GL\_MAZE – Proiect2

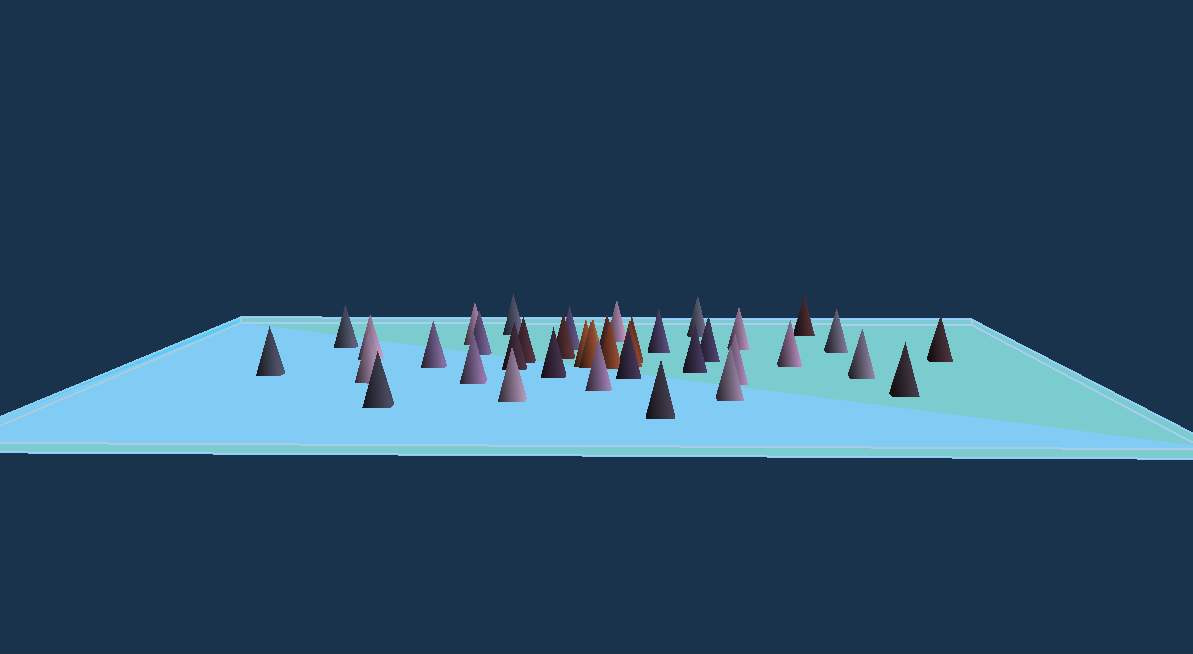
Documentatie

**Conceptul proiectului**

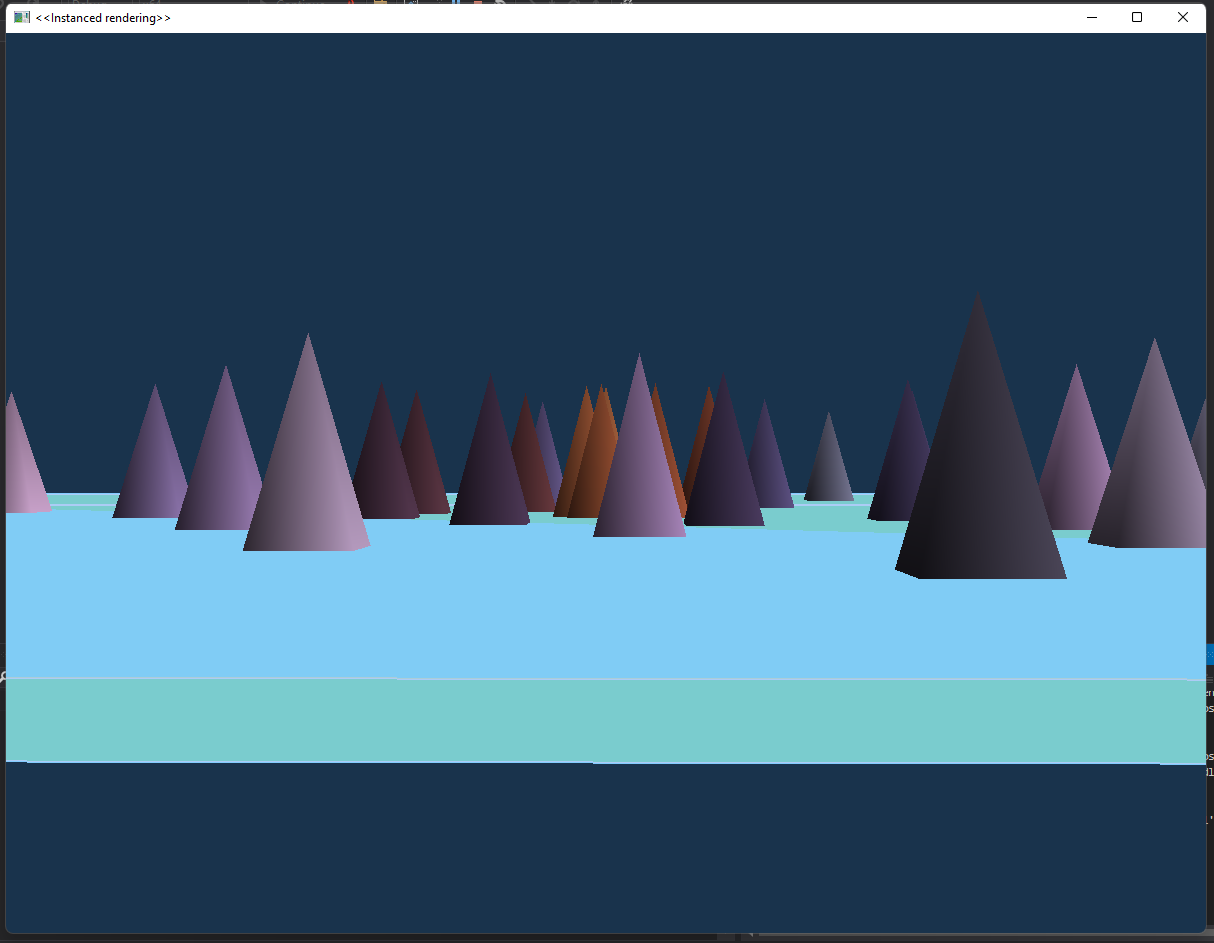
Proiectul ilustrează o scenă 3D complexă în care accentul se pune pe tehnica randării instanțiate, îmbinând totodată tehnicile de iluminare a piramidelor dispuse pe un model. Aceasta scenă abstractă prezintă forma un set de piramide dispuse după un model matematic, fiind perfect utilizabilă ca punct de plecare pentru multiple idei de jocuri.

**Elemente incluse (Tehnici de implementare)**

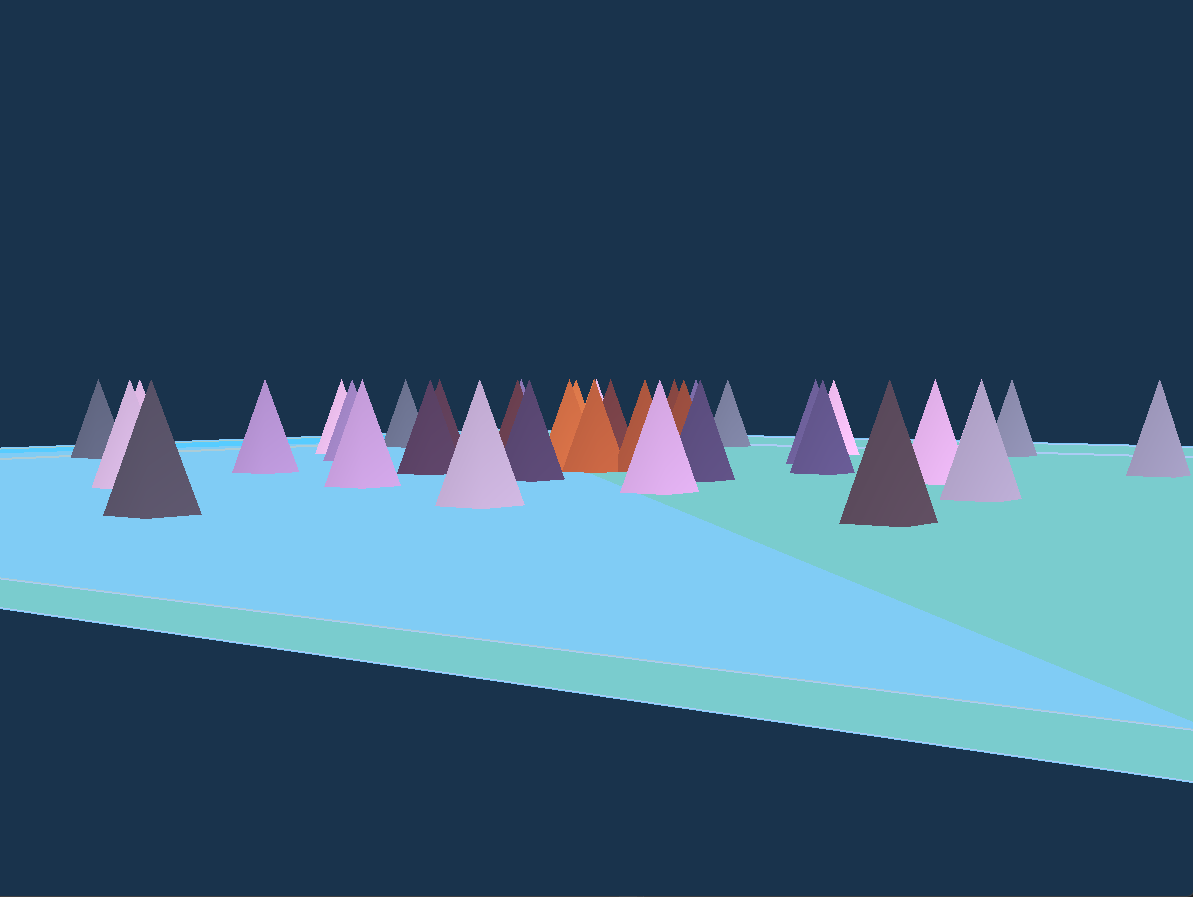
1.Reprezentarea obiectelor 3D:



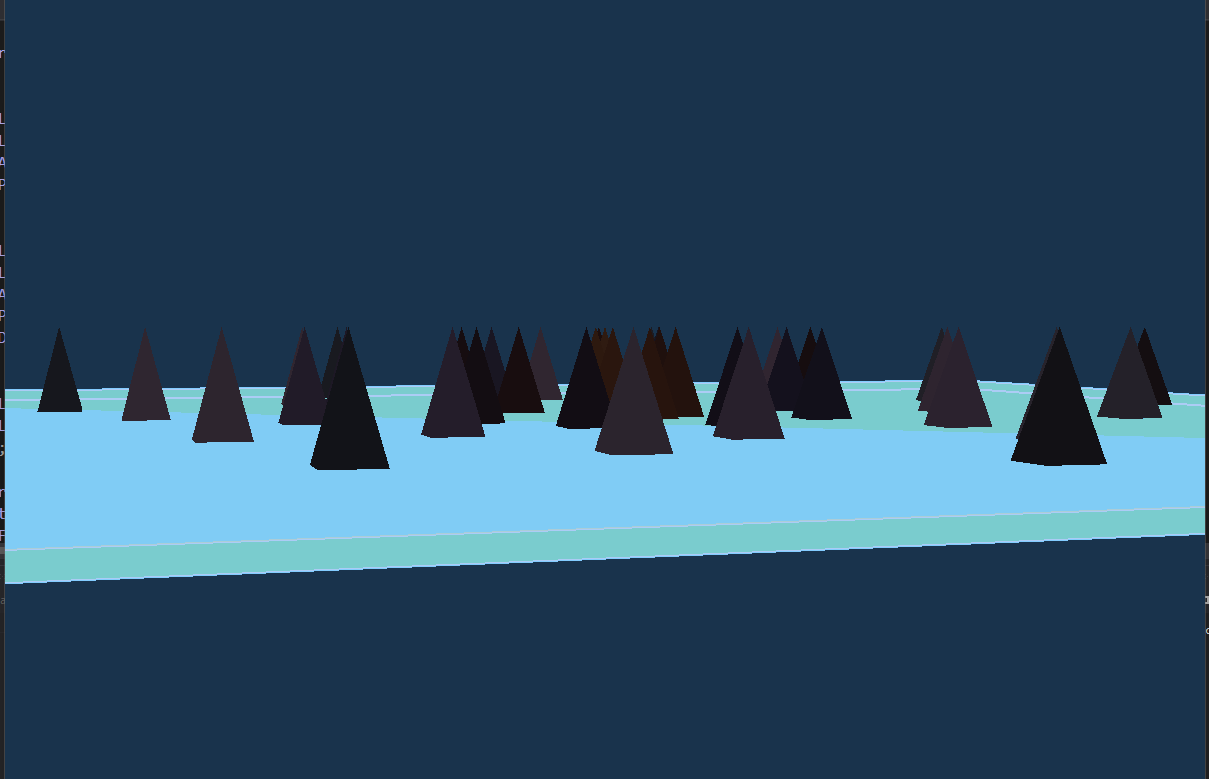
2&3. Iluminare&Umbre

Cand lumina bate din dreapta:

Cand lumina bate din fata

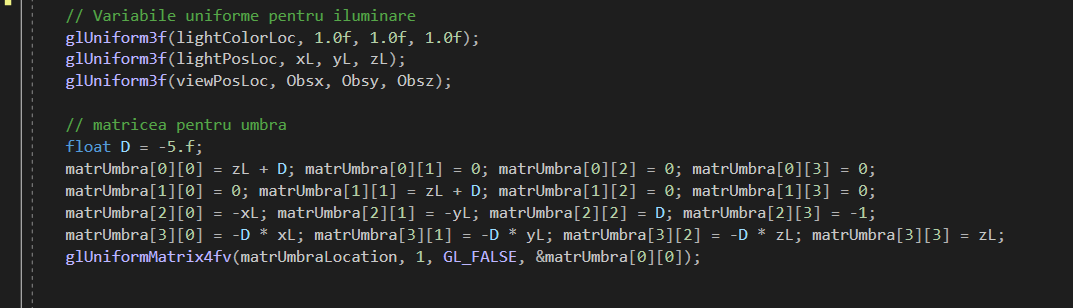


Cand lumina bate din spate:



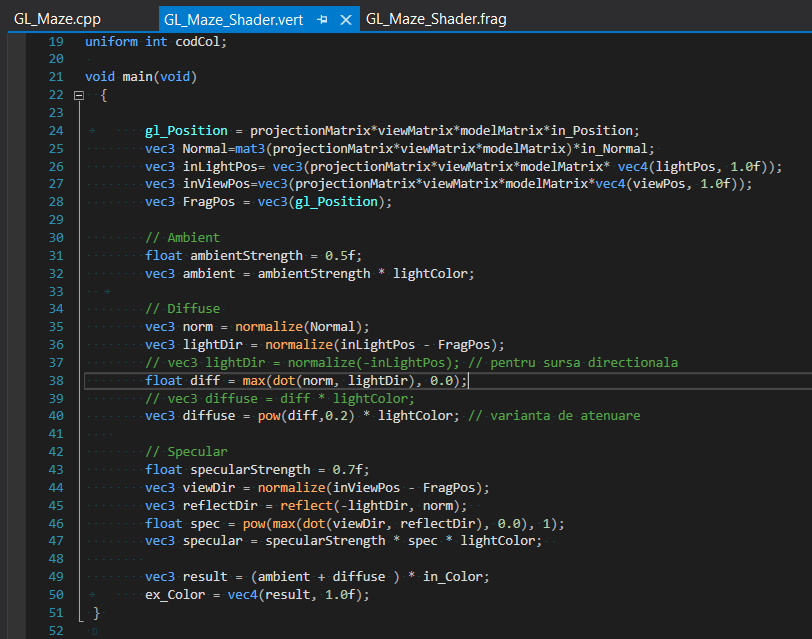
* In Render Function:

Putem observa în acest fragment de cod modul în care este controlată umbrirea:

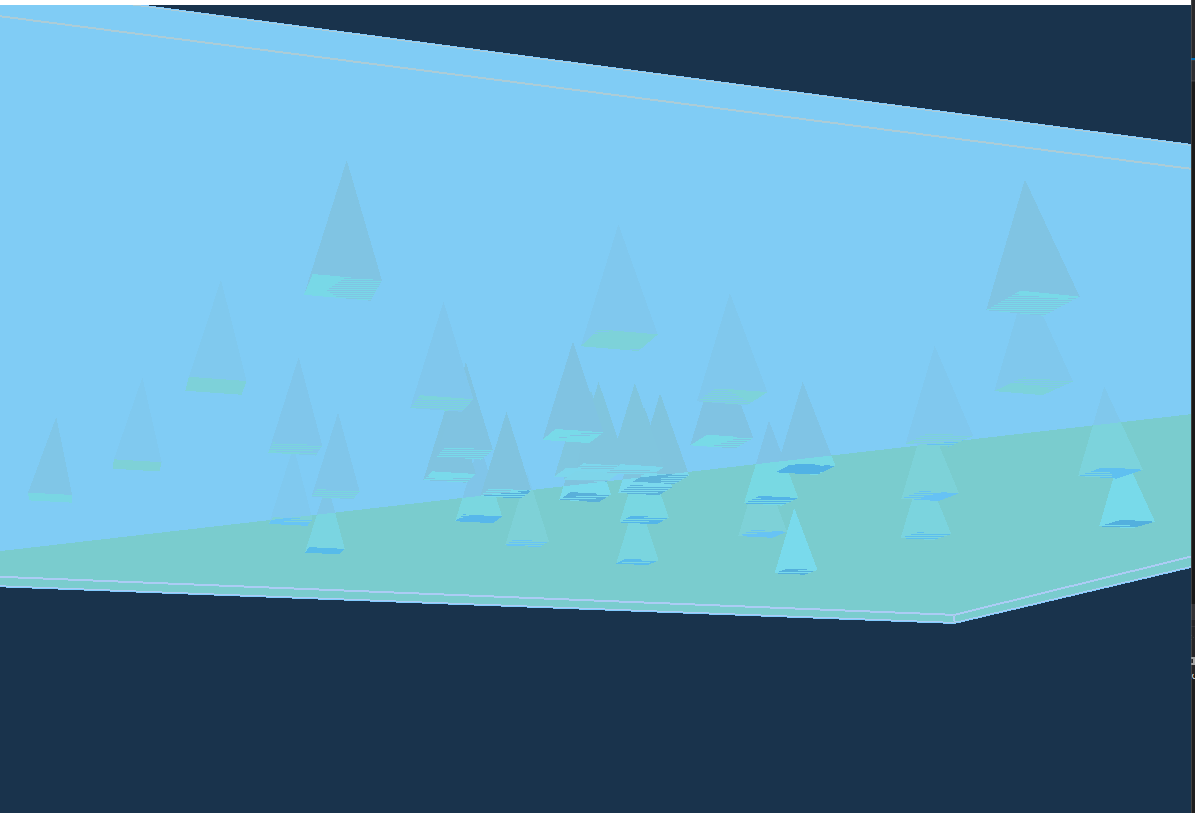


In Shaderul de GL\_Maze\_Shader.vert:

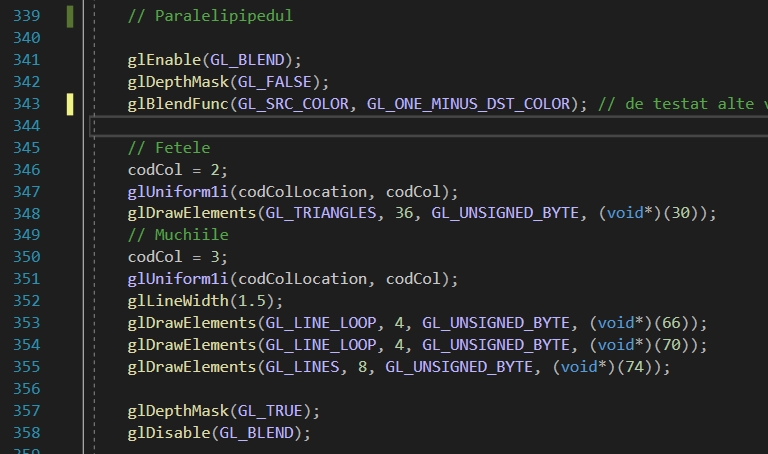
În acest shader observăm implementat modelul lui Phong ce ne permite să ne jucăm cu iluminarea scenei:



1. Amestecare & Transparență



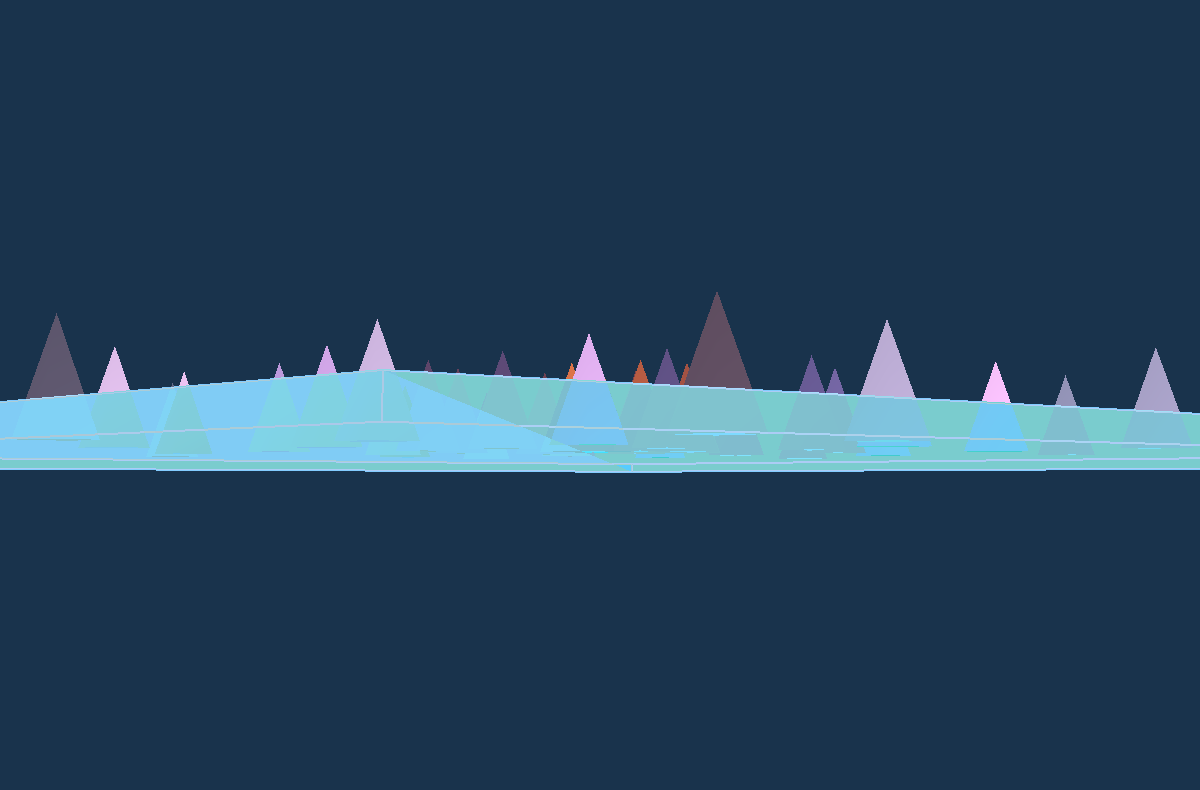
Pentru ca paralelipipedul sa poata fi usor transparent, s-a aplicat amestecarea intre culorile acestuia si culorile obiectelor din spatele lui (se observa GL\_SRC\_COLOR si GL\_ONE\_MINUS\_DST\_COLOR).



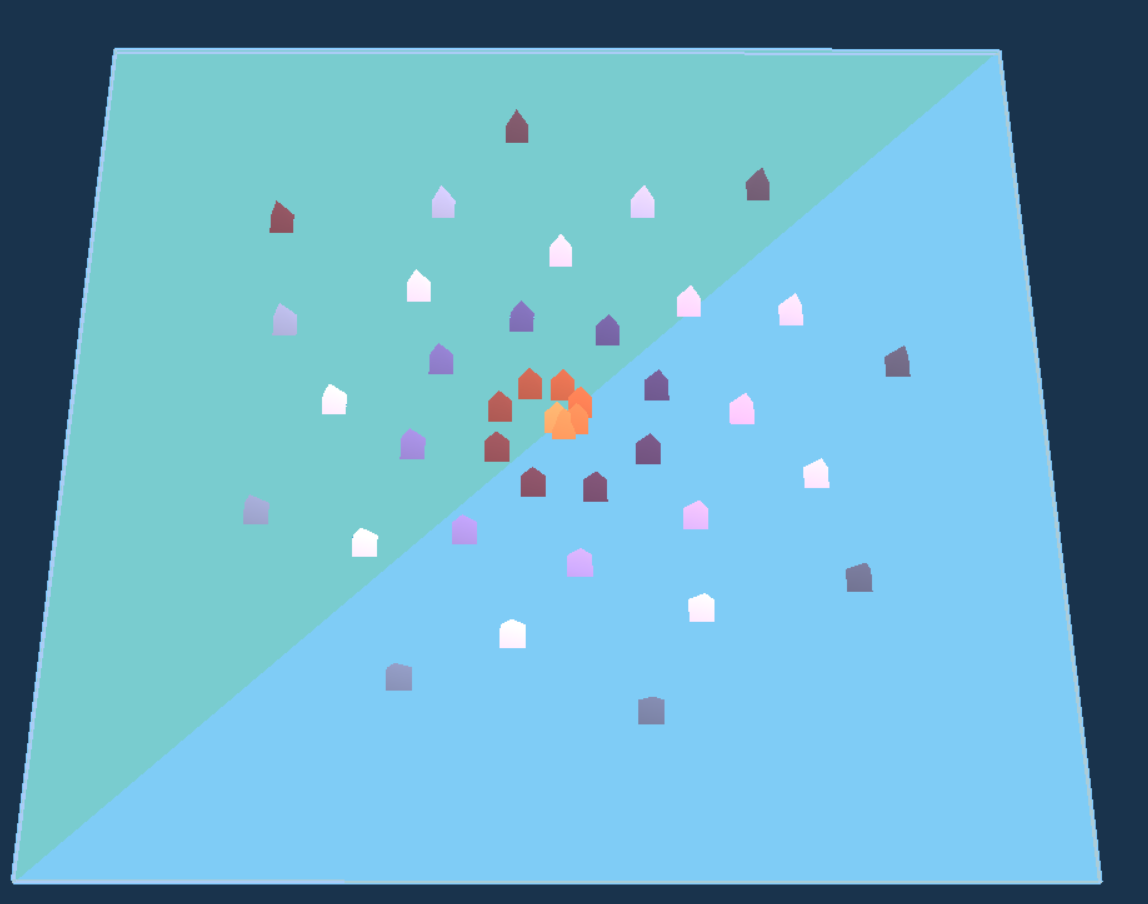
**De ce este original?**

Ceea ce face proiectul sa fie original este creearea iluzii ca paralelipipedul ar fi facut din sticla.



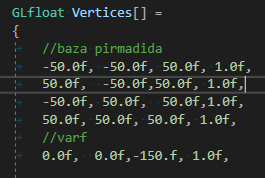


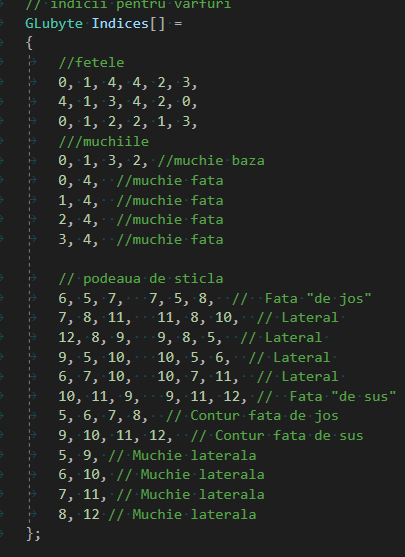
Dar si generarea piramidelor in spirala:



Modelul după care se generează spirală se bazează pe formula:

MatModel[n] = glm::translate(glm::mat4(1.0f), glm::vec3(INSTANCE\_COUNT \* n \* sin(n \* 180 / PI), INSTANCE\_COUNT \* n \* cos(n \* 180 / PI), 0.0)); ,piramidă ce servește drept model având aceste 5 puncte

Apoi, formele sunt generate cu ajutorul tehnicii indicării indicilor:

**Contribuții individuale:**

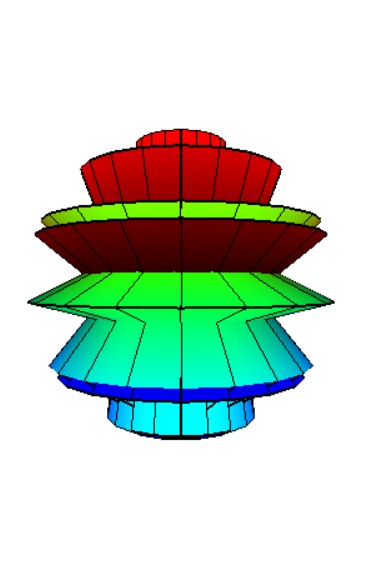
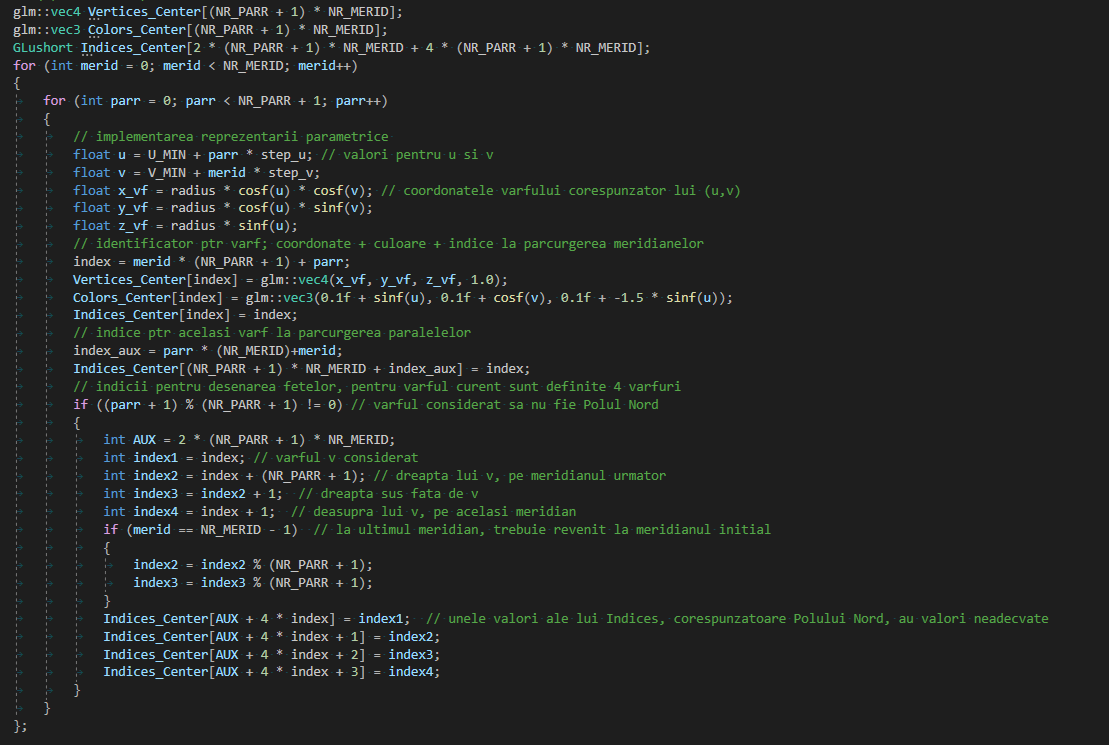
***Octavian-Florin Staicu:***

* Reprezentarea instanțiată a piramidelor tridimensionale
* Distribuția acestora după un model matematic ( o spirală )
* Tunarea parametriilor ce țin de numărul de piramide, distanța dintre ele, iluminarea acestora

***Iulia-Andreea Barbu:***

* Implementarea podelei transparente pe care se situează scena
* Implementarea modelului de iluminare
* Testing

**Idei neutilizate în proiectul final:**

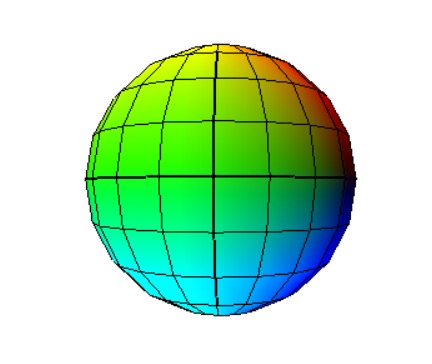
* Centrului modelului ar fi putut diferi de restul piramidelor, utilizând acest concept:

* Animația piramidelor:

Am fi putut folosi cuaternioni pentru a anima piramidele cu o mișcare de rotație

* Mini-joculeț „Evită piramidele”:

Am fi putut face ca acea spirală de piramide să se rotească, iar scopul joculețului ar fi fost ca mingea să poată ajunge la sculptura centrală fără să lovească piramidele

* 

# **// Anexă**

# //GL\_Maze.cpp

// Functii de desenare in Open GL. Randare instantiata

#include <windows.h> // biblioteci care urmeaza sa fie incluse

#include <stdlib.h> // necesare pentru citirea shader-elor

#include <stdio.h>

#include <math.h>

#include <iostream>

#include <GL/glew.h> // glew apare inainte de freeglut

#include <GL/freeglut.h> // nu trebuie uitat freeglut.h

#include "loadShaders.h"

#include "glm/glm/glm.hpp"

#include "glm/glm/gtc/matrix\_transform.hpp"

#include "glm/glm/gtx/transform.hpp"

#include "glm/glm/gtc/type\_ptr.hpp"

using namespace std;

#define INSTANCE\_COUNT 40

const GLfloat PI = 3.141592f;

// identificatori

GLuint

VaoId,

VBPos,

VBCol,

VBModelMat,

EboId,

ColorBufferId,

ProgramId,

viewLocation,

projLocation,

codColLocation,

lightColorLoc,

lightPosLoc,

viewPosLoc,

matrUmbraLocation,

codCol;

// variabile pentru matricea de vizualizare

float Refx = 0.0f, Refy = 0.0f, Refz = 0.0f;

float Alpha = 0.0f, Beta = 0.0f, dist = 200.0f;

float Obsx, Obsy, Obsz;

float Vx = 0.0f, Vy = 0.0f, Vz = -1.0f;

float incr\_alpha1 = 0.01f, incr\_alpha2 = 0.01f;

// variabile pentru matricea de proiectie

float width = 800, height = 600, znear = 1, fov = 30;

// vectori

glm::vec3 Obs, PctRef, Vert;

glm::vec4 Colors[INSTANCE\_COUNT];

// matrice utilizate

glm::mat4 view, projection, MatModel[INSTANCE\_COUNT], matrUmbra;

// sursa de lumina

float xL = 500.f, yL = 0.f, zL = 400.f;

void processNormalKeys(unsigned char key, int x, int y)

{

switch (key) {

case '-':

dist += 50.0;

break;

case '+':

dist -= 50.0;

break;

default:

break;

}

}

void processSpecialKeys(int key, int xx, int yy) {

switch (key)

{

case GLUT\_KEY\_LEFT:

Beta -= 0.01f;

break;

case GLUT\_KEY\_RIGHT:

Beta += 0.01f;

break;

case GLUT\_KEY\_UP:

Alpha += incr\_alpha1;

if (abs(Alpha - PI / 2) < 0.05)

{

incr\_alpha1 = 0.f;

}

else

{

incr\_alpha1 = 0.01f;

}

break;

case GLUT\_KEY\_DOWN:

Alpha -= incr\_alpha2;

if (abs(Alpha + PI / 2) < 0.05)

{

incr\_alpha2 = 0.f;

}

else

{

incr\_alpha2 = 0.01f;

}

break;

}

}

void CreateVBO(void)

{

// Varfurile

GLfloat Vertices[] =

{

//baza pirmadida

-50.0f, -50.0f, 50.0f, 1.0f,

50.0f, -50.0f,50.0f, 1.0f,

-50.0f, 50.0f, 50.0f,1.0f,

50.0f, 50.0f, 50.0f, 1.0f,

//varf

0.0f, 0.0f,-150.f, 1.0f,

// podeaua din sticla

// varfurile din planul z=-50

// coordonate

-2000.0f, -2000.0f, 80.0f, 1.0f,

2000.0f, -2000.0f, 80.0f, 1.0f,

2000.0f, 2000.0f, 80.0f, 1.0f,

-2000.0f, 2000.0f, 80.0f, 1.0f,

// varfurile din planul z=+50

// coordonate

-2000.0f, -2000.0f, 50.0f, 1.0f,

2000.0f, -2000.0f, 50.0f, 1.0f,

2000.0f, 2000.0f, 50.0f, 1.0f,

-2000.0f, 2000.0f, 50.0f, 1.0f,

};

// Culorile instantelor

for (int n = 0; n < INSTANCE\_COUNT; n++)

{

float a = float(n) / 4.0f;

float b = float(n) / 5.0f;

float c = float(n) / 6.0f;

Colors[n][0] = 0.35f + 0.30f \* (sinf(a + 2.0f) + 1.0f);

Colors[n][1] = 0.25f + 0.25f \* (sinf(b + 3.0f) + 1.0f);

Colors[n][2] = 0.25f + 0.35f \* (sinf(c + 4.0f) + 1.0f);

Colors[n][3] = 1.0f;

}

// Matricele instantelor

//int fib1=0, fib2=1;

for (int n = 0; n < INSTANCE\_COUNT; n++)

{

//if (n < 3)

//MatModel[n] = glm::mat4(0);

//else

//int fib = fib1 + fib2;

//fib1 = fib2;

//fib2 = fib;

//MatModel[n] = glm::translate(glm::mat4(1.0f), glm::vec3(fib \* sin( n \* 180 / PI), fib \* cos( n \* 180 / PI), 0.0)) ;

MatModel[n] = glm::translate(glm::mat4(1.0f), glm::vec3(INSTANCE\_COUNT \* n \* sin(n \* 180 / PI), INSTANCE\_COUNT \* n \* cos(n \* 180 / PI), 0.0));

}

// indicii pentru varfuri

GLubyte Indices[] =

{

//fetele

0, 1, 4, 4, 2, 3,

4, 1, 3, 4, 2, 0,

0, 1, 2, 2, 1, 3,

///muchiile

0, 1, 3, 2, //muchie baza

0, 4, //muchie fata

1, 4, //muchie fata

2, 4, //muchie fata

3, 4, //muchie fata

// podeaua de sticla

6, 5, 7, 7, 5, 8, // Fata "de jos"

7, 8, 11, 11, 8, 10, // Lateral

12, 8, 9, 9, 8, 5, // Lateral

9, 5, 10, 10, 5, 6, // Lateral

6, 7, 10, 10, 7, 11, // Lateral

10, 11, 9, 9, 11, 12, // Fata "de sus"

5, 6, 7, 8, // Contur fata de jos

9, 10, 11, 12, // Contur fata de sus

5, 9, // Muchie laterala

6, 10, // Muchie laterala

7, 11, // Muchie laterala

8, 12 // Muchie laterala

};

// generare buffere

glGenVertexArrays(1, &VaoId);

glGenBuffers(1, &VBPos);

glGenBuffers(1, &VBCol);

glGenBuffers(1, &VBModelMat);

glGenBuffers(1, &EboId);

// legarea VAO

glBindVertexArray(VaoId);

// 0: Pozitie

glBindBuffer(GL\_ARRAY\_BUFFER, VBPos);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(Vertices), Vertices, GL\_STATIC\_DRAW);

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 4, GL\_FLOAT, GL\_FALSE, 4 \* sizeof(GLfloat), (GLvoid\*)0);

// 1: Culoare

glBindBuffer(GL\_ARRAY\_BUFFER, VBCol); // legare buffer

glBufferData(GL\_ARRAY\_BUFFER, sizeof(Colors), Colors, GL\_STATIC\_DRAW);

glEnableVertexAttribArray(1);

glVertexAttribPointer(1, 4, GL\_FLOAT, GL\_FALSE, sizeof(glm::vec4), (GLvoid\*)0);

glVertexAttribDivisor(1, 1); // rata cu care are loc distribuirea culorilor per instanta

// 2..5 (2+i): Matrice de pozitie

glBindBuffer(GL\_ARRAY\_BUFFER, VBModelMat);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(MatModel), MatModel, GL\_STATIC\_DRAW);

for (int i = 0; i < 4; i++) // Pentru fiecare coloana

{

glEnableVertexAttribArray(2 + i);

glVertexAttribPointer(2 + i, // Location

4, GL\_FLOAT, GL\_FALSE, // vec4

sizeof(glm::mat4), // Stride

(void\*)(sizeof(glm::vec4) \* i)); // Start offset

glVertexAttribDivisor(2 + i, 1);

}

// Indicii

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, EboId);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(Indices), Indices, GL\_STATIC\_DRAW);

glEnableVertexAttribArray(3); // atributul 2 = normale

}

void DestroyVBO(void)

{

glDisableVertexAttribArray(2);

glDisableVertexAttribArray(1);

glDisableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glDeleteBuffers(1, &VBPos);

glDeleteBuffers(1, &VBCol);

glDeleteBuffers(1, &VBModelMat);

glDeleteBuffers(1, &EboId);

glBindVertexArray(0);

glDeleteVertexArrays(1, &VaoId);

}

void CreateShaders(void)

{

ProgramId = LoadShaders("GL\_Maze\_Shader.vert", "GL\_Maze\_Shader.frag");

glUseProgram(ProgramId);

}

void DestroyShaders(void)

{

glDeleteProgram(ProgramId);

}

void Initialize(void)

{

glClearColor(0.1f, 0.2f, 0.3f, 1.0f); // culoarea de fond a ecranului

CreateVBO();

CreateShaders();

viewLocation = glGetUniformLocation(ProgramId, "viewMatrix");

projLocation = glGetUniformLocation(ProgramId, "projectionMatrix");

matrUmbraLocation = glGetUniformLocation(ProgramId, "matrUmbra");

lightColorLoc = glGetUniformLocation(ProgramId, "lightColor");

lightPosLoc = glGetUniformLocation(ProgramId, "lightPos");

viewPosLoc = glGetUniformLocation(ProgramId, "viewPos");

codColLocation = glGetUniformLocation(ProgramId, "codCol");

}

void RenderFunction(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glEnable(GL\_DEPTH\_TEST);

// CreateVBO(); // decomentati acest rand daca este cazul

glBindVertexArray(VaoId);

glBindBuffer(GL\_ARRAY\_BUFFER, VBPos);

glBindBuffer(GL\_ARRAY\_BUFFER, VBCol);

glBindBuffer(GL\_ARRAY\_BUFFER, VBModelMat);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, EboId);

//pozitia observatorului

Obsx = Refx + dist \* cos(Alpha) \* cos(Beta);

Obsy = Refy + dist \* cos(Alpha) \* sin(Beta);

Obsz = Refz + dist \* sin(Alpha);

// reperul de vizualizare

glm::vec3 Obs = glm::vec3(Obsx, Obsy, Obsz); // se schimba pozitia observatorului

glm::vec3 PctRef = glm::vec3(Refx, Refy, Refz); // pozitia punctului de referinta

glm::vec3 Vert = glm::vec3(Vx, Vy, Vz); // verticala din planul de vizualizare

view = glm::lookAt(Obs, PctRef, Vert);

glUniformMatrix4fv(viewLocation, 1, GL\_FALSE, &view[0][0]);

// matricea de proiectie

projection = glm::infinitePerspective(fov \* PI / 180, GLfloat(width) / GLfloat(height), znear);

glUniformMatrix4fv(projLocation, 1, GL\_FALSE, &projection[0][0]);

// Variabile uniforme pentru iluminare

glUniform3f(lightColorLoc, 1.0f, 1.0f, 1.0f);

glUniform3f(lightPosLoc, xL, yL, zL);

glUniform3f(viewPosLoc, Obsx, Obsy, Obsz);

// matricea pentru umbra

float D = -5.f;

matrUmbra[0][0] = zL + D; matrUmbra[0][1] = 0; matrUmbra[0][2] = 0; matrUmbra[0][3] = 0;

matrUmbra[1][0] = 0; matrUmbra[1][1] = zL + D; matrUmbra[1][2] = 0; matrUmbra[1][3] = 0;

matrUmbra[2][0] = -xL; matrUmbra[2][1] = -yL; matrUmbra[2][2] = D; matrUmbra[2][3] = -1;

matrUmbra[3][0] = -D \* xL; matrUmbra[3][1] = -D \* yL; matrUmbra[3][2] = -D \* zL; matrUmbra[3][3] = zL;

glUniformMatrix4fv(matrUmbraLocation, 1, GL\_FALSE, &matrUmbra[0][0]);

// Piramidele

//

// Fetele

codCol = 0;

glUniform1i(codColLocation, codCol);

glDrawElementsInstanced(GL\_TRIANGLES, 18, GL\_UNSIGNED\_BYTE, 0, INSTANCE\_COUNT);

// Muchiile

/\*codCol = 1;

glUniform1i(codColLocation, codCol);

glLineWidth(2.5);

glDrawElementsInstanced(GL\_LINE\_LOOP, 4, GL\_UNSIGNED\_BYTE, (void\*)(18), INSTANCE\_COUNT);

glDrawElementsInstanced(GL\_LINE\_LOOP, 8, GL\_UNSIGNED\_BYTE, (void\*)(22), INSTANCE\_COUNT);\*/

// Cubul

glEnable(GL\_BLEND);

glDepthMask(GL\_FALSE);

glBlendFunc(GL\_SRC\_COLOR, GL\_ONE\_MINUS\_DST\_COLOR); // de testat alte variante https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glBlendFunc.xhtml si factori-destinatie: GL\_ONE, GL\_DST\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA

//glBlendEquation(GL\_FUNC\_ADD);

// Fetele

codCol = 2;

glUniform1i(codColLocation, codCol);

glDrawElements(GL\_TRIANGLES, 36, GL\_UNSIGNED\_BYTE, (void\*)(30));

// Muchiile

codCol = 3;

glUniform1i(codColLocation, codCol);

glLineWidth(1.5);

glDrawElements(GL\_LINE\_LOOP, 4, GL\_UNSIGNED\_BYTE, (void\*)(66));

glDrawElements(GL\_LINE\_LOOP, 4, GL\_UNSIGNED\_BYTE, (void\*)(70));

glDrawElements(GL\_LINES, 8, GL\_UNSIGNED\_BYTE, (void\*)(74));

glDepthMask(GL\_TRUE);

glDisable(GL\_BLEND);

glutSwapBuffers();

glFlush();

}

void Cleanup(void)

{

DestroyShaders();

DestroyVBO();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH | GLUT\_DOUBLE);

glutInitWindowPosition(100, 100);

glutInitWindowSize(1200, 900);

glutCreateWindow("GL\_Maze");

glewInit();

Initialize();

glutDisplayFunc(RenderFunction);

glutIdleFunc(RenderFunction);

glutKeyboardFunc(processNormalKeys);

glutSpecialFunc(processSpecialKeys);

glutCloseFunc(Cleanup);

glutMainLoop();

return 0;

}

// Shader-ul de varfuri - GL\_Maze\_Shader.vert

#version 400

layout (location = 0) in vec4 in\_Position; // pozitia este atribut standard

layout (location = 1) in vec3 in\_Color; // culoarea este atribut instantiat

layout (location = 2) in mat4 modelMatrix; // matricea de transformare este atribut instantiat

layout (location = 3) in vec3 in\_Normal;

out vec4 gl\_Position;

out vec4 ex\_Color;

uniform mat4 matrUmbra;

uniform vec3 lightPos;

uniform vec3 viewPos;

uniform vec3 objectColor;

uniform vec3 lightColor;

uniform mat4 viewMatrix;

uniform mat4 projectionMatrix;

uniform int codCol;

void main(void)

{

gl\_Position = projectionMatrix\*viewMatrix\*modelMatrix\*in\_Position;

vec3 Normal=mat3(projectionMatrix\*viewMatrix\*modelMatrix)\*in\_Normal;

vec3 inLightPos= vec3(projectionMatrix\*viewMatrix\*modelMatrix\* vec4(lightPos, 1.0f));

vec3 inViewPos=vec3(projectionMatrix\*viewMatrix\*modelMatrix\*vec4(viewPos, 1.0f));

vec3 FragPos = vec3(gl\_Position);

// Ambient

float ambientStrength = 0.2f;

vec3 ambient = ambientStrength \* lightColor;

// Diffuse

vec3 norm = normalize(Normal);

vec3 lightDir = normalize(inLightPos - FragPos);

// vec3 lightDir = normalize(-inLightPos); // pentru sursa directionala

float diff = max(dot(norm, lightDir), 0.0);

// vec3 diffuse = diff \* lightColor;

vec3 diffuse = pow(diff,0.2) \* lightColor; // varianta de atenuare

// Specular

float specularStrength = 0.5f;

vec3 viewDir = normalize(inViewPos - FragPos);

vec3 reflectDir = reflect(-lightDir, norm);

float spec = pow(max(dot(viewDir, reflectDir), 0.0), 1);

vec3 specular = specularStrength \* spec \* lightColor;

vec3 result = (ambient + diffuse ) \* in\_Color;

ex\_Color = vec4(result, 1.0f);

}

// Shader-ul de fragment / Fragment shader - GL\_Maze\_Shader.frag

#version 400

in vec4 ex\_Color;

uniform int codCol;

out vec4 out\_Color;

void main(void)

{

switch (codCol)

{

case 1: out\_Color=vec4(0.0, 0.0, 0.0,0.0); break;

case 2: out\_Color=vec4(0.5, 0.8, 0.9, 0.9); break;

case 3: out\_Color=vec4(0.65, 0.8, 0.9, 0.9); break;

default: out\_Color=ex\_Color;

}

}

**Bibliografie**

1. **Suportul de curs și codurile sursă din cadrul materiei „Grafică pe calculator”, profesor Sorin Stupariu**