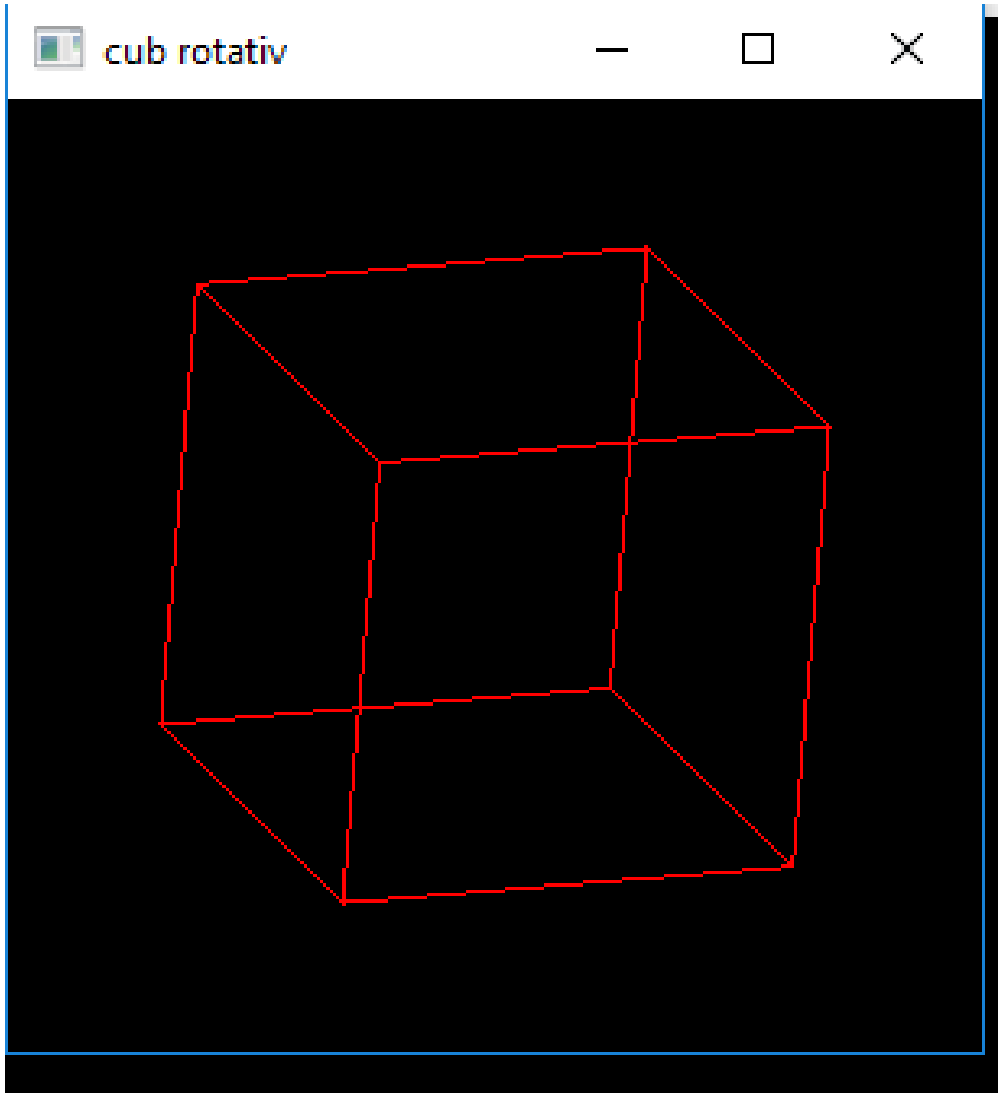
Toate aceste obiecte sunt desenate centrate în originea sistemului de coordonate real.
În momentul în care se fac modificări asupra unui obiect poate apare efectul de “pâlpâire” a imaginii.
Pentru evitarea acestui efect se asociază ferestrei aplicaţiei un buffer dublu. Astfel, într-un buffer se păstrează imaginea nemodificată (imaginea ce este afişată pe ecran), iar în cel de-al doilea se construieşte imaginea modificată. În momentul în care s-a terminat construirea imaginii modificate se interschimbă buffer-ele.

Pentru interschimbarea bufferelor se foloseşte funcţia: **glutSwapBuffers :**

```
void glutSwapBuffers(void) ;
```

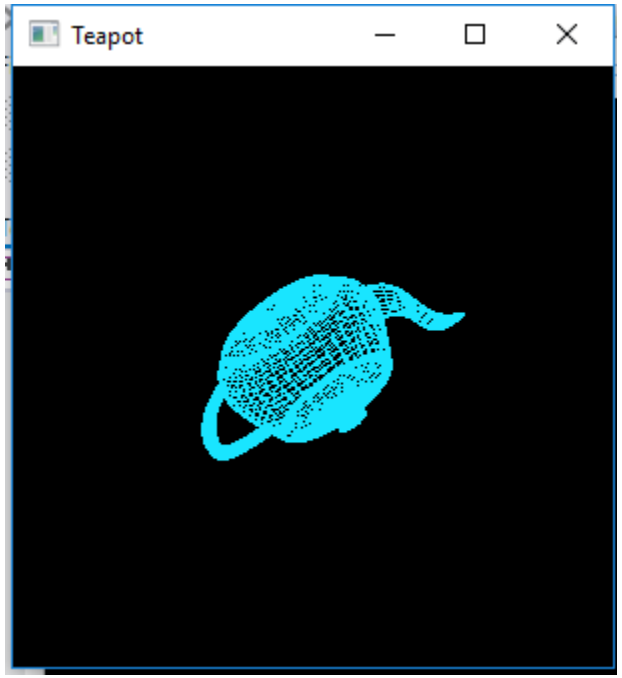
```
#include "stdafx.h"
#include <gl/freeglut.h>
 int GAngle = 30;

void display(void) {
	glClear(GL_COLOR_BUFFER_BIT);

	glLoadIdentity();// Initializeaza sistemul de coordonate
glRotated(GAngle, 1, 1, 1);//face o rotatie a unghiului in jurul vectorilor
x,y,z
	glColor3f(1, 0, 0);
	glutWireCube(0.5);//cub schelet
	GAngle = GAngle + 1;
	glFlush();
}
 void Timer(int extra) {
	glutPostRedisplay();//fereastra curenta este reafisata
	glutTimerFunc(30, Timer, 0);//seteaza timer-ul pentru fereastra curenta
}
```

```
int main(int argc, char** argv)
{

        glutInit(&argc, argv);

        //glutInitWindowSize(640, 480);
        glutCreateWindow("cub rotativ");
        glutDisplayFunc(display);
        glutTimerFunc(0, Timer, 0);
        glutMainLoop();
        return 0;
}
```



```
#include "stdafx.h"
#include <gl/freeglut.h>


void Display_my_pot()
{
static float alpha = 20;

glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.1, 0.9, 1.0);
glLoadIdentity();
glPopMatrix ();
 // Pop matricea veche fără transformări.
 // este salvează ecranul curent pentru o stiva
glRotatef(alpha, 1.9, 0.6, 0);
glutWireTeapot(0.3);
glPushMatrix ();  // Setați matricea curentă pe stivă
//încărca datele din stivă
 glFlush();
alpha = alpha + 0.1;

glutPostRedisplay();
}
```

```
int main(int argc, char** argv)
{

glutInit(&argc, argv);

//glutInitWindowSize(640, 480);
glutInitWindowPosition(10, 10);
glutCreateWindow("Teapot");
glutDisplayFunc(Display_my_pot);

glutMainLoop();
return 0;
}
```



Textures objects and parameters

https://open.gl/textures

O texture este o imagine 2D care este folosita pentru a adauga detalii obiectului.
Ne putem gandi la textura ca la o bucata de hartie (cu un desen pe ea) care este impaturita peste obiectul 3D.
Pentru ca putem adauga oricate detalii intr-o singura imagine, putem da iluzia ca obiectul este foarte detaliat fara sa adaugam vertexuri in plus.

Diferenta intre un obiect texturat si acelasi obiect netexturat este enorma din punct de vedere al acuratetei reprezentarii obiectului respectiv

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.

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, …

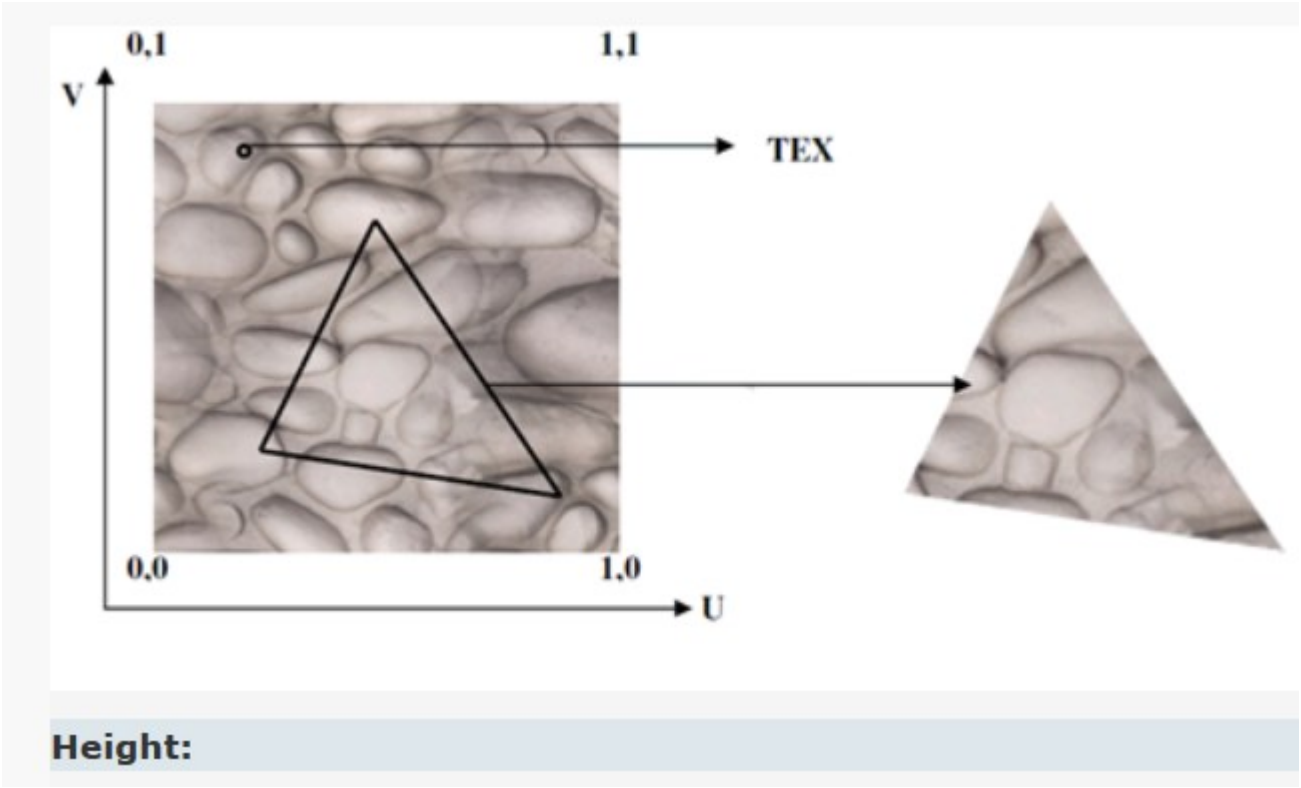
Maparea texturilor

Pentru a mapa o textura peste un triunghi, trebuie sa specificam in ce parte din textura corespunde fiecare vertex.

Asadar, fiecare vertex ar trebui sa aiba asociata o coordonata de textura (2D adica glm::vec2) care specifica partea din textura unde se afla. Interpolarea intre vertecsi se face in fragment shader.

Coordonatele de textura se afla in intervalul 0 si 1 pentru axele x si y (texturile sunt in general imagini 2D). Coordonatele texturii incep din punctul (0, 0) pentru coltul din stanga jos a imaginii pana la punctul (1, 1) care se afla in coltul din dreapta sus.

Un punct de pe imagine si care este in spatiul [0,1]x[0,1] se numeste texel, numele venind de la texture element.
In functie de coordonatele fiecarui vertex al triunghiului, partea din textura care este mapata peste triunghi poate fi diferita:



Adaugarea unei texturi

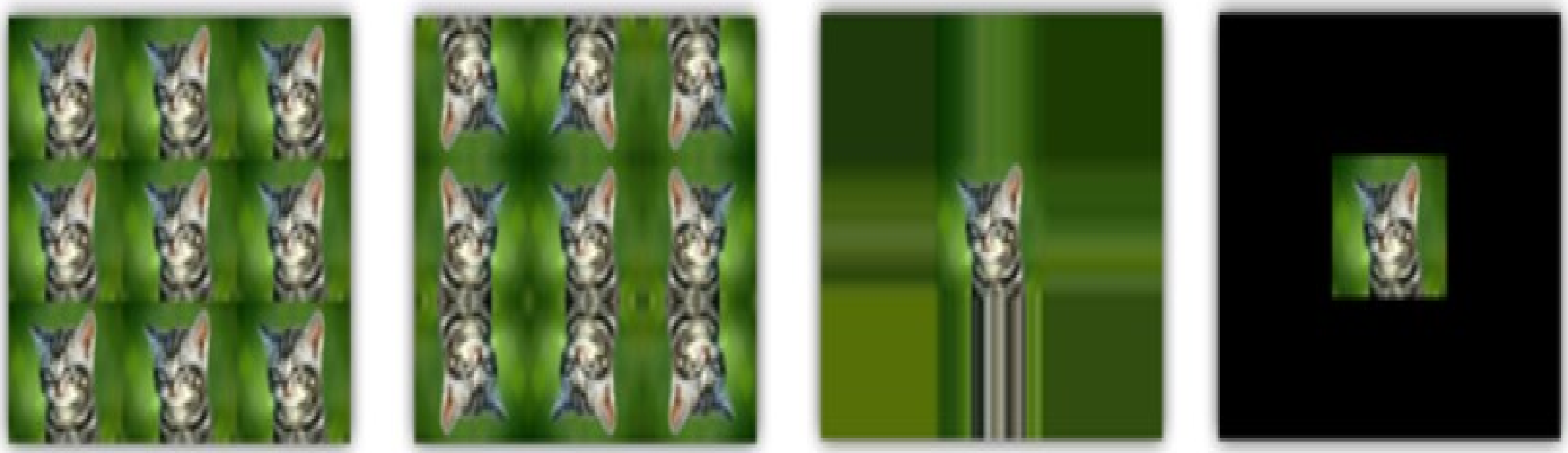
Pentru a construi o textura in OpenGL avem nevoie in primul rand de pixelii imaginii ce va fi folosita ca textura.
Acestia trebuie ori generati functional ori trebuie incarcati dintr-o imagine, iar acest procedeu este independent de OpenGL.

Pentru a citi texturi, se poate folosi clasa Texture2D :

Texture2D::Load2D(const char* fileName, GLenum wrapping_mode)

unde wrapping_mode poate fi

- GL_REPEAT : textura se repeta pe toata suprafata obiectului
- GL_MIRRORED_REPEAT: textura se repeta dar va fi vazuta in oglinda pentru repetarile impare
- GL_CLAMP_TO_EDGE: coordonatele vor fi intre 0 si 1
- GL_CLAMP_TO_BORDER: asemanator cu clamp to edge doar ca va exista un border


GL_REPEAT
GL_MIRRORED_REPEAT
GL_CLAMP_TO_EDGE
GL_CLAMP_TO_BORDER

Dupa ce avem pixelii imaginii incarcate putem genera o textura de OpenGL cu comanda:

```
unsigned int gl_texture_object;
glGenTextures(1, &gl_texture_object);
```

In OpenGL nu lucram direct cu textura ci trebuie sa o mapam la un punct de legare.
Mai mult, la randul lor punctele de legare pentru texturi sunt dependente de unitatile de texturare.
O unitate de texturare e foarte similara ca si concept cu pipe-urile pe care trimitem atribute.
Setam unitatea de texturare cu comanda (o singura unitate de texturare poate fi activa):

```
glActiveTexture(GL_TEXTURE0 + nr_unitatii_de_texturare_dorite);
```

Pentru a lega obiectul de tip textura generat anterior la unitatea de textura activa folosim:

```
glBindTexture(GL_TEXTURE_2D, gl_texture_object);
```

Pentru a incarca date intr-o textura folosim comanda:

```
glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE data);
```

```
glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
```

Argumentele functie:
GL_TEXTURE_2D, inseamna ca aceasta functie va asocia texturii binduite o textura 2D (deci daca avem binduite un GL_TEXTURE_1D sau GL_TEXTURE_3D, acestea nu vor fi afectate).

- al 2-lea argument specifica nivelul de mipmap pentru care vrem sa cream imaginea. Vom explica ce este un mipmap pe parcusul laboratorului. Pentru moment, putem sa lasam valoarea aceasta 0.
- al 3-lea argument specifica formatul in care vrem sa fie stocata imaginea. In cazul nostru este RGB.
- al 4-lea si al 5-lea argument seteaza marimea imaginii.
- al 6-lea argument ar trebui sa fie mereu 0 (legacy stuff)
- arg. 7 si 8 specifica formatul si tipul de date al imaginii sursa.
- arg. 9 il reprezinta vectorul de date al imaginii.

glTextImage2D incarca o imagine definita printr-un array de unsigned char-uri pe textura legata in punctul de legare GL_TEXTURE_2D al unitatii de texturare legate curent, nivelul 0 (o sa luam aceasta constanta ca atare pentru moment), cu formatul intern GL_RGB cu lungimea width si cu inaltimea height, din formatul GL_RGB.

Datele citite sunt de tip GL_UNSIGNED_BYTE (adica unsigned char) si sunt citite de la adresa data.

Etape pentru utilizare texturii

```
glActiveTexture(GL_TEXTURE0);

glBindTexture(GL_TEXTURE_2D, texture1);

glUniform1i(glGetUniformLocation(ourShader.Program, "texture_1"), 0);

glActiveTexture(GL_TEXTURE1);

glBindTexture(GL_TEXTURE_2D, texture2);

glUniform1i(glGetUniformLocation(ourShader.Program, "texture_2"), 1);
```

Unitatea de texturare este folositoare in momentul in care vrem sa atribuim textura unei variabile uniforme din shader.

Scopul acestui mecanism este de a ne permite sa folosim mai mult de 1 textura in shaderele noastre.

Prin folosirea unitatilor de texturare, putem face bind la multiple texturi atat timp cat le setam ca fiind active.

OpenGl are minim 16 unitati de texturare care pot fi activate folosind GL_TEXTURE0 ..16

Se pot obtine GL_TEXTURE8 prin operatia GL_TEXTURE0 + 8.

Filtrarea

Coordonatele prin care se mapeaza vertecsii obiectului pe textura nu depind de resolutia imaginii ci pot fi oricare valoare float intre 0 si 1, iar OpenGl trebuie sa-si dea seama ce texel (texture pixel) sa mapeze pentru coordonatele date.

Pentru a rezolva aceasta problema se foloste filtrarea, care este o metoda de esantionare si reconstructie pe un semnal.

Reconstructia reprezinta procesul prin care utilizand acesti pixeli putem obtine valori pentru oricare din pozitiile in textura (adica nu neaparat exact la coordonatele din mijlocul pixelului, acolo unde a fost esantionata realitatea in spatiul post proiectie).

Pentru a face acest proces mai usor, OpenGl are o serie de filtre care pot fi folosite pentru a obtine maparea dorita, iar cele mai des utilizate sunt :

GL_NEAREST si GL_LINEAR.

GL_NEAREST (care se mai numeste si nearest neighbor filtering) este filtrarea default pentru OpenGL.
Cand este folosit acest filtru, OpenGL selecteaza pixelul al carui centru este cel mai aproape de coordonatele de texturare.

Se pot vedea 4 pixeli unde crucea reprezinta exact coordonatele de texturare. Pixelul cel din stanga sus are centrul cel mai aproape de coordonata texturii si este ales :

https://learnopengl.com/Getting-started/Textures

GL_LINEAR (cunoscut drept filtrare biliniara) ia valoarea interpolata din **texelii vecini ai coordonatei de texturare**, aproximand astfel culoarea mai bine. Cu cat distanta de la coordonata de texturare pana la centrul texelului este mai mica, cu atat contributia culorii acelui texel este mai mare.

Se pot folosi filtre diferite pentru cazul in care vrem sa marim imaginea si cand vrem sa o micsoram (upscaling si downscaling).

Filtrul se specifica folosind metoda **glTextPameter** :

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glTexParameter.xhtml

http://www.opengl-tutorial.org/beginners-tutorials/tutorial-5-a-textured-cube/

https://learnopengl.com/Advanced-OpenGL/Cubemaps

https://www.opengl.org/archives/resources/code/samples/glut_examples/examples/examples.html

https://docs.oracle.com/javase/tutorial/2d/index.html

https://docs.oracle.com/cd/E17802_01/j2se/javase/technologies/desktop/java3d/forDevelopers/j3dguide/Intro.doc.html