Grafică pe Calculator - Proiect 2

Simularea grafică a unui peisaj de iarnă

Membrii echipei: Dilirici Mihai, Florea Mădălin-Alexandru, Nechita Maria-Ilinca

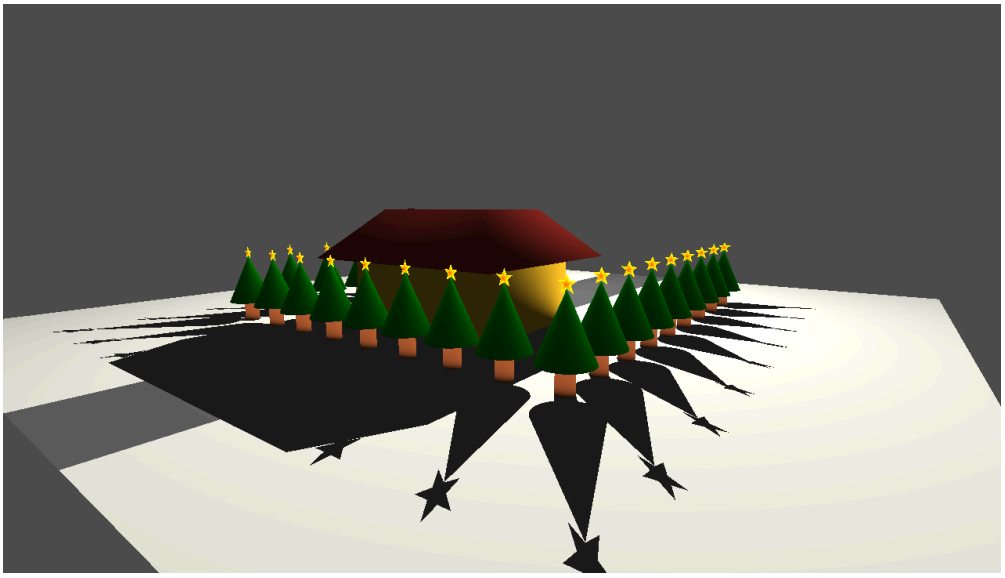
Grupa: 343

Cuprinsul documentației:

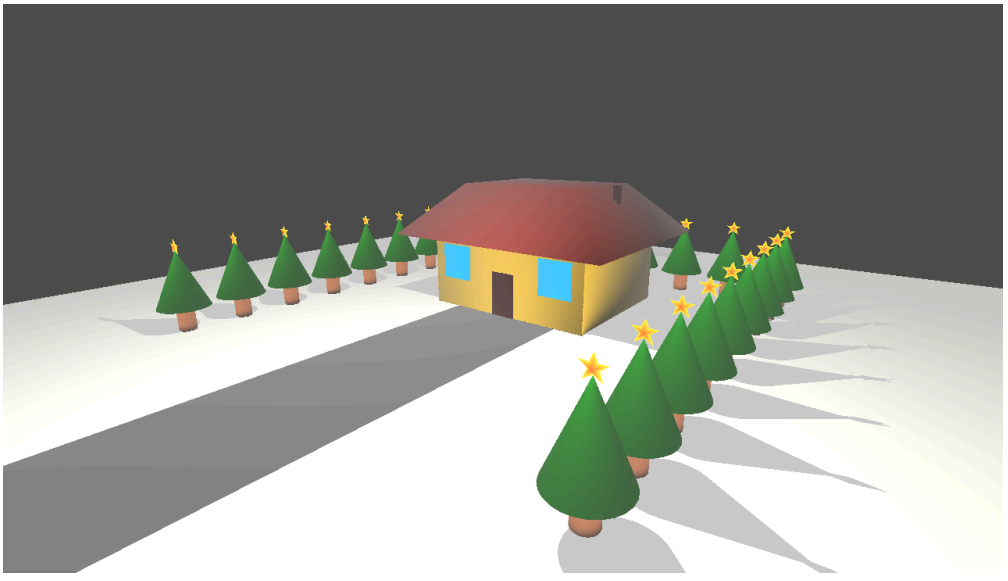
1. Conceptul proiectului

- Proiectul dezvoltat de noi simulează un peisaj de iarnă în mod cât mai realist folosind figuri 3D. Scena conține o casă aflată într-o curte delimitată de brazi, efecte vizuale precum iluminare, umbre și ceață și un observator ce se poate deplasa pe cele 3 axe de coordonate și poate privi scena de la depărtare sau se poate apropia. Scena poate fi vizualizată în imaginile de mai jos:

  Fără efectul de ceață:



Cu ceață:



2. Elemente incluse

- Reprezentare obiecte 3D: pentru a desena elementele din peisaj, ne-am folosit de mai multe obiecte 3D, după cum urmează:
  - ❖ Bradul este compus din:
    - ➢ Trunchi (cilindru) - definit în funcția *CreateVAOTreeTrunk*
    - ➢ Coroană (con) - definit în funcția *CreateVAOTreeCrown*
    - ➢ Stea - figură complexă definită în funcția *CreateVAOTreeStar*
  - ❖ Casa este compusă din:
    - ➢ Pereți, podea, tavan (paralelipiped dreptunghic)
    - ➢ Acoperiș (trunchi de piramidă patrulateră)
    - ➢ Horn (paralelipiped dreptunghic)
    - ➢ Ușă, ferestre (patrulatere 2D)

  Toate aceste elemente sunt definite în funcția *CreateVAOHouse.*

- Iluminare: Asupra elementelor din peisaj am aplicat iluminarea, sursa de lumină creată fiind poziționată în fața casei, la înălțime. Coordonatele acesteia sunt reținute în variabilele

  float xL = 400.0f, yL = 0.0f, zL = 850.0f

  De asemenea, normalele pentru fiecare figură 3D din scenă indică fața poligoanelor desenate și sunt necesare pentru calculul iluminării pentru a determina cum este afectat un obiect de sursele de lumină. În shaderele noastre, ele sunt implicate în calculul componentelor difuze și specular ale iluminării.

- Ceață: De asemenea, am implementat efectul de ceață pentru peisajul din scena noastră. Astfel, cu cât obiectele sunt situate la o distanță mai mare față de observator, culoarea acestora este amestecată cu o nuanță de gri deschis. Acest lucru se realizează în shaderul de fragment, prin următoarele linii de cod:

```
float fogDistance = length(inViewPos - FragPos);
float fogAmount = smoothstep(500.0f, 3500.0f, fogDistance);
if (codCol == 0) {
   ...
   result = mix(result, fogColor, fogAmount);  // adaugam efectul de ceata
}
if (codCol == 1) {
   ...
   vec3 result = mix(shadowColor, fogColor, fogAmount);
}
```

- Umbre: Am definit o matrice utilizată în cadrul tehnicii de umbrire pentru a calcula umbra produsă de o sursă de lumină asupra unui obiect - acesta este procesul de umbrire, care transformă coordonatele obiectelor în spațiul umbrei. Pentru a desena umbra unui obiect și nu obiectul în sine, variabila uniformă *codCol* trebuie să aibă valoarea 1.

```
//  matricea pentru umbra
float D = -3.0;
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]);
```

3. Originalitatea scenei 3D

- Cel mai complex și original element din scena noastră este reprezentat de steaua din vârful bradului. Aceasta a fost realizată prin alegerea câtor 5 puncte aflate pe 2 cercuri concentrice de dimensiuni diferite, cu centrul în „mijlocul” stelei, și a altor 2 puncte pe axa verticală, coliniare cu mijlocul stelei, situate la distanțe egale în fața și în spatele acestuia. După definirea acestor puncte, am desenat toate fețele triunghiulare ale stelei, unind câte un punct de pe cercul mare cu punctul corespunzător de pe cercul mic și cu unul dintre cele 2 centre.

4. Modificări aduse în plus după discuțiile de la laborator

- Comparativ cu discuțiile din cadrul laboratorului, a fost modificat modul în care lumina este reflectată pe pereții casei, acum fiind mult mai realistă.

5. Contribuții personale

- Proiectului a fost realizat prin intermediul mai multor întâlniri online unde au participat toți membri echipei, procesul de proiectare și dezvoltare fiind unul exclusiv colectiv (nu am lucrat decât împreună).

6. Coduri sursă

- **Shader.frag:**

```
// ===========================================
// |          Grafica pe calculator          |
```

```
// ================================================
// |          PROIECT 2 - Shader.frag          |
// ================================================
//
//  Shaderul de fragment / Fragment shader - afecteaza culoarea pixelilor;

#version 330 core

in vec3 FragPos;
in vec3 Normal;
in vec3 inLightPos;
in vec3 inViewPos;
in vec3 dir;
in vec3 ex_Color;

out vec4 out_Color;

uniform vec3 lightColor;
uniform vec3 fogColor;
uniform int codCol;

void main(void) {
   // fog
   float fogDistance = length(inViewPos - FragPos);
   float fogAmount = smoothstep(500.0f, 3500.0f, fogDistance);

   // pentru codCol == 0 este aplicata iluminarea
   if (codCol == 0) {
         // Ambient
      float ambientStrength = 0.2f;
      vec3 ambient = ambientStrength * lightColor;

      // Diffuse
      vec3 normala = normalize(Normal);
      vec3 lightDir = normalize(inLightPos - FragPos);
      float diff = max(dot(normala, lightDir), 0.0);
      vec3 diffuse = diff * lightColor;

      // Specular
      float specularStrength = 0.1f;
```

```
        vec3 viewDir = normalize(inViewPos - FragPos);  //vector catre observator normalizat (V)
        vec3 reflectDir = reflect(-lightDir, normala);  // reflexia razei de lumina (R)
        float spec = pow(max(dot(viewDir, reflectDir), 0.0), 1);
        vec3 specular = specularStrength * spec * lightColor;
        vec3 emission = vec3(0.0, 0.0, 0.0);
        vec3 result = emission + (ambient + diffuse + specular) * ex_Color;

        result = mix(result, fogColor, fogAmount);  // adaugam efectul de ceata

            out_Color = vec4(result, 1.0f);
    }

    // pentru codCol == 1 este desenata umbra
    if (codCol == 1) {
        vec3 shadowColor = vec3(0.1, 0.1, 0.1);

        // without fog
        out_Color = vec4(shadowColor, 1.0f);

        // with fog
        vec3 result = mix(shadowColor, fogColor, fogAmount);
        out_Color = vec4(result, 1.0);
    }
}
```

- **Shader.vert:**

```
// ==================================================
// |          Grafica pe calculator          |
// ==================================================
// |          PROIECT 2 - Shader.vert          |
// ==================================================
//
//  Shaderul de varfuri / Vertex shader - afecteaza geometria scenei;

#version 330 core

layout(location=0) in vec4 in_Position;
layout(location=1) in vec3 in_Color;
layout(location=2) in vec3 in_Normal;
```

```
out vec3 FragPos;
out vec3 Normal;
out vec3 inLightPos;
out vec3 inViewPos;
out vec3 ex_Color;
out vec3 dir;

uniform mat4 matrUmbra;
uniform mat4 myMatrix;
uniform mat4 view;
uniform mat4 projection;
uniform vec3 lightPos;
uniform vec3 viewPos;
uniform vec3 lightColor;
uniform int codCol;

void main(void) {
   ex_Color = in_Color;

        if (codCol == 0) {
                gl_Position = projection * view * myMatrix * in_Position;
      Normal = mat3(projection * view * myMatrix) * in_Normal;
      inLightPos = vec3(projection * view * myMatrix * vec4(lightPos, 1.0f));
      inViewPos = vec3(projection * view * myMatrix * vec4(viewPos, 1.0f));
         dir = mat3(projection * view * myMatrix) * vec3(0.0, 100.0, 200.0); // pentru
sursa directionala
   }
        if (codCol == 1)
                gl_Position = projection * view * matrUmbra * myMatrix * in_Position;

   FragPos = vec3(gl_Position);
}
```

- **Proiect.cpp:**

```
// ===================================================
// |          Grafica pe calculator           |
// ===================================================
// |        PROIECT 2 - Peisaj de iarna        |
// ===================================================
```

```
//	Biblioteci
#include <iostream>	// Biblioteca standard pentru operatii de intrare/iesire (e.g. std::cout)
#include <windows.h>	// Utilizarea functiilor de sistem Windows (crearea de ferestre, manipularea fisierelor si directoarelor);
#include <stdlib.h>	// Biblioteci necesare pentru citirea shaderelor;
#include <stdio.h>
#include <math.h>	// Biblioteca pentru calcule matematice;
#include <GL/glew.h>	// Defineşte prototipurile functiilor OpenGL si constantele necesare pentru programarea OpenGL moderna;
#include <GL/freeglut.h>	// Include functii pentru:
//	- gestionarea ferestrelor si evenimentelor de tastatura si mouse,
//	- desenarea de primitive grafice precum dreptunghiuri, cercuri sau linii,
//	- crearea de meniuri si submeniuri;
#include "loadShaders.h"	// Fisierul care face legatura intre program si shadere;
#include "glm/glm.hpp"	// Bibloteci utilizate pentru transformari grafice;
#include "glm/gtc/matrix_transform.hpp"
#include "glm/gtx/transform.hpp"
#include "glm/gtc/type_ptr.hpp"


//  Identificatorii obiectelor de tip OpenGL
GLuint
VaoGround, VboGround, EboGround,
VaoTreeTrunk, VboTreeTrunk, EboTreeTrunk,
VaoTreeCrown, VboTreeCrown, EboTreeCrown,
VaoTreeStar, VboTreeStar, EboTreeStar,
VaoHouse, VboHouse, EboHouse,

ColorBufferId,
ProgramId,
myMatrixLocation,
```

```
matrUmbraLocation,
viewLocation,
projLocation,
matrRotlLocation,
lightColorLocation,
lightPosLocation,
fogColorLocation,
viewPosLocation,
codColLocation;

int codCol;

// matrice utilizate
glm::mat4 myMatrix;


// valoarea lui PI
float const PI = 3.141592f;


// elemente pentru matricea de vizualizare
float Refx = 0.0f, Refy = 0.0f, Refz = 100.0f;
float alpha = PI / 8, beta = 0.0f, dist = 750.0f;
float Obsx, Obsy, Obsz;
float Vx = 0.0, Vy = 0.0, Vz = 1.0;
glm::mat4 view;

// elemente pentru matricea de proiectie
float width = 1280, height = 730, xwmin = -800.f, xwmax = 800, ywmin = -600,
ywmax = 600, znear = 0.1, zfar = 1, fov = 45;
glm::mat4 projection;

const GLfloat winWidth = 1280.0f, winHeight = 730.0f;

// sursa de lumina
float xL = 400.0f, yL = 0.0f, zL = 850.0f;

// matricea umbrei
float matrUmbra[4][4];
```

```
// Elemente pentru reprezentarea cilindrului (trunchiul bradului)
const int NR_MERIDIANE_CILINDRU = 20, RAZA_CILINDRU = 25, INALTIME_CILINDRU = 75;

// Elemente pentru reprezentarea cilindrului (coroana bradului)
const int NR_MERIDIANE_CON = 25, RAZA_CON = 75, INALTIME_CON = 175;


void processNormalKeys(unsigned char key, int x, int y) {
	switch (key)
	{
	case '+':
		dist += 10.0;
		break;
	case '-':
		dist -= 10.0;
		break;
	case 'w':
		Refx -= 10.0;
		break;
	case 'a':
		Refy -= 10.0;
		break;
	case 's':
		Refx += 10.0;
		break;
	case 'd':
		Refy += 10.0;
		break;
	}

	if (key == 27)
		exit(0);
}


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

```
		case GLUT_KEY_LEFT:
			beta -= 0.03;
			break;
		case GLUT_KEY_RIGHT:
			beta += 0.03;
			break;
		case GLUT_KEY_UP:
			alpha += 0.03;
			break;
		case GLUT_KEY_DOWN:
			alpha -= 0.03;
			break;
		}
}


// GROUND
void CreateVAOGround(void) {
		GLfloat Vertices[] = {
			// GRADINA DIN STANGA
					// coordonate		// culori		// normale
		  -1500.0f, -1500.0f,  0.0f, 1.0f,  1.0f, 1.0f, 0.95f,  0.0f, 0.0f, 1.0f,		//
stanga sus (mai departe de observator)
			1500.0f, -1500.0f,  0.0f, 1.0f,  1.0f, 1.0f, 0.95f,  0.0f, 0.0f, 1.0f,
		// stanga jos (mai aproape de observator)
			1500.0f,  -200.0f,  0.0f, 1.0f,  1.0f, 1.0f, 0.95f,  0.0f, 0.0f, 1.0f,		//
dreapta jos (mai aproape de observator)
		  -1500.0f,  -200.0f,  0.0f, 1.0f,  1.0f, 1.0f, 0.95f,  0.0f, 0.0f, 1.0f,		//
dreapta sus (mai departe de observator)

		  // ALEEA DIN MIJLOC
					// coordonate		// culori		// normale
		  -1500.0f,  -200.0f,  0.0f, 1.0f,  0.4f, 0.4f,  0.4f,  0.0f, 0.0f, 1.0f,		//
stanga sus (mai departe de observator)
			1500.0f,  -200.0f,  0.0f, 1.0f,  0.4f, 0.4f,  0.4f,  0.0f, 0.0f, 1.0f,		//
stanga jos (mai aproape de observator)
			1500.0f,   200.0f,  0.0f, 1.0f,  0.4f, 0.4f,  0.4f,  0.0f, 0.0f, 1.0f,		//
dreapta jos (mai aproape de observator)
		  -1500.0f,   200.0f,  0.0f, 1.0f,  0.4f, 0.4f,  0.4f,  0.0f, 0.0f, 1.0f,		//
dreapta sus (mai departe de observator)
```

```
        // GRADINA DIN DREAPTA
                                // coordonate            // culori                // normale
        -1500.0f,   200.0f,  0.0f, 1.0f,  1.0f, 1.0f,  0.95f,  0.0f, 0.0f, 1.0f,              //
stanga sus (mai departe de observator)
                1500.0f,   200.0f,  0.0f, 1.0f,  1.0f, 1.0f,  0.95f,  0.0f, 0.0f, 1.0f,
        // stanga jos (mai aproape de observator)
                1500.0f,  1500.0f,  0.0f, 1.0f,  1.0f, 1.0f,  0.95f,  0.0f, 0.0f, 1.0f,
// dreapta jos (mai aproape de observator)
        -1500.0f,  1500.0f,  0.0f, 1.0f,  1.0f, 1.0f,  0.95f,  0.0f, 0.0f, 1.0f,              //
dreapta sus (mai departe de observator)
        };

        GLubyte Indices[] = {
                0, 1,  2,  0,  2,  3,  // gradina din stanga
                4, 5,  6,  4,  6,  7,  // aleea din mijloc
                8, 9, 10,  8, 10, 11   // gradina din dreapta
        };

        glGenVertexArrays(1, &VaoGround);
        glBindVertexArray(VaoGround);

        glGenBuffers(1, &VboGround);
        glGenBuffers(1, &EboGround);

        glBindBuffer(GL_ARRAY_BUFFER, VboGround);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboGround);

        glBufferData(GL_ARRAY_BUFFER,           sizeof(Vertices),           Vertices,
GL_STATIC_DRAW);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER,   sizeof(Indices),    Indices,
GL_STATIC_DRAW);

        // atributul 0 = pozitie
        glEnableVertexAttribArray(0);
        glVertexAttribPointer(0,  4,  GL_FLOAT,  GL_FALSE,  10  *  sizeof(GLfloat),
(GLvoid*)0);
        // atributul 1 = culoare
        glEnableVertexAttribArray(1);
```

```
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(4 * sizeof(GLfloat)));
	// atributul 2 = normale
	glEnableVertexAttribArray(2);
	glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(7 * sizeof(GLfloat)));
}


// TRUNCHIUL BRADULUI
void CreateVAOTreeTrunk(void) {
	GLfloat Vertices[10 * (NR_MERIDIANE_CILINDRU * 2)];
	GLubyte Indices[6 * NR_MERIDIANE_CILINDRU];

	for (int merid = 0; merid < NR_MERIDIANE_CILINDRU; ++merid)
	{
		float u = 2 * PI * merid / NR_MERIDIANE_CILINDRU;
		float x_vf = RAZA_CILINDRU * cosf(u);
		float y_vf = RAZA_CILINDRU * sinf(u);


		// baza inferioara

		// coordonatele varfului
		Vertices[10 * merid] = x_vf;
		Vertices[10 * merid + 1] = y_vf;
		Vertices[10 * merid + 2] = 0.0;
		Vertices[10 * merid + 3] = 1.0;

		// culoarea varfului
		Vertices[10 * merid + 4] = 0.627f;
		Vertices[10 * merid + 5] = 0.321f;
		Vertices[10 * merid + 6] = 0.168f;

		// normalele varfului
		Vertices[10 * merid + 7] = x_vf;
		Vertices[10 * merid + 8] = y_vf;
		Vertices[10 * merid + 9] = 0.0;

		// indicii pentru baza inferioara
```

```
            Indices[merid] = merid;


            // baza superioara

            // coordonatele varfului
            Vertices[10 * (NR_MERIDIANE_CILINDRU + merid)] = x_vf;
            Vertices[10 * (NR_MERIDIANE_CILINDRU + merid) + 1] = y_vf;
            Vertices[10 * (NR_MERIDIANE_CILINDRU + merid) + 2] =
INALTIME_CILINDRU;
            Vertices[10 * (NR_MERIDIANE_CILINDRU + merid) + 3] = 1.0;

            // culoarea varfului
            Vertices[10 * (NR_MERIDIANE_CILINDRU + merid) + 4] = 0.627f;
            Vertices[10 * (NR_MERIDIANE_CILINDRU + merid) + 5] = 0.321f;
            Vertices[10 * (NR_MERIDIANE_CILINDRU + merid) + 6] = 0.168f;

            // normalele varfului
            Vertices[10 * (NR_MERIDIANE_CILINDRU + merid) + 7] = x_vf;
            Vertices[10 * (NR_MERIDIANE_CILINDRU + merid) + 8] = y_vf;
            Vertices[10 * (NR_MERIDIANE_CILINDRU + merid) + 9] =
INALTIME_CILINDRU;

            // indicii pentru baza superioara
            Indices[NR_MERIDIANE_CILINDRU + merid] =
NR_MERIDIANE_CILINDRU + merid;

            // indicii pentru fetele laterale
            Indices[2 * NR_MERIDIANE_CILINDRU + 4 * merid] = merid;
            Indices[2 * NR_MERIDIANE_CILINDRU + 4 * merid + 1] =
NR_MERIDIANE_CILINDRU + merid;
            Indices[2 * NR_MERIDIANE_CILINDRU + 4 * merid + 2] =
NR_MERIDIANE_CILINDRU + (merid + 1) % NR_MERIDIANE_CILINDRU;
            Indices[2 * NR_MERIDIANE_CILINDRU + 4 * merid + 3] = (merid +
1) % NR_MERIDIANE_CILINDRU;
        };


        glGenVertexArrays(1, &VaoTreeTrunk);
        glBindVertexArray(VaoTreeTrunk);
```

```
	glGenBuffers(1, &VboTreeTrunk);
	glGenBuffers(1, &EboTreeTrunk);

	glBindBuffer(GL_ARRAY_BUFFER, VboTreeTrunk);
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboTreeTrunk);

	glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices,
GL_STATIC_DRAW);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices,
GL_STATIC_DRAW);

	// atributul 0 = pozitie
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat),
(GLvoid*)0);
	// atributul 1 = culoare
	glEnableVertexAttribArray(1);
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat),
(GLvoid*)(4 * sizeof(GLfloat)));
	// atributul 2 = normale
	glEnableVertexAttribArray(2);
	glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat),
(GLvoid*)(7 * sizeof(GLfloat)));
}


// COROANA BRADULUI
void CreateVAOTreeCrown(void) {
	GLfloat Vertices[10 * (NR_MERIDIANE_CON + 1)];
	GLubyte Indices[4 * NR_MERIDIANE_CON];

	// Se creeaza baza conului (un cerc)
	for (int i = 0; i < NR_MERIDIANE_CON; ++i)
	{
		float u = 2 * PI * i / NR_MERIDIANE_CON;

		float x = RAZA_CON * cos(u);
		float y = RAZA_CON * sin(u);
		float z = INALTIME_CILINDRU;
```

```
            // coordonatele varfului
            Vertices[10 * i] = x;
            Vertices[10 * i + 1] = y;
            Vertices[10 * i + 2] = z;
            Vertices[10 * i + 3] = 1.0;

            // culoarea varfului
            Vertices[10 * i + 4] = 0.0;
            Vertices[10 * i + 5] = 0.4;
            Vertices[10 * i + 6] = 0.0;

            // normalele varfului
            Vertices[10 * i + 7] = x;
            Vertices[10 * i + 8] = y;
            Vertices[10 * i + 9] = z - 50.0;

            // indicii pentru baza conului
            Indices[i] = i;
        }

        // Se creeaza varful conului

        // coordonatele varfului
        Vertices[10 * NR_MERIDIANE_CON] = 0.0;
        Vertices[10 * NR_MERIDIANE_CON + 1] = 0.0;
        Vertices[10 * NR_MERIDIANE_CON + 2] = INALTIME_CILINDRU +
INALTIME_CON;
        Vertices[10 * NR_MERIDIANE_CON + 3] = 1.0;

        // culoarea varfului
        Vertices[10 * NR_MERIDIANE_CON + 4] = 0.0;
        Vertices[10 * NR_MERIDIANE_CON + 5] = 0.6;
        Vertices[10 * NR_MERIDIANE_CON + 6] = 0.0;

        // normalele varfului
        Vertices[10 * NR_MERIDIANE_CON + 7] = 0.0;
        Vertices[10 * NR_MERIDIANE_CON + 8] = 0.0;
        Vertices[10 * NR_MERIDIANE_CON + 9] = 1.0;
```

```
	// Se creeaza indicii pentru fetele conului
	for (int i = 0; i < NR_MERIDIANE_CON; ++i)
	{
		Indices[NR_MERIDIANE_CON + 3 * i] = NR_MERIDIANE_CON;
		Indices[NR_MERIDIANE_CON + 3 * i + 1] = i;
		Indices[NR_MERIDIANE_CON + 3 * i + 2] = (i + 1) % NR_MERIDIANE_CON;
	}


	glGenVertexArrays(1, &VaoTreeCrown);
	glBindVertexArray(VaoTreeCrown);

	glGenBuffers(1, &VboTreeCrown);
	glGenBuffers(1, &EboTreeCrown);

	glBindBuffer(GL_ARRAY_BUFFER, VboTreeCrown);
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboTreeCrown);

	glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices, GL_STATIC_DRAW);

	// atributul 0 = pozitie
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)0);
	// atributul 1 = culoare
	glEnableVertexAttribArray(1);
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(4 * sizeof(GLfloat)));
	// atributul 2 = normale
	glEnableVertexAttribArray(2);
	glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(7 * sizeof(GLfloat)));
}


// STEAUA DIN VARFUL BRADULUI
```

```
void CreateVAOTreeStar(void) {
	GLfloat Vertices[] = {
		// CERCUL MIC
				// coordonate		// culori			// normale
		 0.0f,   0.0f, 250.0f, 1.0f,   1.0f, 1.0f, 0.0f,    0.0f,   0.0f, 250.0f,
	// jos
		 0.0f,  10.0f, 258.0f, 1.0f,   1.0f, 1.0f, 0.0f,    0.0f,  10.0f, 258.0f,
	// dreapta jos
		 0.0f,   7.0f, 269.0f, 1.0f,   1.0f, 1.0f, 0.0f,    0.0f,   7.0f, 269.0f,
	// dreapta sus
		 0.0f,  -7.0f, 269.0f, 1.0f,   1.0f, 1.0f, 0.0f,    0.0f,  -7.0f, 269.0f,
	// stanga sus
		 0.0f, -10.0f, 258.0f, 1.0f,   1.0f, 1.0f, 0.0f,    0.0f, -10.0f, 258.0f,
	// stanga jos

		 // CERCUL MARE
				// coordonate		// culori			// normale
		 0.0f,  17.0f, 240.0f, 1.0f,   1.0f, 1.0f, 0.0f,    0.0f,  17.0f, 240.0f,
	// dreapta jos
		 0.0f,  25.0f, 269.0f, 1.0f,   1.0f, 1.0f, 0.0f,    0.0f,  25.0f, 269.0f,
	// dreapta sus
		 0.0f,   0.0f, 285.0f, 1.0f,   1.0f, 1.0f, 0.0f,    0.0f,   0.0f, 285.0f,
	// sus
		 0.0f, -25.0f, 269.0f, 1.0f,   1.0f, 1.0f, 0.0f,    0.0f, -25.0f, 269.0f,
	// stanga sus
		 0.0f, -17.0f, 240.0f, 1.0f,   1.0f, 1.0f, 0.0f,    0.0f, -17.0f, 240.0f,
	// stanga jos

		// CENTRUL STELEI
				// coordonate		// culori			// normale
		-7.0f,   0.0f, 260.0f, 1.0f,   1.0f, 0.3f, 0.0f,   -7.0f,   0.0f, 260.0f,
	// mijloc spate
		 7.0f,   0.0f, 260.0f, 1.0f,   1.0f, 0.3f, 0.0f,    7.0f,   0.0f, 260.0f,
	// mijloc fata
	};

	GLubyte Indices[] = {
		10, 0, 5, 10, 5, 1,
		10, 1, 6, 10, 6, 2,
		10, 2, 7, 10, 7, 3,
```

```
            10, 3, 8, 10, 8, 4,
            10, 4, 9, 10, 9, 0,

            11, 0, 5, 11, 5, 1,
            11, 1, 6, 11, 6, 2,
            11, 2, 7, 11, 7, 3,
            11, 3, 8, 11, 8, 4,
            11, 4, 9, 11, 9, 0,
        };

        glGenVertexArrays(1, &VaoTreeStar);
        glBindVertexArray(VaoTreeStar);

        glGenBuffers(1, &VboTreeStar);
        glGenBuffers(1, &EboTreeStar);

        glBindBuffer(GL_ARRAY_BUFFER, VboTreeStar);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboTreeStar);

        glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices,
GL_STATIC_DRAW);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices,
GL_STATIC_DRAW);

        // atributul 0 = pozitie
        glEnableVertexAttribArray(0);
        glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat),
(GLvoid*)0);
        // atributul 1 = culoare
        glEnableVertexAttribArray(1);
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat),
(GLvoid*)(4 * sizeof(GLfloat)));
        // atributul 2 = normale
        glEnableVertexAttribArray(2);
        glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat),
(GLvoid*)(7 * sizeof(GLfloat)));
}


// CASA
```

```
void CreateVAOHouse(void) {
	GLfloat Vertices[] = {
		// PERETII CASEI (CUBUL)
		//
		// Baza de jos
				// coordonate		// culori			// normale
		  -500.0f, -300.0f,   0.0f, 1.0f,  0.93f, 0.71f, 0.14f,   -500.0f, -300.0f,
-1.0f,   // stanga sus
		    0.0f, -300.0f,   0.0f, 1.0f,  0.93f, 0.71f, 0.14f,     0.0f, -300.0f,  -1.0f,
// stanga jos
		  0.0f,  300.0f,   0.0f, 1.0f,  0.93f, 0.71f, 0.14f,     0.0f,  300.0f,  -1.0f,   //
dreapta jos
		 -500.0f,  300.0f,   0.0f, 1.0f,  0.93f, 0.71f, 0.14f,   -500.0f,  300.0f,  -1.0f,
// dreapta sus

		// Baza de sus
				// coordonate		// culori			// normale
		  -500.0f, -300.0f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,   -500.0f, -300.0f,
1.0f,   // stanga sus
		    0.0f, -300.0f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,     0.0f, -300.0f,   1.0f,
// stanga jos
		    0.0f,  300.0f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,     0.0f,  300.0f,   1.0f,
// dreapta jos
		  -500.0f,  300.0f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,     500.0f,  300.0f,
1.0f,   // dreapta sus

		// Peretele din stanga
		  -500.0f, -300.0f,   0.0f, 1.0f,  0.93f, 0.71f, 0.14f,  -500.0f,  -1.0f,   0.0f,
		    0.0f, -300.0f,   0.0f, 1.0f,  0.93f, 0.71f, 0.14f,     0.0f,  -1.0f,   0.0f,
			0.0f, -300.0f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,     0.0f,  -1.0f,
250.0f,
		  -500.0f, -300.0f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,   -500.0f,  -1.0f,
250.0f,

		// Peretele din dreapta
		  -500.0f,  300.0f,   0.0f, 1.0f,  0.93f, 0.71f, 0.14f,  -500.0f,   1.0f,   0.0f,
			0.0f,  300.0f,   0.0f, 1.0f,  0.93f, 0.71f, 0.14f,     0.0f,   1.0f,
0.0f,
			0.0f,  300.0f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,     0.0f,   1.0f,
250.0f,
```

```
                -500.0f,  300.0f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,   -500.0f,     1.0f,
250.0f,

            // Peretele din spate
                -500.0f, -300.0f,    0.0f, 1.0f,  0.93f, 0.71f, 0.14f,   -500.0f, -300.0f,
0.0f,
             -500.0f, -300.0f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,  -500.0f, -300.0f, 250.0f,
                -500.0f,  300.0f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,   -500.0f,  300.0f,
250.0f,
                -500.0f,  300.0f,    0.0f, 1.0f,  0.93f, 0.71f, 0.14f,   -500.0f,  300.0f,
0.0f,

                // Peretele din fata
                    0.0f, -300.5f, 250.0f, 1.0f,   0.93f, 0.71f, 0.14f,        1.0f, -300.0f,
250.0f,
                  0.0f, -300.5f,   0.0f, 1.0f,  0.93f, 0.71f, 0.14f,      1.0f, -300.0f, 150.0f,
               0.0f,  300.5f,   0.0f, 1.0f,  0.93f, 0.71f, 0.14f,      1.0f,  300.0f, 150.0f,
               0.0f,  300.5f, 250.0f, 1.0f,  0.93f, 0.71f, 0.14f,      1.0f,  300.0f, 250.0f,


                // ACOPERISUL CASEI (TRUNCHIUL DE PIRAMIDA)
                //
                // Baza de jos a acoperisului
                                    // coordonate            // culori               // normale
                -600.0f, -400.0f, 250.0f, 1.0f,   0.6f, 0.17f, 0.14f,    -600.0f, -400.0f,
250.0f,   // stanga sus
                 100.0f, -400.0f, 250.0f, 1.0f,   0.6f, 0.17f, 0.14f,     100.0f, -400.0f,
250.0f,   // stanga jos
                 100.0f,  400.0f, 250.0f, 1.0f,   0.6f, 0.17f, 0.14f,     100.0f,  400.0f,
250.0f,   // dreapta jos
                -600.0f,  400.0f, 250.0f, 1.0f,   0.6f, 0.17f, 0.14f,    -600.0f,  400.0f,
250.0f,   // dreapta sus

                // Baza de sus a acoperisului
                                    // coordonate            // culori               // normale
                -425.0f, -225.0f, 425.0f, 1.0f,   0.6f, 0.17f, 0.14f,    -425.0f, -225.0f,
425.0f,   // stanga sus
                 -75.0f, -225.0f, 425.0f, 1.0f,   0.6f, 0.17f, 0.14f,     -75.0f, -225.0f,
425.0f,   // stanga jos
```

```
                -75.0f,  225.0f, 425.0f, 1.0f,   0.6f, 0.17f, 0.14f,     -75.0f,  225.0f,
425.0f,   // dreapta jos
               -425.0f,  225.0f, 425.0f, 1.0f,   0.6f, 0.17f, 0.14f,    -425.0f,  225.0f,
425.0f,   // dreapta sus


            // USA CASEI
                                 // coordonate            // culori              // normale
               3.0f,  -45.0f,   0.0f, 1.0f,   0.8f, 0.17f, 0.14f,    -1.0f,  0.0f,  0.0f,   //
stanga jos
               3.0f,   45.0f,   0.0f, 1.0f,   0.8f, 0.17f, 0.14f,    -1.0f,  0.0f,  0.0f,   //
dreapta jos
               3.0f,   45.0f, 150.0f, 1.0f,   0.8f, 0.17f, 0.14f,    -1.0f,  0.0f,  0.0f,   //
dreapta sus
               3.0f,  -45.0f, 150.0f, 1.0f,   0.8f, 0.17f, 0.14f,    -1.0f,  0.0f,  0.0f,   //
stanga sus


            // FERESTRELE CASEI
                                 // coordonate            // culori              // normale
               3.0f,  -60.0f, 100.0f, 1.0f,   0.0f,  0.7f,  1.0f,    3.0f,  -60.0f, 120.0f,
// stanga jos
                    3.0f,    60.0f,  100.0f, 1.0f,    0.0f,  0.7f,  1.0f,    3.0f,    60.0f,
120.0f,   // dreapta jos
                    3.0f,    60.0f,  220.0f, 1.0f,    0.0f,  0.7f,  1.0f,    3.0f,    60.0f,
240.0f,   // dreapta sus
                    3.0f,   -60.0f,  220.0f, 1.0f,    0.0f,  0.7f,  1.0f,    3.0f,   -60.0f,
240.0f,   // stanga sus


            // HORNUL CASEI (paralelipiped)
            //
            // Baza de jos a hornului
                                 // coordonate            // culori              // normale
              -260.0f,   260.0f,  365.0f, 1.0f,   0.2f,  0.0f,  0.0f,    -260.0f,   260.0f,
365.0f,    // stanga sus
               -240.0f,   260.0f,  365.0f, 1.0f,   0.2f,  0.0f,  0.0f,    -240.0f,   260.0f,
365.0f,    // stanga jos
               -240.0f,   280.0f,  365.0f, 1.0f,   0.2f,  0.0f,  0.0f,    -240.0f,   280.0f,
365.0f,    // dreapta jos
```

```
                -260.0f,  280.0f,  365.0f,  1.0f,    0.2f,  0.0f,  0.0f,     -260.0f,  280.0f,
365.0f,    // dreapta sus

              // Baza de sus a hornului
                        // coordonate            // culori                 // normale
                -260.0f,  260.0f,  430.0f,  1.0f,    0.2f,  0.0f,  0.0f,     -260.0f,  260.0f,
430.0f,    // stanga sus
                -240.0f,  260.0f,  430.0f,  1.0f,    0.2f,  0.0f,  0.0f,     -240.0f,  260.0f,
430.0f,    // stanga jos
                -240.0f,  280.0f,  430.0f,  1.0f,    0.2f,  0.0f,  0.0f,     -240.0f,  280.0f,
430.0f,    // dreapta jos
                -260.0f,  280.0f,  430.0f,  1.0f,    0.2f,  0.0f,  0.0f,     -260.0f,  280.0f,
430.0f,    // dreapta sus
        };


        GLubyte Indices[] = {
          // PERETII CASEI (CUBUL)
                0,  1,  2,    0,  2,  3,          // baza inferioara
                4,  5,  6,    4,  6,  7,          // baza superioara
                8,  9, 10,    8, 10, 11,                  // peretele din stanga
                12, 13, 14,  12, 14, 15,                  // peretele din dreapta
                16, 17, 18,  16, 18, 19,                  // peretele din spate
                20, 21, 22,  20, 22, 23,                  // peretele din fata

          // ACOPERISUL CASEI (TRUNCHIUL DE PIRAMIDA)
                24, 25, 26,  24, 26, 27,      // baza inferioara
                28, 29, 30,  28, 30, 31,      // baza superioara
                25, 26, 30,  25, 30, 29,      // fata
       24, 27, 31,  24, 31, 28,       // spate
                24, 25, 29,  24, 29, 28,      // stanga
                27, 26, 30,  27, 30, 31,      // dreapta

          // USA CASEI
                32, 33, 34,  32, 34, 35,

          // FERESTRELE CASEI
                36, 37, 38,  36, 38, 39,

          // HORNUL CASEI (paralelipiped)
```

```
        40, 41, 42,  40, 42, 43,      // baza inferioara
        44, 45, 46,  44, 46, 47,      // baza superioara
        41, 42, 46,  41, 46, 45,      // fata
        40, 43, 47,  40, 47, 44,      // spate
        40, 41, 45,  40, 45, 44,      // stanga
        43, 42, 46,  43, 46, 47       // dreapta

    };


    glGenVertexArrays(1, &VaoHouse);
    glBindVertexArray(VaoHouse);

    glGenBuffers(1, &VboHouse);
    glGenBuffers(1, &EboHouse);

    glBindBuffer(GL_ARRAY_BUFFER, VboHouse);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboHouse);

    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices,
GL_STATIC_DRAW);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices,
GL_STATIC_DRAW);

    // atributul 0 = pozitie
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat),
(GLvoid*)0);
    // atributul 1 = culoare
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat),
(GLvoid*)(4 * sizeof(GLfloat)));
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat),
(GLvoid*)(7 * sizeof(GLfloat)));
}


void DestroyVBO(void) {
```

```
	//  Eliberarea atributelor din shadere (pozitie, culoare, normale etc.)
	glDisableVertexAttribArray(2);
	glDisableVertexAttribArray(1);
	glDisableVertexAttribArray(0);

	//  Stergerea bufferelor pentru VARFURI (Coordonate, Culori), INDICI;
	glBindBuffer(GL_ARRAY_BUFFER, 0);

	glDeleteBuffers(1, &VboGround);
	glDeleteBuffers(1, &EboGround);

	glDeleteBuffers(1, &VboTreeTrunk);
	glDeleteBuffers(1, &EboTreeTrunk);

	glDeleteBuffers(1, &VboTreeCrown);
	glDeleteBuffers(1, &EboTreeCrown);

	glDeleteBuffers(1, &VboTreeStar);
	glDeleteBuffers(1, &EboTreeStar);

	glDeleteBuffers(1, &VboHouse);
	glDeleteBuffers(1, &EboHouse);

	//  Dezactivarea VAO (Vertex Array Object)
	glBindVertexArray(0);
	glDeleteVertexArrays(1, &VaoGround);
	glDeleteVertexArrays(1, &VaoTreeTrunk);
	glDeleteVertexArrays(1, &VaoTreeCrown);
	glDeleteVertexArrays(1, &VaoTreeStar);
	glDeleteVertexArrays(1, &VaoHouse);
}


void CreateShaders(void) {
	ProgramId = LoadShaders("Shader.vert", "Shader.frag");
	glUseProgram(ProgramId);
}


void DestroyShaders(void) {
```

```
	glDeleteProgram(ProgramId);
}


void Cleanup(void) {
	DestroyShaders();
	DestroyVBO();
}


void Initialize(void) {
	glClearColor(0.3f, 0.3f, 0.3f, 0.0f);    // culoarea de fundal a ecranului

	CreateVAOGround();
	CreateVAOTreeTrunk();
	CreateVAOTreeCrown();
	CreateVAOTreeStar();
	CreateVAOHouse();

	CreateShaders();

	// locatii pentru shader-e
	myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
	matrUmbraLocation = glGetUniformLocation(ProgramId, "matrUmbra");
	viewLocation = glGetUniformLocation(ProgramId, "view");
	projLocation = glGetUniformLocation(ProgramId, "projection");
	lightColorLocation = glGetUniformLocation(ProgramId, "lightColor");
	lightPosLocation = glGetUniformLocation(ProgramId, "lightPos");
	viewPosLocation = glGetUniformLocation(ProgramId, "viewPos");
	codColLocation = glGetUniformLocation(ProgramId, "codCol");
	fogColorLocation = glGetUniformLocation(ProgramId, "fogColor");
}


void RenderFunction(void) {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glEnable(GL_DEPTH_TEST);

	//  pozitia observatorului
	Obsx = Refx + dist * cos(alpha) * cos(beta);
```

```
        Obsy = Refy + dist * cos(alpha) * sin(beta);
        Obsz = Refz + dist * sin(alpha);

        //  matrice de vizualizare + proiectie
        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]);
        projection = glm::infinitePerspective(fov, GLfloat(width) / GLfloat(height),
znear);
        glUniformMatrix4fv(projLocation, 1, GL_FALSE, &projection[0][0]);

        //  matricea pentru umbra
        float D = -3.0;
        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]);

        //  Variabile uniforme pentru iluminare
        glUniform3f(lightColorLocation, 1.0f, 1.0f, 1.0f);
        glUniform3f(lightPosLocation, xL, yL, zL);
        glUniform3f(viewPosLocation, Obsx, Obsy, Obsz);

        //  culoarea pentru ceata
        glUniform3f(fogColorLocation, 0.8f, 0.8f, 0.8f);



        // --- DESENARE GROUND ---
```

```
	glBindVertexArray(VaoGround);

	codCol = 0;
	glUniform1i(codColLocation, codCol);

	myMatrix = glm::mat4(1.0f);
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

	glDrawElements(GL_TRIANGLES, 18, GL_UNSIGNED_BYTE, 0);



	// --- DESENARE BRAD ---

	for (int i = 0; i < 25; ++i) {
		if (i < 9)
			myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(700.0f - (175.0 * i), -700.0f, 0.0f));
		else if (i < 18)
			myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(700.0f - (175.0 * (i - 9)), 700.0f, 0.0f));
		else
			myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(-700.0f, -700.0f + (175.0 * (i - 17)), 0.0f));

		glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);


		// DESENARE TRUNCHI
		glBindVertexArray(VaoTreeTrunk);

		codCol = 0;
		glUniform1i(codColLocation, codCol);

		// desenarea bazei superioare
		glDrawElements(GL_TRIANGLE_FAN, NR_MERIDIANE_CILINDRU, GL_UNSIGNED_BYTE, 0);

		// desenarea bazei inferioare
```

```
		glDrawElements(GL_TRIANGLE_FAN, NR_MERIDIANE_CILINDRU, GL_UNSIGNED_BYTE, (GLvoid*)(NR_MERIDIANE_CILINDRU * sizeof(GLubyte)));

		// desenarea fetelor laterale
		glDrawElements(GL_QUADS, 4 * NR_MERIDIANE_CILINDRU, GL_UNSIGNED_BYTE, (GLvoid*)(2 * NR_MERIDIANE_CILINDRU * sizeof(GLubyte)));

		// desenare umbra
		codCol = 1;
		glUniform1i(codColLocation, codCol);
		glDrawElements(GL_QUADS, 4 * NR_MERIDIANE_CILINDRU, GL_UNSIGNED_BYTE, (GLvoid*)(2 * NR_MERIDIANE_CILINDRU * sizeof(GLubyte)));


		// DESENARE COROANA
		glBindVertexArray(VaoTreeCrown);

		codCol = 0;
		glUniform1i(codColLocation, codCol);

		// desenarea bazei
		glDrawElements(GL_TRIANGLE_FAN, NR_MERIDIANE_CON, GL_UNSIGNED_BYTE, 0);

		// desenarea fetelor laterale
		for (int i = 0; i < NR_MERIDIANE_CON; ++i)
		{
			glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_BYTE, (GLvoid*)((NR_MERIDIANE_CON + 3 * i) * sizeof(GLubyte)));
		}

		// desenare umbra
		codCol = 1;
		glUniform1i(codColLocation, codCol);
		glDrawElements(GL_TRIANGLES, 3 * NR_MERIDIANE_CON, GL_UNSIGNED_BYTE, (GLvoid*)(NR_MERIDIANE_CON * sizeof(GLubyte)));
```

```
        // DESENARE STEA
        glBindVertexArray(VaoTreeStar);

        codCol = 0;
        glUniform1i(codColLocation, codCol);

        glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_BYTE, 0);

        // desenare umbra
        codCol = 1;
        glUniform1i(codColLocation, codCol);
        glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_BYTE, 0);
}



// --- DESENARE CASA ---

glBindVertexArray(VaoHouse);

// desenare pereti + acoperis + usa
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

codCol = 0;
glUniform1i(codColLocation, codCol);

glDrawElements(GL_TRIANGLES, 78, GL_UNSIGNED_BYTE, 0);


// desenare ferestre
glLineWidth(5.0f);

// fereastra din stanga
codCol = 0;
glUniform1i(codColLocation, codCol);

myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -200.0f, 0.0f));
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
```

```
	glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, (GLvoid*)(78 * sizeof(GLubyte)));

	// fereastra din dreapta
	codCol = 0;
	glUniform1i(codColLocation, codCol);

	myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 200.0f, 0.0f));
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

	glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, (GLvoid*)(78 * sizeof(GLubyte)));


	// desenare horn
	codCol = 0;
	glUniform1i(codColLocation, codCol);

	myMatrix = glm::mat4(1.0f);
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

	glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (GLvoid*)(84 * sizeof(GLubyte)));


	// desenare umbra (casa + horn)
	myMatrix = glm::mat4(1.0f);
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

	codCol = 1;
	glUniform1i(codColLocation, codCol);

	glDrawElements(GL_TRIANGLES, 72, GL_UNSIGNED_BYTE, 0);
	glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (GLvoid*)(84 * sizeof(GLubyte)));


	glutSwapBuffers();
	glFlush();
```

```
}


int main(int argc, char* argv[]) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
	glutInitWindowSize(winWidth, winHeight);                                    //
Dimensiunile ferestrei de vizualizare
	glutInitWindowPosition(25, 25);
	//  Pozitia initiala a ferestrei de vizualizare
	glutCreateWindow("Peisaj de iarna");                                        //
	Creeaza fereastra de vizualizare, indicand numele acesteia;

	glewInit();

	Initialize();
	glutDisplayFunc(RenderFunction);
	//  Desenarea scenei in fereastra;
	glutIdleFunc(RenderFunction);                                               //
Functie de apelare constanta

	glutKeyboardFunc(processNormalKeys);
	glutSpecialFunc(processSpecialKeys);

	glutCloseFunc(Cleanup);
		//  Eliberarea resurselor alocate de program;

	glutMainLoop();
	return 0;
}
```