Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

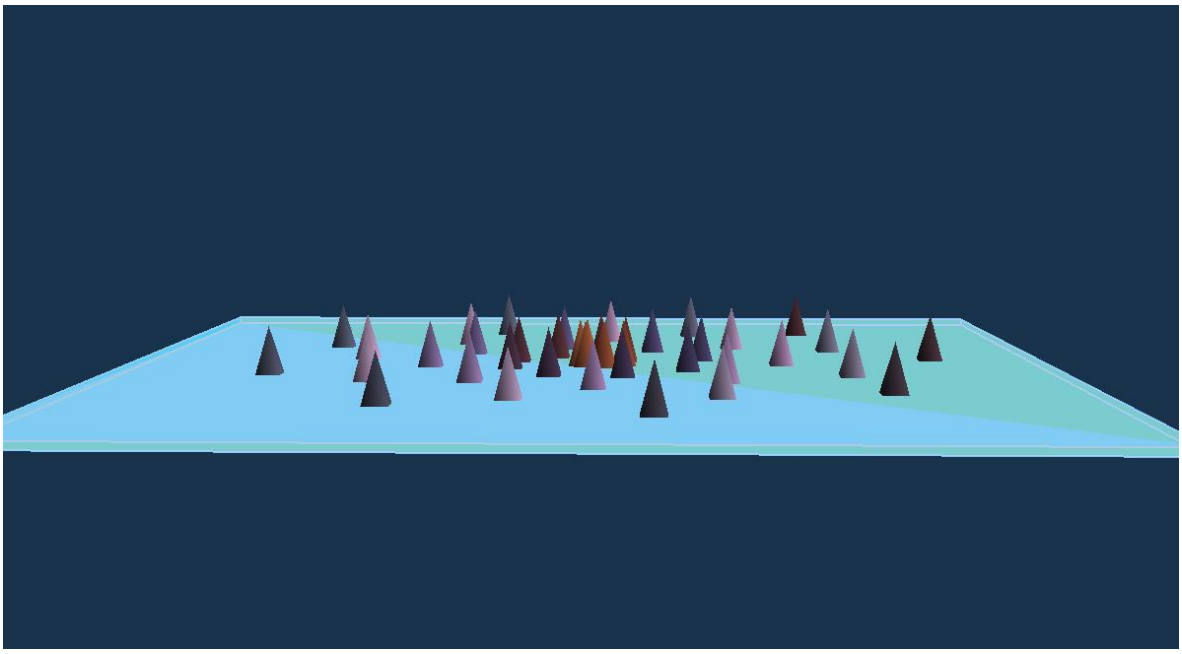
GL MAZE – Proiect2 Documentatie

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:

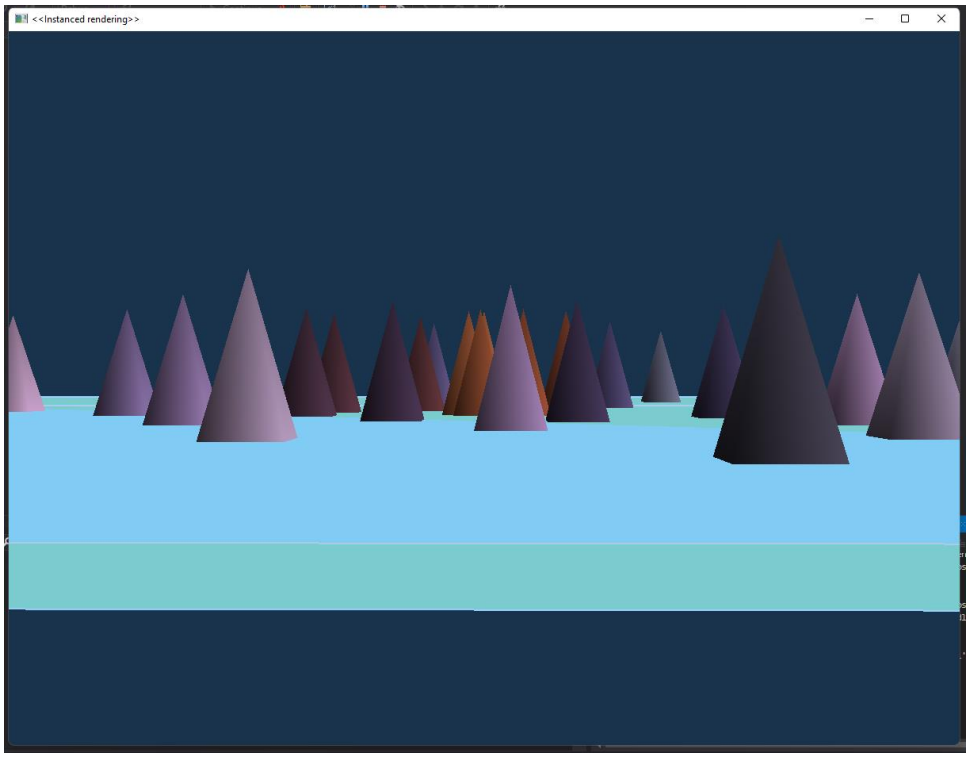


Barbu Iulia-Andreea – Grupa 331

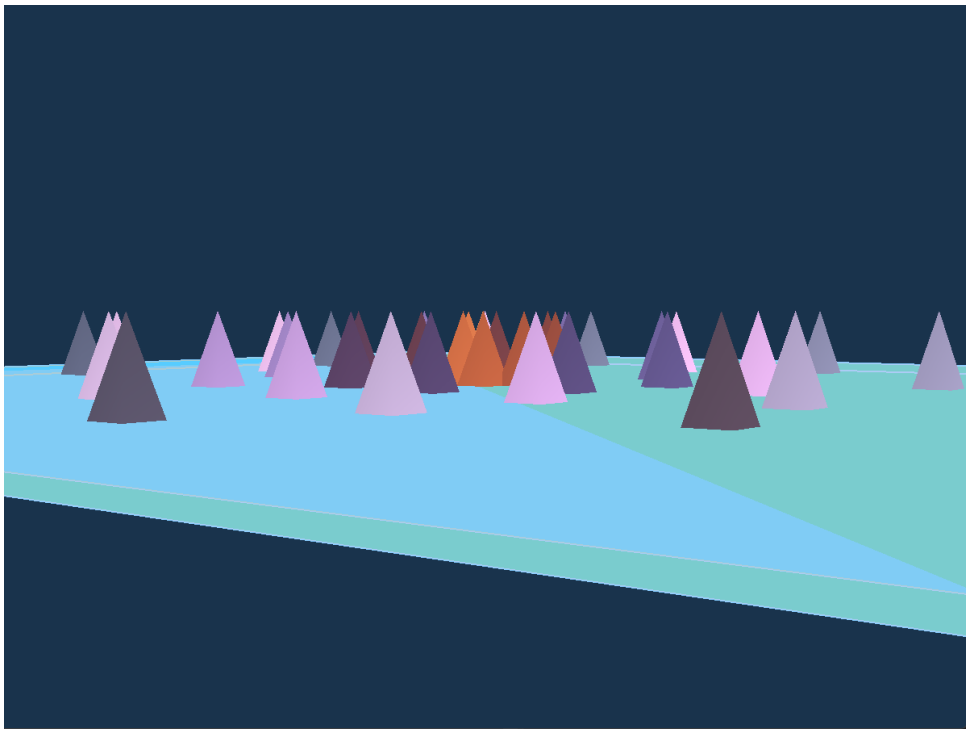
Staicu Octavian-Florin – Grupa 331

2&3. Iluminare&Umbre

Cand lumina bate din dreapta:



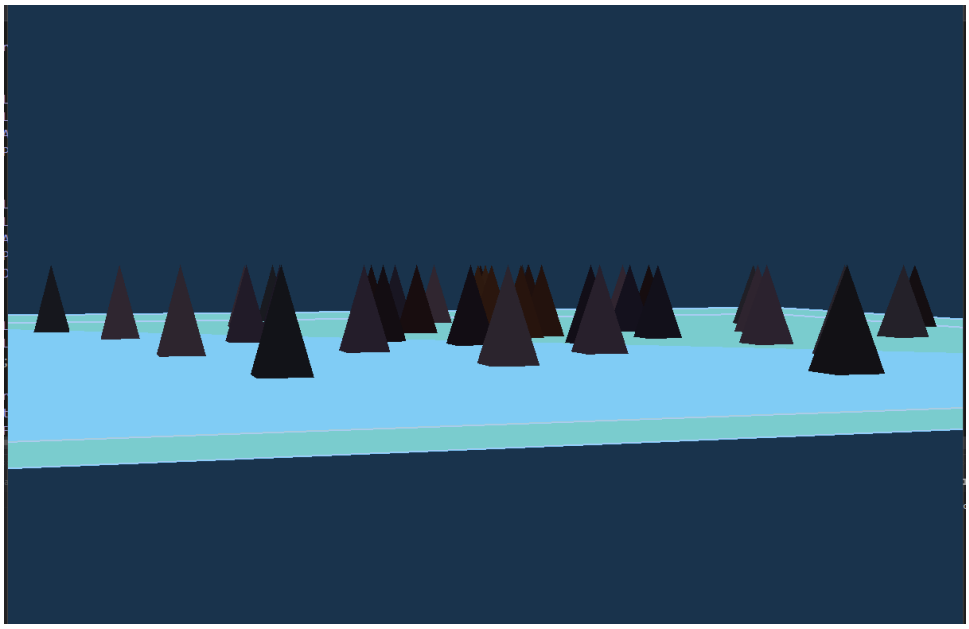
Cand lumina bate din fata



Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

Cand lumina bate din spate:



- In Render Function:

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

```
// 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]);
```

Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

In Shaderul de GL_Maze_Shader.vert:

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

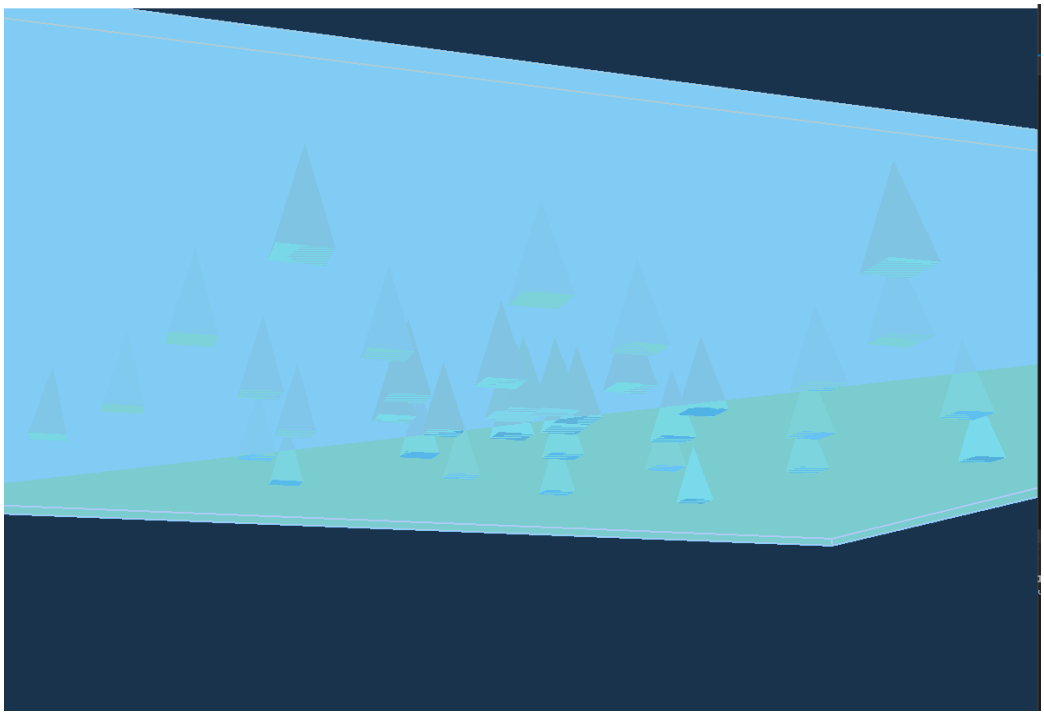
GL_Maze.cpp | GL_Maze_Shader.vert | GL_Maze_Shader.frag

```
19  uniform int codCol;
20
21  void main(void)
22  {
23
24      gl_Position = projectionMatrix*viewMatrix*modelMatrix*in_Position;
25      vec3 Normal=mat3(projectionMatrix*viewMatrix*modelMatrix)*in_Normal;
26      vec3 inLightPos= vec3(projectionMatrix*viewMatrix*modelMatrix* vec4(lightPos, 1.0f));
27      vec3 inViewPos=vec3(projectionMatrix*viewMatrix*modelMatrix*vec4(viewPos, 1.0f));
28      vec3 FragPos = vec3(gl_Position);
29
30      // Ambient
31      float ambientStrength = 0.5f;
32      vec3 ambient = ambientStrength * lightColor;
33
34      // Diffuse
35      vec3 norm = normalize(Normal);
36      vec3 lightDir = normalize(inLightPos - FragPos);
37      // vec3 lightDir = normalize(-inLightPos); // pentru sursa directionala
38      float diff = max(dot(norm, lightDir), 0.0);
39      // vec3 diffuse = diff * lightColor;
40      vec3 diffuse = pow(diff,0.2) * lightColor; // varianta de atenuare
41
42      // Specular
43      float specularStrength = 0.7f;
44      vec3 viewDir = normalize(inViewPos - FragPos);
45      vec3 reflectDir = reflect(-lightDir, norm);
46      float spec = pow(max(dot(viewDir, reflectDir), 0.0), 1);
47      vec3 specular = specularStrength * spec * lightColor;
48
49      vec3 result = (ambient + diffuse ) * in_Color;
50      ex_Color = vec4(result, 1.0f);
51  }
52
```

Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

4. 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).

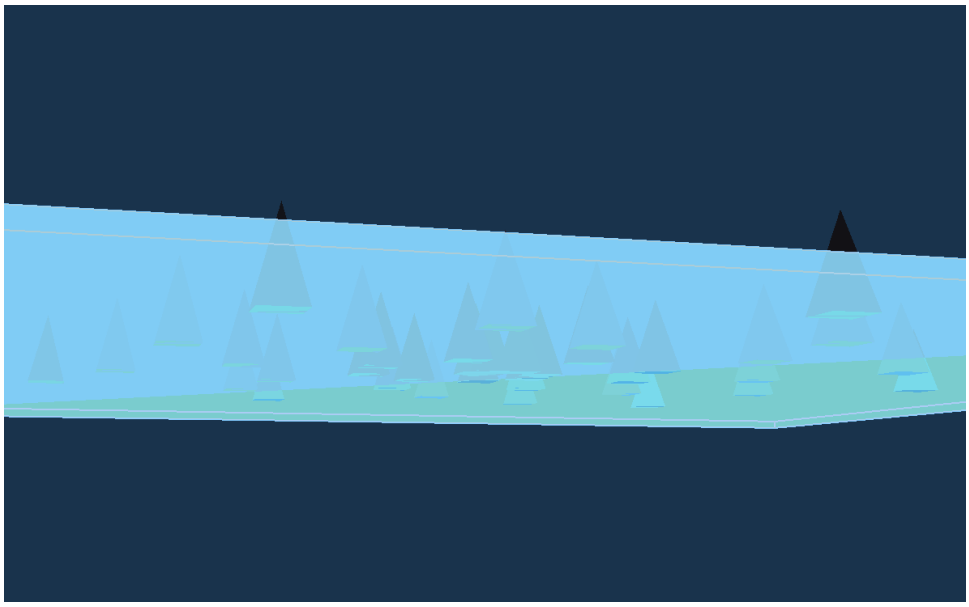
```
339     // Paralelipipedul
340
341     glEnable(GL_BLEND);
342     glDepthMask(GL_FALSE);
343     glBlendFunc(GL_SRC_COLOR, GL_ONE_MINUS_DST_COLOR); // de testat alte v
344
345     // Fetele
346     codCol = 2;
347     glUniform1i(codColLocation, codCol);
348     glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(30));
349     // Muchiile
350     codCol = 3;
351     glUniform1i(codColLocation, codCol);
352     glLineWidth(1.5);
353     glDrawElements(GL_LINE_LOOP, 4, GL_UNSIGNED_BYTE, (void*)(66));
354     glDrawElements(GL_LINE_LOOP, 4, GL_UNSIGNED_BYTE, (void*)(70));
355     glDrawElements(GL_LINES, 8, GL_UNSIGNED_BYTE, (void*)(74));
356
357     glDepthMask(GL_TRUE);
358     glDisable(GL_BLEND);
```

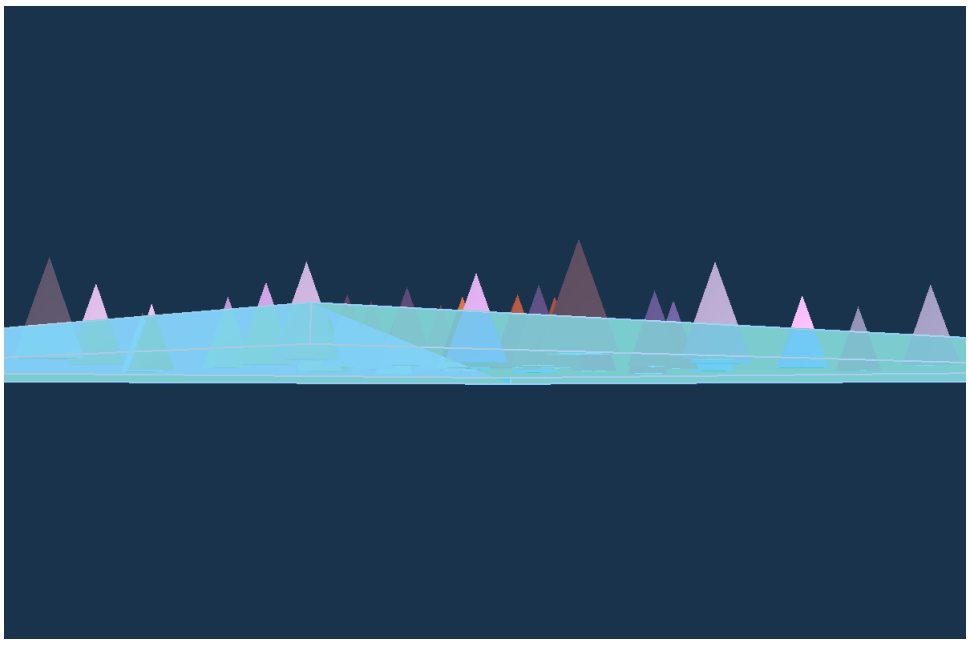
Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

De ce este original?

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

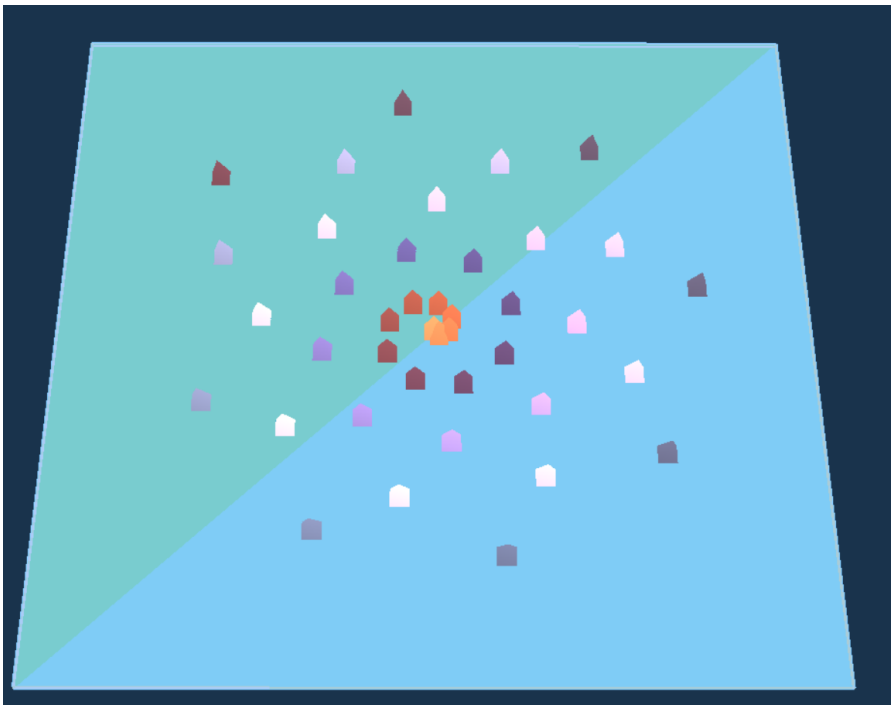




Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

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:

```
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,
```

Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

```
// 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
};
```

Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

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:

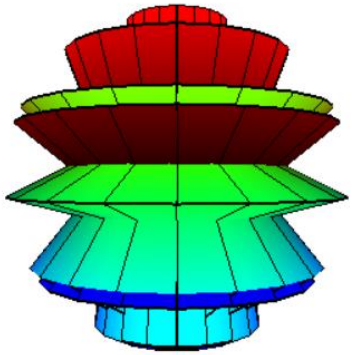
- Implementarea podelei transparente pe care se situează scena
- Implementarea modelului de iluminare
- Testing

Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

Idei neutilizate în proiectul final:

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



```
glm::vec4 Vertices_Center[(NR_PARR + 1) * NR_MERID];
glm::vec3 Colors_Center[(NR_PARR + 1) * NR_MERID];
GLushort Indices_Center[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PARR + 1) * NR_MERID];
for (int merid = 0; merid < NR_MERID; merid++)
{
    for (int parr = 0; parr < NR_PARR + 1; parr++)
    {
        // implementarea reprezentarii parametrice
        float u = U_MIN + parr * step_u; // valori pentru u si v
        float v = V_MIN + merid * step_v;
        float x_vf = radius * cosf(u) * cosf(v); // coordonatele varfului corespunzator lui (u,v)
        float y_vf = radius * cosf(u) * sinf(v);
        float z_vf = radius * sinf(u);
        // identificator ptr varf; coordonate + culoare + indice la parcurgerea meridianelor
        index = merid * (NR_PARR + 1) + parr;
        Vertices_Center[index] = glm::vec4(x_vf, y_vf, z_vf, 1.0);
        Colors_Center[index] = glm::vec3(0.1f + sinf(u), 0.1f + cosf(v), 0.1f + -1.5 * sinf(u));
        Indices_Center[index] = index;
        // indice ptr acelasi varf la parcurgerea paralelelor
        index_aux = parr * (NR_MERID)+merid;
        Indices_Center[(NR_PARR + 1) * NR_MERID + index_aux] = index;
        // indicii pentru desenarea fetelor, pentru varful curent sunt definite 4 varfuri
        if ((parr + 1) % (NR_PARR + 1) != 0) // varful considerat sa nu fie Polul Nord
        {
            int AUX = 2 * (NR_PARR + 1) * NR_MERID;
            int index1 = index; // varful v considerat
            int index2 = index + (NR_PARR + 1); // dreapta lui v, pe meridianul urmator
            int index3 = index2 + 1;  // dreapta sus fata de v
            int index4 = index + 1;  // deasupra lui v, pe acelasi meridian
            if (merid == NR_MERID - 1)  // la ultimul meridian, trebuie revenit la meridianul initial
            {
                index2 = index2 % (NR_PARR + 1);
                index3 = index3 % (NR_PARR + 1);
            }
            Indices_Center[AUX + 4 * index] = index1;  // unele valori ale lui Indices, corespunzatoare Polului Nord, au valori neadecvate
            Indices_Center[AUX + 4 * index + 1] = index2;
            Indices_Center[AUX + 4 * index + 2] = index3;
            Indices_Center[AUX + 4 * index + 3] = index4;
        }
    }
};
```

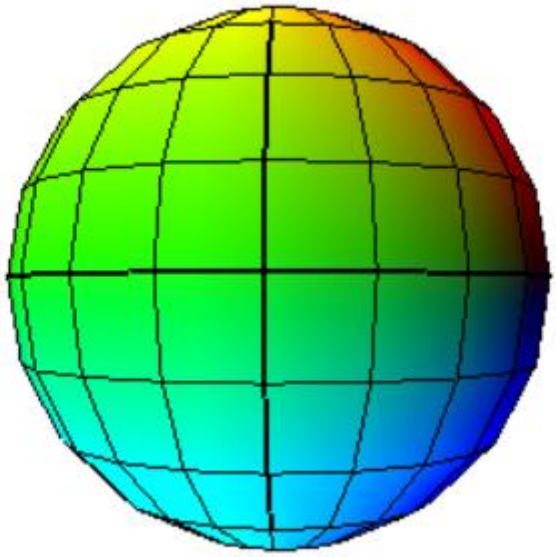
Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

- 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 jocuțului ar fi fost ca mingea să poată ajunge la sculptura centrală fără să lovească piramidele
-



Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

// 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;
```

Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

```
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);
}
```

Barbu Iulia-Andreea – Grupa 331
Staicu Octavian-Florin – Grupa 331

```
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;
}
```

Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

```
// 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);
 }
```

Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

```
// 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;
    }
  }
```

Barbu Iulia-Andreea – Grupa 331

Staicu Octavian-Florin – Grupa 331

Bibliografie

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