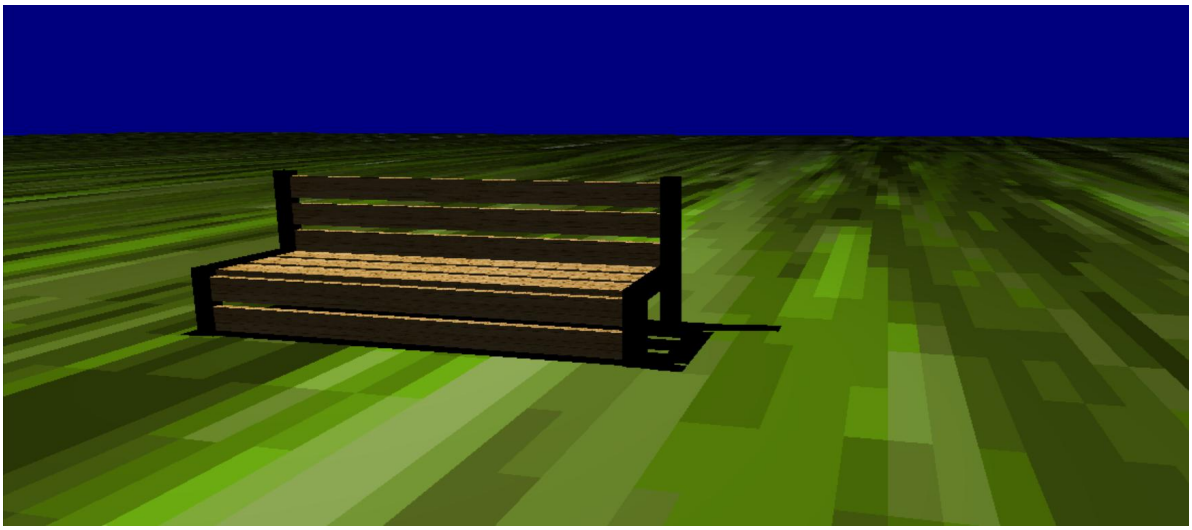
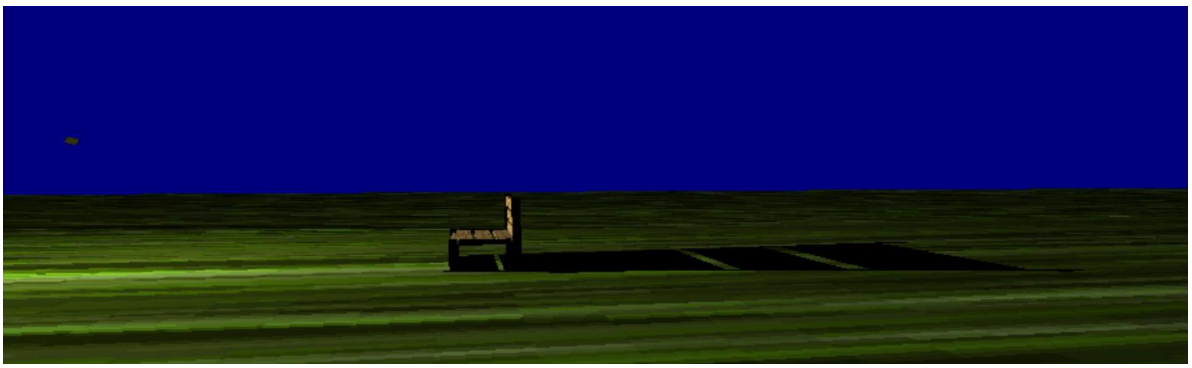
PROIECT 2

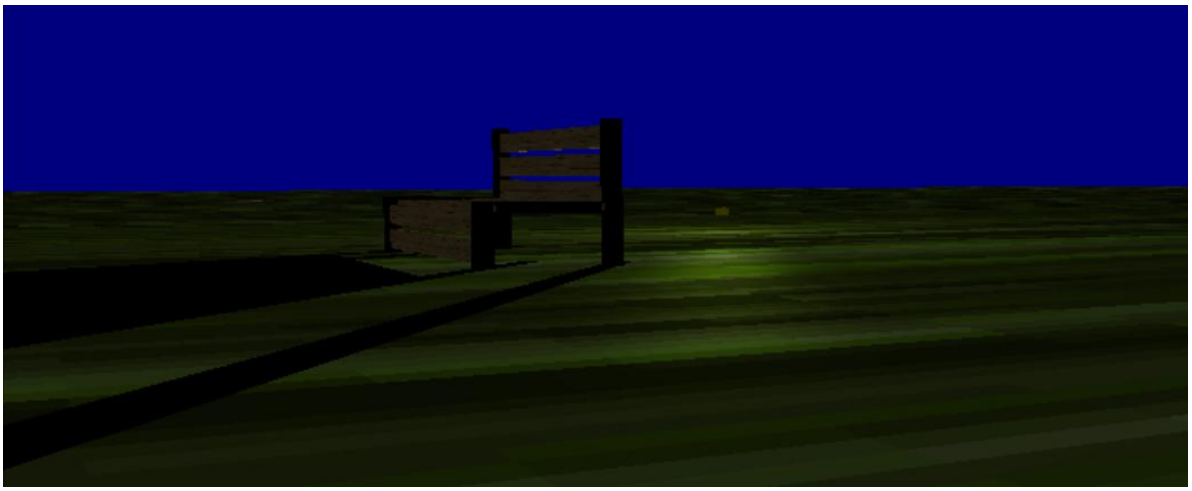
ECHIPA : ADAM ADRIAN CLAUDIU, grupa 342
POINARITA ANDREEA DIANA, grupa 343

1. CONCEPTUL PROIECTULUI

Proiectul reprezinta o animatie 3D pentru a simula iluminarea si formarea secventiala a umbrei unei banci atunci cand lumina provine atat de la o sursa fixa, dar si de la o sursa aflata in miscare (e.x. soarele pe parcursul unei zile).







2. ELEMENTE INCLUSE:

- *Obiectele 3D* (podeaua, sursa de lumina, banca) - reprezentate prin scalarea, translatarea si rotatia unui paralelipiped.

```
void DrawPlank(float rotationAngle, glm::vec3 rotationAxis, glm::vec3 translation) {
    translationMat = glm::translate(glm::mat4(1.0f), translation);
    scaleMat = glm::scale(glm::mat4(1.0f), glm::vec3(LPlank/unit, hPlank / unit, lPlank / unit));
    if (rotationAngle != 0) {
        rotationMat = glm::rotate(glm::mat4(1.0f), rotationAngle, rotationAxis);
        myMatrix = translationMat * rotationMat * scaleMat;
    }
    else {
        myMatrix = translationMat * scaleMat;
    }
    glUniformMatrix4fv(myMatrixLoc, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_TRIANGLES, 11, 12);
}
```

- *Textura* - bancii si podelei le-a fost aplicata texturarea, prin inlocuirea variabilei corespunzatoare culorii obiectului cu coordonatele pentru texturare, incluse in continuare in formula pentru iluminare:

```
vec3 color = objectColor;

if(tex_code == 1) {
      vec3 texColor =  texture(myTexture, tex_Coord).xyz;
      color = texColor;
  }

  // Ambient
  float ambientStrength = 0.2f;
  vec3 ambient_light = ambientStrength * lightColor;
  vec3 ambient_term= ambient_light * color;

  // Diffuse
  vec3 norm = normalize(Normal);
  vec3 lightDir = normalize(inLightPos - FragPos);
  float diff = max(dot(norm, lightDir), 0.0);
  vec3 diffuse_light = lightColor;
  vec3 diffuse_term = diff * diffuse_light * color;
```

Deoarece obiectele au fost reprezentate folosind acelasi obiect initial (aplicandu-i transformari), am introdus variabila tex_code pentru a informa in shadere ca se vrea a fi folosita textura sau nu, ci culoarea simpla a obiectului.

```
void setTexCode(int texCode) {
    glUniform1i(glGetUniformLocation(ProgramIdf, "tex_code"), texCode);
    glUniform1i(glGetUniformLocation(ProgramIdv, "tex_code"), texCode);
}
```

```
void DrawPlanks() {
    const char* imageName = "wood.png";
    LoadTexture(imageName);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    setTexCode(1);

    // de jos
    DrawPlank(0.f, glm::vec3(0.f, 0.f, 0.f), glm::vec3(0, -50, -50));
    DrawPlank(0.f, glm::vec3(0.f, 0.f, 0.f), glm::vec3(0, -50, 0));

    // de stat
    DrawPlank(PI/2, glm::vec3(1.f, 0.f, 0.f), glm::vec3(0, -25, 25));
    DrawPlank(PI/2, glm::vec3(1.f, 0.f, 0.f), glm::vec3(0, 25, 25));
    DrawPlank(PI/2, glm::vec3(1.f, 0.f, 0.f), glm::vec3(0, 75, 25));

    // de sus
    DrawPlank(0.f, glm::vec3(0.f, 0.f, 0.f), glm::vec3(0, 100, 50));
    DrawPlank(0.f, glm::vec3(0.f, 0.f, 0.f), glm::vec3(0, 100, 100));
    DrawPlank(0.f, glm::vec3(0.f, 0.f, 0.f), glm::vec3(0, 100, 150));

    setTexCode(0);
}
```

- *Umbra* - umbra bancii a fost desenata folosind aceleasi coordonate initiale, insa prin setarea culorii negre (fara a folosi texturarea sau a aplica iluminarea) si prin adaugarea matricei - umbra, care este actualizata la fiecare modificare a pozitiei sursei de lumina:

```
void setMatriceUmbra(float xL, float yL, float zL) {
    float D = 77.0f;
    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;

    matrUmbraLocation = glGetUniformLocation(ProgramIdv, "matrUmbra");
    matrUmbraLocation = glGetUniformLocation(ProgramIdf, "matrUmbra");
    glUniformMatrix4fv(matrUmbraLocation, 1, GL_FALSE, &matrUmbra[0][0]);
}
```

```
// Variabile uniforme pentru iluminare
glUniform3f(objectColorLoc, 1.0f, 0.5f, 0.4f);
glUniform3f(lightColorLoc, 1.0f, 1.0f, 1.0f);
if (360 <= Time && Time < 1080) {
    glUniform3f(lightPosLoc, SunPos.x, SunPos.y, SunPos.z);
    setMatriceUmbra(SunPos.x, SunPos.y, SunPos.z);
} else {
    glUniform3f(lightPosLoc, XLight, YLight, ZLight);
    setMatriceUmbra(XLight, YLight, ZLight);
}
```

```
void DrawBenchShadows() {
    glUniform1i(codColLocation, 1);
    DrawPlanks();
    DrawMargins();
    glUniform1i(codColLocation, 0);
}
```

- *Lumina* – pentru simularea luminii am folosit formulele clasice prezentate in cadrul laboratorului.

```
// Ambient
float ambientStrength = 0.2f;
vec3 ambient_light = ambientStrength * lightColor;
vec3 ambient_term= ambient_light * color;

// Diffuse
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(inLightPos - FragPos);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse_light = lightColor;
vec3 diffuse_term = diff * diffuse_light * color;

// Specular
float specularStrength = 0.5f;
float shininess = 100.0f;
vec3 viewDir = normalize(inViewPos - FragPos); // versorul catre observator
vec3 reflectDir = normalize(reflect(-lightDir, norm));  // versorul vectorului R
float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
vec3 specular_light = specularStrength  * lightColor; // specular_light=specularStrength  * lightColor
vec3 specular_term = spec * specular_light * color;   // specular_material=objectColor

// Culoarea finala
  vec3 emission=vec3(0.0, 0.0, 0.0);
//vec3 emission=vec3(1.0,0.8,0.4);
vec3 result = emission + (ambient_term + diffuse_term + specular_term);
out_Color = vec4(result, 1.0f);
```

Soarele si licuriciul de noapte au coordonate diferite, fiecare fiind reprezentat in momente diferite ale zilei, niciodata simultan. Pentru desenarea acestora am creeat cate o functie.

```
void DrawLight()
{
    glUniform3f(objectColorLoc, 1.f, 1.f, 0.f);
    translationMat = glm::translate(glm::mat4(1.0f), glm::vec3(XLight, YLight, ZLight));
    myMatrix = translationMat;
    glUniformMatrix4fv(myMatrixLoc, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_TRIANGLES, 11, 12);
}

void DrawSun() {
    glUniform3f(objectColorLoc, 1.f, 1.f, 0.f);
    float rotationAngle = 2 * PI * (float)Time / (float)MAX_MINS_IN_DAY + PI;
    rotationMat = glm::rotate(glm::mat4(1.0f), rotationAngle, glm::vec3(1, 0, 0));
    translationMat = glm::translate(glm::mat4(1.0f), SunInitPos);
    myMatrix = rotationMat * translationMat;
    SunPos = myMatrix * glm::vec4(SunAnchor, 1.0f);
    glUniformMatrix4fv(myMatrixLoc, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_TRIANGLES, 11, 12);
}
```

- *Timpul* - Luand in considerare tematica trecerii timpului, am incercat sa simulam acest fenomen printr-o reprezentare grafica. Am impartit ziua in 1440 minute, iar la fiecare apasare a tastei **SPACE** incrementam **Time** cu **LAPS_TIME** minute.

```
int LAPS_TIME = 10;
int MAX_MINS_IN_DAY = 1440;
int Time = 720; // (0, 1440) mins in a day
```

Cu ajutorul acestei referinte putem decide cand reprezentam soarele si cand reprezentam licuriciul.

```
if (360 <= Time && Time < 1080) {
    glUniform3f(lightPosLoc, SunPos.x, SunPos.y, SunPos.z);
    setMatriceUmbra(SunPos.x, SunPos.y, SunPos.z);
} else {
    glUniform3f(lightPosLoc, XLight, YLight, ZLight);
    setMatriceUmbra(XLight, YLight, ZLight);
}
glUniform3f(viewPosLoc, Obsx, Obsy, Obsz);

switch (rendermode) { ... }

DrawPlanks();
DrawMargins();
DrawFloor();

if (360 <= Time && Time < 1080) {
    DrawSun();
} else {
    DrawLight();
}
```

Cum 720 am considerat ora 12:00, 360 este ora 6:00, iar 1080 este 18:00. Soarele rasare astfel la 6:00 AM is apune la 6:00 PM. Acesta are o singura libertate de miscare, depinzand doar de timp.

Pe de alta parte, licuriciul are 3 libertati de miscare, putand fi controlat dealungul celor 3 axe de miscare (A(<), D(>), S(v), W(^), Q(O), E(o)).

3. ELEMENTE DE DIFICULTATE

Elementele de dificultate din aceasta scena (descrise la punctul anterior) sunt reprezentate de existenta celor doua surse de lumina aflate in miscare (soarele si licuriciul) si de simularea trecerii timpului prin reprezentarea treptata a umbrei obiectului 3D care se formeaza in raport cu pozitia sursei de lumina curente, de la rasaritul soarelui pana la apus.

4. ORIGINALITATE

In scena 3D sursa de lumina este folosita sub 2 forme, astfel incat se poate urmari secvential modul in care obiectul este iluminat si cum se formeaza umbra acestuia.

5. COD SURSA

```
// Iluminare: aplicarea mecanismului la nivel de varf / de fragment
#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"
#include "SOIL.h"

using namespace std;

// identificatori
GLuint
VaoId,
VboId,
ColorBufferId,
ProgramIdv,
ProgramIdf,
viewLocation,
projLocation,
codColLocation,
depthLocation,
matrUmbraLocation,
rendermode,
TextureId,
l1, l2,
codCol,
texture;

GLint objectColorLoc, lightColorLoc, lightPosLoc, viewPosLoc, myMatrixLoc;

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

// variabile pentru matricea de vizualizare
float Obsx = 0.0, Obsy = -600.0, Obsz = 0.f;
float Refx = 0.0f, Refy = 1000.0f, Refz = 0.0f;
float Vx = 0.0, Vy = 0.0, Vz = 1.0;

// variabile pentru matricea de proiectie
```

```
float width = 800, height = 600, znear = 0.1, fov = 45;
float PI = 3.141592;

// matrice utilizate
glm::mat4 view, projection, myMatrix, rotationMat, translationMat, scaleMat;

float unit = 10;
float spacing = 10;
float LPlank = 250, lPlank = 20, hPlank = 5;

float XLight = -400, YLight = 0, ZLight = 400;

int LAPS_TIME = 10;
int MAX_MINS_IN_DAY = 1440;
int Time = 720; // (0, 1440) mins in a day
glm::vec3 SunInitPos = glm::vec3(-100.0f, 0.0f, 1000.0f);
glm::vec3 SunAnchor = glm::vec3(0.0f, 0.0f, 0.0f);
glm::vec3 SunPos = SunInitPos;

enum {
	ll_Frag, ll_Frag_Av, ll_Vert, ll_Vert_Av
};

void menu(int selection)
{
	rendermode = selection;
	glutPostRedisplay();
}

void processNormalKeys(unsigned char key, int x, int y)
{
	switch (key) {
	case 'l':
		Vx -= 0.1;
		break;
	case 'r':
		Vx += 0.1;
		break;
	case '+':
		Obsy += 10;
		break;
	case '-':
		Obsy -= 10;
		break;
	case 'w':
		ZLight += 10;
		break;
```

```
	case 's':
		ZLight -= 10;
		break;
	case 'a':
		XLight -= 10;
		break;
	case 'd':
		XLight += 10;
		break;
	case 'q':
		YLight -= 10;
		break;
	case 'e':
		YLight += 10;
		break;
	case ' ':
		Time += LAPS_TIME;
		Time = Time % MAX_MINS_IN_DAY;
		break;
	case '`':
		Time -= LAPS_TIME;
		if (Time < 0) {
			Time = MAX_MINS_IN_DAY - 1;
		}
		cout << Time <<endl;
		break;

	}
	if (key == 27)
		exit(0);
}
void processSpecialKeys(int key, int xx, int yy)
{
	switch (key) {
	case GLUT_KEY_LEFT:
		Obsx -= 20;
		break;
	case GLUT_KEY_RIGHT:
		Obsx += 20;
		break;
	case GLUT_KEY_UP:
		Obsz += 20;
		break;
	case GLUT_KEY_DOWN:
		Obsz -= 20;
		break;
	}
```

```
}
void CreateVBO(void)
{
	GLfloat Vertices[] =
	{
		// FRAGMENT

		// inspre Oz'
			(-1)* unit, (-1)* unit, (-1)* unit,    0.0f,    0.0f, -1.0f,
				unit, (-1)* unit, (-1)* unit,    0.0f,    0.0f, -1.0f,
				unit,           unit, (-1)* unit,    0.0f,    0.0f, -1.0f,
				unit,           unit, (-1)* unit,    0.0f,    0.0f, -1.0f,
			(-1)* unit,           unit, (-1)* unit,    0.0f,    0.0f, -1.0f,
			(-1)* unit, (-1)* unit, (-1)* unit,    0.0f,    0.0f, -1.0f,

			// inspre Oz
			(-1)* unit, (-1)* unit,    unit,          0.0f,    0.0f,    1.0f,
				unit, (-1)* unit,    unit,          0.0f,    0.0f,    1.0f,
				unit,           unit,    unit,          0.0f,    0.0f,    1.0f,
				unit,           unit,    unit,          0.0f,    0.0f,    1.0f,
			(-1)* unit,           unit,    unit,          0.0f,    0.0f,    1.0f,
			(-1)* unit, (-1)* unit,    unit,          0.0f,    0.0f,    1.0f,

			// inspre Ox'
			(-1)* unit,           unit,           unit, -1.0f,    0.0f,    0.0f,
			(-1)* unit,           unit, (-1)* unit, -1.0f,    0.0f,    0.0f,
			(-1)* unit, (-1)* unit, (-1)* unit, -1.0f,    0.0f,    0.0f,
			(-1)* unit, (-1)* unit, (-1)* unit, -1.0f,    0.0f,    0.0f,
			(-1)* unit, (-1)* unit,           unit, -1.0f,    0.0f,    0.0f,
			(-1)* unit,           unit,           unit, -1.0f,    0.0f,    0.0f,

			// inspre Ox
			  unit,           unit,           unit,    1.0f,    0.0f,    0.0f,
			  unit,           unit, (-1)* unit,       1.0f,    0.0f,    0.0f,
			  unit,    (-1)* unit, (-1)* unit,      1.0f,    0.0f,    0.0f,
			  unit, (-1)* unit, (-1)* unit,         1.0f,    0.0f,    0.0f,
			  unit, (-1)* unit,           unit,      1.0f,    0.0f,    0.0f,
			  unit,          unit,          unit,    1.0f,    0.0f,    0.0f,

			  // inspre Oy'
			(-1)* unit,    (-1)* unit, (-1)* unit,    0.0f, -1.0f,    0.0f,
				unit,    (-1)* unit, (-1)* unit,    0.0f, -1.0f,    0.0f,
				unit,    (-1)* unit,         unit,    0.0f, -1.0f,    0.0f,
				unit,    (-1)* unit,         unit,    0.0f, -1.0f,    0.0f,
			(-1)* unit,    (-1)* unit,           unit,    0.0f, -1.0f,    0.0f,
			(-1)* unit,    (-1)* unit, (-1)* unit,    0.0f, -1.0f,    0.0f,
```

```
// inspre Oy
(-1)* unit,    unit, (-1)* unit,        0.0f,    1.0f,    0.0f,
        unit,    unit, (-1)* unit,        0.0f,    1.0f,    0.0f,
        unit,    unit,      unit,    0.0f,    1.0f,    0.0f,
        unit,    unit,      unit,    0.0f,    1.0f,    0.0f,
(-1)* unit,    unit,      unit,    0.0f,    1.0f,    0.0f,
(-1)* unit,    unit, (-1)* unit,        0.0f,    1.0f,    0.0f,

//////////////////////////////////////////////////
// Vertex

// inspre Oz'
(-1)*unit, (-1)*unit, (-1)*unit,    -1.0f,    -1.0f, -1.0f,
         unit, (-1)*unit, (-1)*unit,      1.0f,    -1.0f, -1.0f,
         unit,          unit, (-1)*unit,      1.0f,      1.0f, -1.0f,
         unit,          unit, (-1)*unit,      1.0f,      1.0f, -1.0f,
(-1)*unit,           unit, (-1)*unit,    -1.0f,      1.0f, -1.0f,
(-1)*unit, (-1)*unit, (-1)*unit,    -1.0f,    -1.0f, -1.0f,

// inspre Oz
(-1)*unit, (-1)*unit,    unit,          -1.0f,    -1.0f,    1.0f,
        unit, (-1)*unit,    unit,           1.0f,    -1.0f,    1.0f,
        unit,          unit,    unit,           1.0f,      1.0f,    1.0f,
        unit,          unit,    unit,           1.0f,      1.0f,    1.0f,
(-1)*unit,           unit,    unit,         -1.0f,      1.0f,    1.0f,
(-1)*unit, (-1)*unit,    unit,          -1.0f,    -1.0f,    1.0f,

// inspre Ox'
(-1)*unit,           unit,           unit, -1.0f,      1.0f,      1.0f,
(-1)*unit,           unit, (-1)*unit, -1.0f,      1.0f,    -1.0f,
(-1)*unit, (-1)*unit, (-1)*unit, -1.0f,    -1.0f,    -1.0f,
(-1)*unit, (-1)*unit, (-1)*unit, -1.0f,    -1.0f,    -1.0f,
(-1)*unit, (-1)*unit,         unit, -1.0f,    -1.0f,      1.0f,
(-1)*unit,           unit,           unit, -1.0f,      1.0f,      1.0f,

// inspre Ox
  unit,          unit,          unit,    1.0f,      1.0f,      1.0f,
  unit,          unit, (-1)*unit,        1.0f,      1.0f,    -1.0f,
  unit, (-1)*unit, (-1)*unit,    1.0f,    -1.0f,    -1.0f,
  unit, (-1)*unit, (-1)*unit,    1.0f,    -1.0f,    -1.0f,
  unit, (-1)*unit,          unit,       1.0f,    -1.0f,      1.0f,
  unit,          unit,          unit,    1.0f,      1.0f,      1.0f,

  // inspre Oy'
(-1)*unit, (-1)*unit, (-1)*unit,    -1.0f, -1.0f,    -1.0f,
        unit, (-1)*unit, (-1)*unit,      1.0f, -1.0f,    -1.0f,
        unit, (-1)*unit,        unit,      1.0f, -1.0f,      1.0f,
```

```
        unit, (-1)*unit,        unit,     1.0f, -1.0f,     1.0f,
    (-1)*unit, (-1)*unit,        unit,    -1.0f, -1.0f,     1.0f,
    (-1)*unit, (-1)*unit, (-1)*unit,    -1.0f, -1.0f,    -1.0f,

    // inspre Oy
    (-1)*unit,    unit, (-1)*unit,        -1.0f,    1.0f,    -1.0f,
        unit,    unit, (-1)*unit,         1.0f,    1.0f,    -1.0f,
        unit,    unit,      unit,      1.0f,    1.0f,      1.0f,
        unit,    unit,      unit,      1.0f,    1.0f,      1.0f,
    (-1)*unit,    unit,       unit,    -1.0f,    1.0f,      1.0f,
    (-1)*unit,    unit, (-1)*unit,        -1.0f,    1.0f,    -1.0f
};



static const GLfloat Textures[] =
{
    0.0f, 0.0f,
    1.0f, 0.0f,
    1.0f, 1.0f,
    1.0f, 1.0f,
    0.0f, 1.0f,
    0.0f, 0.0f,

    0.0f, 0.0f,
    1.0f, 0.0f,
    1.0f, 1.0f,
    1.0f, 1.0f,
    0.0f, 1.0f,
    0.0f, 0.0f,

    0.0f, 0.0f,
    1.0f, 0.0f,
    1.0f, 1.0f,
    1.0f, 1.0f,
    0.0f, 1.0f,
    0.0f, 0.0f,

    0.0f, 0.0f,
    1.0f, 0.0f,
    1.0f, 1.0f,
    1.0f, 1.0f,
    0.0f, 1.0f,
    0.0f, 0.0f,

    0.0f, 0.0f,
    1.0f, 0.0f,
```

```
        1.0f, 1.0f,
        1.0f, 1.0f,
        0.0f, 1.0f,
        0.0f, 0.0f,

        0.0f, 0.0f,
        1.0f, 0.0f,
        1.0f, 1.0f,
        1.0f, 1.0f,
        0.0f, 1.0f,
        0.0f, 0.0f
    };


    glGenVertexArrays(1, &VaoId);
    glGenBuffers(1, &VboId);
    glBindVertexArray(VaoId);
    glBindBuffer(GL_ARRAY_BUFFER, VboId);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);

    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(1); // atributul 1 = normale
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)(3 *
sizeof(GLfloat)));


    glGenBuffers(1, &TextureId);
    glBindBuffer(GL_ARRAY_BUFFER, TextureId);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Textures), Textures, GL_STATIC_DRAW);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(2);
}
void DestroyVBO(void)
{
    glDisableVertexAttribArray(2);
    glDisableVertexAttribArray(1);
    glDisableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glDeleteBuffers(1, &TextureId);
    glDeleteBuffers(1, &VboId);
    glBindVertexArray(0);
    glDeleteVertexArrays(1, &VaoId);
}

void LoadTexture(const char* imageName)
{
```

```
	glGenTextures(1, &texture);
	glBindTexture(GL_TEXTURE_2D, texture);

	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
	int width, height;
	unsigned char* image = SOIL_load_image(imageName, &width, &height, 0, SOIL_LOAD_RGB);
	glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
	glGenerateMipmap(GL_TEXTURE_2D);

	SOIL_free_image_data(image);
	glBindTexture(GL_TEXTURE_2D, 0);
}

void setTexCode(int texCode) {
	glUniform1i(glGetUniformLocation(ProgramIdf, "tex_code"), texCode);
	glUniform1i(glGetUniformLocation(ProgramIdv, "tex_code"), texCode);
}


void setMatriceUmbra(float xL, float yL, float zL) {
	float D = 77.0f;
	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;

	matrUmbraLocation = glGetUniformLocation(ProgramIdv, "matrUmbra");
	matrUmbraLocation = glGetUniformLocation(ProgramIdf, "matrUmbra");
	glUniformMatrix4fv(matrUmbraLocation, 1, GL_FALSE, &matrUmbra[0][0]);
}

void CreateShadersFragment(void)
{
	ProgramIdf = LoadShaders("10_02f_Shader.vert", "10_02f_Shader.frag");
	glUseProgram(ProgramIdf);
}
void CreateShadersVertex(void)
```

```
{
    ProgramIdv = LoadShaders("10_02v_Shader.vert", "10_02v_Shader.frag");
    glUseProgram(ProgramIdv);
}
void DestroyShaders(void)
{
    glDeleteProgram(ProgramIdv);
    glDeleteProgram(ProgramIdf);
}
void Initialize(void)
{
    glClearColor(0.0f, 0.0f, 0.5f, 0.1f);
    CreateVBO();
    CreateShadersFragment();
    objectColorLoc = glGetUniformLocation(ProgramIdf, "objectColor");
    lightColorLoc = glGetUniformLocation(ProgramIdf, "lightColor");
    lightPosLoc = glGetUniformLocation(ProgramIdf, "lightPos");
    viewPosLoc = glGetUniformLocation(ProgramIdf, "viewPos");
    viewLocation = glGetUniformLocation(ProgramIdf, "view");
    projLocation = glGetUniformLocation(ProgramIdf, "projection");
    myMatrixLoc = glGetUniformLocation(ProgramIdf, "myMatrix");
    codColLocation = glGetUniformLocation(ProgramIdf, "codCol");
    CreateShadersVertex();
    objectColorLoc = glGetUniformLocation(ProgramIdv, "objectColor");
    lightColorLoc = glGetUniformLocation(ProgramIdv, "lightColor");
    lightPosLoc = glGetUniformLocation(ProgramIdv, "lightPos");
    viewPosLoc = glGetUniformLocation(ProgramIdv, "viewPos");
    viewLocation = glGetUniformLocation(ProgramIdv, "view");
    projLocation = glGetUniformLocation(ProgramIdv, "projection");
    myMatrixLoc = glGetUniformLocation(ProgramIdv, "myMatrix");
    codColLocation = glGetUniformLocation(ProgramIdv, "codCol");
}

void DrawPlank(float rotationAngle, glm::vec3 rotationAxis, glm::vec3 translation) {
    translationMat = glm::translate(glm::mat4(1.0f), translation);
    scaleMat = glm::scale(glm::mat4(1.0f), glm::vec3(LPlank/unit, hPlank / unit, lPlank /
unit));
    if (rotationAngle != 0) {
        rotationMat = glm::rotate(glm::mat4(1.0f), rotationAngle, rotationAxis);
        myMatrix = translationMat * rotationMat * scaleMat;
    }
    else {
        myMatrix = translationMat * scaleMat;
    }
    glUniformMatrix4fv(myMatrixLoc, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_TRIANGLES, l1, l2);
}
```

```
void DrawPlanks() {
	const char* imageName = "wood.png";
	LoadTexture(imageName);
	glActiveTexture(GL_TEXTURE0);
	glBindTexture(GL_TEXTURE_2D, texture);
	setTexCode(1);

	// de jos
	DrawPlank(0.f, glm::vec3(0.f, 0.f, 0.f), glm::vec3(0, -50, -50));
	DrawPlank(0.f, glm::vec3(0.f, 0.f, 0.f), glm::vec3(0, -50, 0));

	// de stat
	DrawPlank(PI/2, glm::vec3(1.f, 0.f, 0.f), glm::vec3(0, -25, 25));
	DrawPlank(PI/2, glm::vec3(1.f, 0.f, 0.f), glm::vec3(0, 25, 25));
	DrawPlank(PI/2, glm::vec3(1.f, 0.f, 0.f), glm::vec3(0, 75, 25));

	// de sus
	DrawPlank(0.f, glm::vec3(0.f, 0.f, 0.f), glm::vec3(0, 100, 50));
	DrawPlank(0.f, glm::vec3(0.f, 0.f, 0.f), glm::vec3(0, 100, 100));
	DrawPlank(0.f, glm::vec3(0.f, 0.f, 0.f), glm::vec3(0, 100, 150));

	setTexCode(0);
}

void DrawMargin(glm::vec3 translation, glm::vec3 scale) {
	translationMat = glm::translate(glm::mat4(1.0f), translation);
	scaleMat = glm::scale(glm::mat4(1.0f), scale);
	myMatrix = translationMat * scaleMat;
	glUniformMatrix4fv(myMatrixLoc, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawArrays(GL_TRIANGLES, l1, l2);
}

void DrawMargins() {
	glUniform3f(objectColorLoc, 0.f, 0.f, 0.f);
	DrawMargin(glm::vec3(LPlank, 100, 50), glm::vec3(1.0f, 1.0f, 13.0f));
	DrawMargin(glm::vec3(-LPlank, 100, 50), glm::vec3(1.0f,1.0f,13.0f));

	DrawMargin(glm::vec3(LPlank, 25, 25), glm::vec3(1.0f, 6.0f, 1.0f));
	DrawMargin(glm::vec3(-LPlank,25, 25), glm::vec3(1.0f, 6.0f, 1.0f));

	DrawMargin(glm::vec3(LPlank, -50, -25), glm::vec3(1.0f, 1.0f, 6.0f));
	DrawMargin(glm::vec3(-LPlank, -50, -25), glm::vec3(1.0f, 1.0f, 6.0f));
}

void DrawBenchShadows() {
	glUniform1i(codColLocation, 1);
```

```
	DrawPlanks();
	DrawMargins();
	glUniform1i(codColLocation, 0);
}

void DrawFloor()
{
	const char* imageName = "grass.png";
	LoadTexture(imageName);
	glActiveTexture(GL_TEXTURE0);
	glBindTexture(GL_TEXTURE_2D, texture);
	glUniform1i(glGetUniformLocation(ProgramIdf, "myTexture"), 0);
	glUniform1i(glGetUniformLocation(ProgramIdv, "myTexture"), 0);

	translationMat = glm::translate(glm::mat4(1.0f), glm::vec3(0, 0, -8*unit));
	scaleMat = glm::scale(glm::mat4(1.0f), glm::vec3(1000.0f, 1000.0f, 0.1f));
	myMatrix = translationMat * scaleMat;
	glUniformMatrix4fv(myMatrixLoc, 1, GL_FALSE, &myMatrix[0][0]);

	setTexCode(1);
	glDrawArrays(GL_TRIANGLES, 0, 36);

	setTexCode(0);
}

void DrawLight()
{
	glUniform3f(objectColorLoc, 1.f, 1.f, 0.f);
	translationMat = glm::translate(glm::mat4(1.0f), glm::vec3(XLight, YLight, ZLight));
	myMatrix = translationMat;
	glUniformMatrix4fv(myMatrixLoc, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawArrays(GL_TRIANGLES, l1, l2);
}

void DrawSun() {
	glUniform3f(objectColorLoc, 1.f, 1.f, 0.f);
	float rotationAngle = 2 * PI * (float)Time / (float)MAX_MINS_IN_DAY + PI;
	rotationMat = glm::rotate(glm::mat4(1.0f), rotationAngle, glm::vec3(1, 0, 0));
	translationMat = glm::translate(glm::mat4(1.0f), SunInitPos);
	myMatrix = rotationMat * translationMat;
	SunPos = myMatrix * glm::vec4(SunAnchor, 1.0f);
	glUniformMatrix4fv(myMatrixLoc, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawArrays(GL_TRIANGLES, l1, l2);
}

void RenderFunction(void)
{
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST);

glm::vec3 Obs = glm::vec3(Obsx, Obsy, Obsz);
glm::vec3 PctRef = glm::vec3(Refx, Refy, Refz);
glm::vec3 Vert = glm::vec3(Vx, Vy, Vz);
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]);

// Variabile uniforme pentru iluminare
glUniform3f(objectColorLoc, 1.0f, 0.5f, 0.4f);
glUniform3f(lightColorLoc, 1.0f, 1.0f, 1.0f);
if (360 <= Time && Time < 1080) {
	glUniform3f(lightPosLoc, SunPos.x, SunPos.y, SunPos.z);
	setMatriceUmbra(SunPos.x, SunPos.y, SunPos.z);
} else {
	glUniform3f(lightPosLoc, XLight, YLight, ZLight);
	setMatriceUmbra(XLight, YLight, ZLight);
}
glUniform3f(viewPosLoc, Obsx, Obsy, Obsz);

switch (rendermode)
{
case Il_Frag:
	glUseProgram(ProgramIdf);
	l1 = 0; l2 = 36;
	break;
case Il_Frag_Av:
	glUseProgram(ProgramIdf);
	l1 = 36; l2 = 36;
	break;
case Il_Vert:
	glUseProgram(ProgramIdv);
	l1 = 0; l2 = 36;
	break;
case Il_Vert_Av:
	glUseProgram(ProgramIdv);
	l1 = 36; l2 = 36;
	break;
};

DrawPlanks();
DrawMargins();
DrawFloor();
```

```
	if (360 <= Time && Time < 1080) {
		DrawSun();
	} else {
		DrawLight();
	}

	DrawBenchShadows();

	glutSwapBuffers();
	glFlush();
}
void Cleanup(void)
{
	DestroyShaders();
	DestroyVBO();
}
int main(int argc, char* argv[])
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
	glutInitWindowPosition(100, 100);
	glutInitWindowSize(1200, 900);
	glutCreateWindow("3D Luminosity");
	glewInit();
	Initialize();
	glutIdleFunc(RenderFunction);
	glutDisplayFunc(RenderFunction);
	glutKeyboardFunc(processNormalKeys);
	glutSpecialFunc(processSpecialKeys);
	glutCreateMenu(menu);
	glutAddMenuEntry("Fragment", Il_Frag);
	glutAddMenuEntry("Fragment+Mediere_Normale", Il_Frag_Av);
	glutAddMenuEntry("Varfuri", Il_Vert);
	glutAddMenuEntry("Varfuri+Mediere_Normale", Il_Vert_Av);
	glutAttachMenu(GLUT_RIGHT_BUTTON);
	glutCloseFunc(Cleanup);
	glutMainLoop();

}
```