



# ProiectPG

Nume: Paul-Adrian Kovacs

Grupa: 30232

Email: kovacs\_paul\_adrian@yahoo.com

January 2023

## Contents

<b>1</b>	<b>Prezentarea temei</b>	<b>2</b>
<b>2</b>	<b>Scenariul</b>	<b>2</b>
2.1	Descrierea scenei și a obiectelor . . . . .	2
2.2	Funcționalități . . . . .	5
<b>3</b>	<b>Detalii de implementare</b>	<b>5</b>
3.1	Funcții și algoritmi . . . . .	5
3.1.1	Umbre . . . . .	5
3.1.2	Ploaie . . . . .	6
3.1.3	Animație moară . . . . .	6
3.1.4	Lumini . . . . .	7
3.1.5	Ceață . . . . .	8
3.1.6	Eliminarea fragmentelor . . . . .	8
3.1.7	Animație de prezentare . . . . .	8
3.2	Modelul grafic . . . . .	9
3.3	Ierarhia de clase . . . . .	9
<b>4</b>	<b>Prezentarea intrefetei grafice</b>	<b>10</b>
<b>5</b>	<b>Concluzii și dezvoltări ulterioare</b>	<b>10</b>

# 1 Prezentarea temei

In cadrul acestui proiect utilizat OpenGL pentru a crea o scena 3D. OpenGL este o interfață de programare a aplicațiilor (API) pentru crearea de conținut grafic 3D. Am utilizat Blender pentru a crea obiectele 3D din scena și pentru a le anima, apoi am exportat aceste obiecte în formatul .obj și .mtl, pentru a le putea utiliza în codul meu OpenGL.

Proiectul va fi implementat în Microsoft visual studio 2019 utilizând libraria header only GLM (OpenGL Mathematics) care ne conferă clase și funcții matematice bazate pe specificațiile GLSL (OpenGL Shading Language) [1] pentru a manipula matricile de transformare și GLFW (Graphics Library Framework) 3.3.2 pentru a crea și gestiona fereastra de afisare. GLFW ofera o interfață pentru a crea și gestiona o fereastră OpenGL și intrarea utilizatorului (mouse, tastatură), permitându-mi să interacționez cu scena și să o afisez pe ecran.

## 2 Scenariul

### 2.1 Descrierea scenei și a obiectelor

Scena reprezinta un sat medieval, religios axat pe creșterea de cai și cultivarea de grâu.

Pentru crearea terenului, am creat în blender un plan pe care l-am divizat în 80 de părți după care am folosit unealta de sculptare pentru a crea dealuri, iar apoi am aplicat o textura de iarbă.

Pentru crearea lămpilor am importat o lampă și am sterș vârfurile aflate în mijloc pentru a pastra doar rama, iar apoi am creat un cub pe care l-am texturat. Am făcut acest lucru pentru a folosi alt shader pentru a crea efectul de lumină.

Ploaia a fost implementată folosind un obiect de 2000 de ori, decimat pentru a avea un număr redus de vârfuri.

Moara a fost separată de elice pentru a se crea efectul de rotație.

Vegetație și moara prezintă fragmente care au fost eliminate



Figure 1: Moară



Figure 2: Case

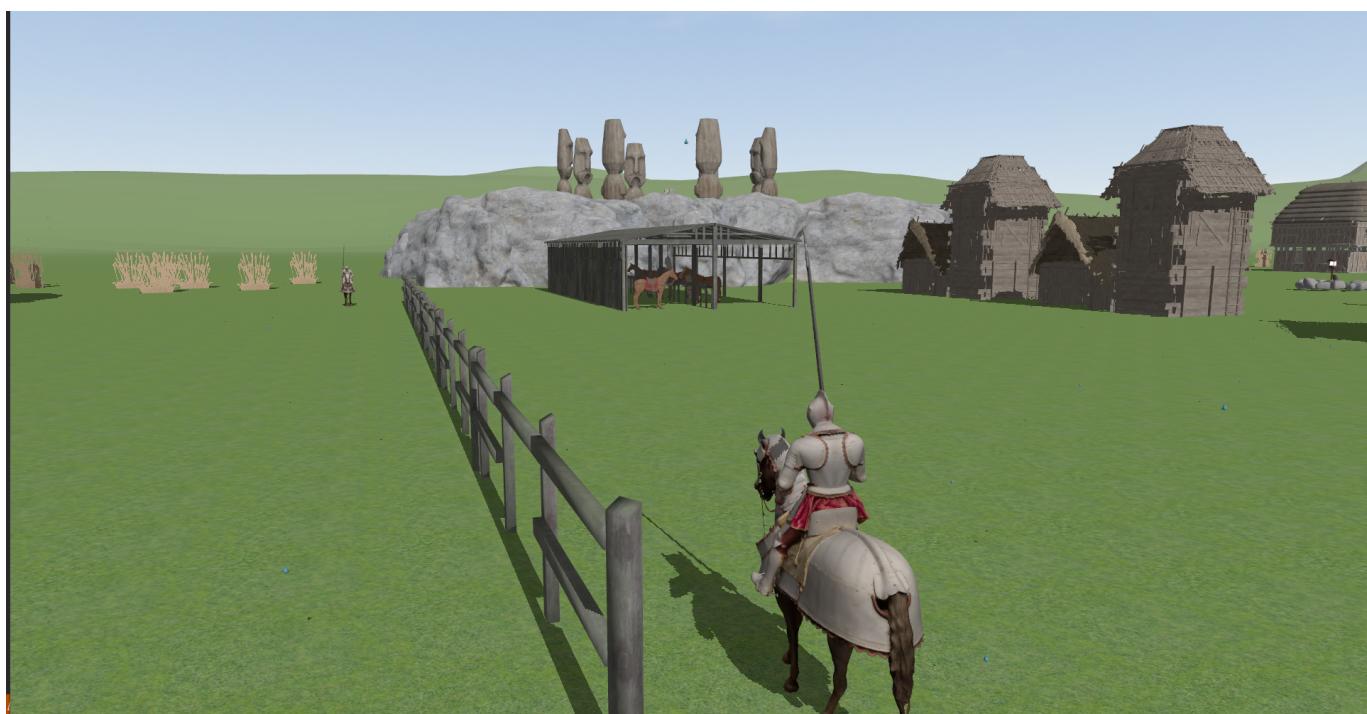


Figure 3: Cavaleri



Figure 4: Cai



Figure 5: Toteme

## 2.2 Funcționalități

1. Ploaie
2. Ceată
3. Animație moară
4. Lumini direcționale și punctiforme
5. Eliminare fragmente
6. Mod de vizualizare wireframe
7. Mod de vizualizare poligonal
8. Mod de vizualizare solid
9. Detectare proximitate cameră pentru a porni lămpi
10. Miscare și rotire cameră
11. Animație de prezentare a scenei

## 3 Detalii de implementare

### 3.1 Funcții și algoritmi

#### 3.1.1 Umbre

Calculez o hartă de adâncime din punctul de vedere al luminii, valorile ca nu se regăsesc în această hartă înseamnă ca sunt în umbră. Din cauza rezoluției limitate a hărții de adâncime, apare acnee când mai multe fragmente esantionează aceeași valoare din harta de adâncime. Rezolv această problema folosind modificând adâncimea fragmentului, prin o valoare bazată pe unghiul pe care suprafața îl face cu direcția luminii. În cazul în care avem valori în afara hărții de umbre cu valori  $> 1.0$  returnăm 0 pentru a le ilumina

Listing 1: Calcularea umbrelor

```
float computeShadow()
{
    // perform perspective divide
    vec3 normalizedCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;

    // Transform to [0,1] range
    normalizedCoords = normalizedCoords * 0.5 + 0.5;

    // Get closest depth value from light's perspective
    float closestDepth = texture(shadowMap, normalizedCoords.xy).r;

    // Get depth of current fragment from light's perspective
    float currentDepth = normalizedCoords.z;

    // Check whether current frag pos is in shadow
    float bias = max(0.05f * (1.0f - dot(fNormal, lightDir[0])), 0.005f);
    float shadow = currentDepth - bias > closestDepth ? 1.0f : 0.0f;

    if (normalizedCoords.z > 1.0f)
        return 0.0f;

    return shadow;
}

shadow = computeShadow();
vec3 color = min((ambient + (1.0f - shadow)*diffuse) + (1.0f - shadow)*specular, 1.0f);
```

### 3.1.2 Ploaie

Încarc modelul de picătură o singură dată și atribui result picăturilor acelasi model deoarece este mai rapid decât încărcarea a picăturii de n ori. Apoi generez poziția picaturilor folosind funcția rand().

Listing 2: Inițializare picaturi

```
water[0].LoadModel("models/water/water.obj");
srand(time(0));
for (int i = 1; i < 2000; i++)
{
    water[i] = water[i-1];
}

for (int i = 0; i < 2000; i++)
{
    float x = (rand() / (float)RAND_MAX) * 30 * 9 - 15;
    float y = (rand() / (float)RAND_MAX) * 10;
    float z = (rand() / (float)RAND_MAX) * 25 * 9 - 3;
    water_drops[i] = glm::vec3(x, y, z);
}
```

Pentru animația picăturilor voi traslata picăturile pe axa y până ajung sub plan, după care voi genera o poziție nouă pentru acestea

Listing 3: Animație picături

```
for (int i = 0; i < 2000; i++)
{
    water_drops[i].y -= 0.10f;
    if (water_drops[i].y < -2)
    {
        water_drops[i] = glm::vec3((rand() / (float)RAND_MAX) * 30 * 9 - 15 * 9, (rand() /
            (float)RAND_MAX) * 7, (rand() / (float)RAND_MAX) * 25 * 9 - 3 * 9);
    }

    model = glm::translate(glm::mat4(1.0f), water_drops[i]);
    model = glm::scale(model, glm::vec3(1/90.0f));

    glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE,
        glm::value_ptr(model));
    water[i].Draw(shader);
}
```

### 3.1.3 Animație moară

Calculez rotația elicei în funcție de timp pentru a menține o viteză constantă. Traslatez elicea în centru, o rotesc în funcție de axa z și o traslatez înapoi

Listing 4: Calculare delta

```
float delta = 0;
float movementSpeed = 10; // units per second
void updateDelta(double elapsedSeconds)
{
    delta = delta + movementSpeed * elapsedSeconds;
}
double lastTimeStamp = glfwGetTime();
```

Listing 5: Animație Moară

---

```

double currentTimeStamp = glfwGetTime();
updateDelta(currentTimeStamp - lastTimeStamp);
lastTimeStamp = currentTimeStamp;

model = glm::translate(model, glm::vec3(-0.374719f, 1.66209f, -0.749788f));
model = glm::rotate(model, glm::radians(delta), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::translate(model, glm::vec3(0.374719f, -1.66209f, 0.749788f));
glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE,
    glm::value_ptr(model));
windmill.Draw(shader);

```

---

### 3.1.4 Lumini

În cazul în care avem mai multe lumini vom calcula suma lor. De asemenea daca valoarea de enable pentru o lumină este 0 nu o vom lua în considerare.

Listing 6: Calcularea luminilor

---

```

void computeLightComponents()
{
    vec3 cameraPosEye = vec3(0.0f); //in eye coordinates, the viewer is situated at the origin

    //transform normal
    vec3 normalEye = normalize(fNormal);

    ambient = vec3(0.0f);
    diffuse = vec3(0.0f);
    specular = vec3(0.0f);

    for(int i = 0; i < NUMBER_OF_LIGHTS; i++)
    {
        if(lightEnable[i] == 0) continue;

        //compute light direction
        lightDirN = normalize(lightDir[i]);

        dist = length(lightDir[i]);
        if(i < 3) att = 1.0f / (constant + linear * dist + quadratic * (dist * dist));
        else att = 1.0f / (constant + linear * dist + 0.04f * (dist * dist));

        //compute view direction
        vec3 viewDirN = normalize(cameraPosEye - fPosEye.xyz);

        //compute ambient light
        ambient += att * ambientStrength * lightColor[i];

        //compute diffuse light
        diffuse += att * max(dot(normalEye, lightDirN), 0.0f) * lightColor[i];

        //compute specular light
        reflection = reflect(-lightDirN, normalEye);
        specCoeff = pow(max(dot(viewDirN, reflection), 0.0f), shininess);
        specular += att * specularStrength * specCoeff * lightColor[i];
    }
}

```

---

### 3.1.5 Ceață

Vom amesteca culoarea ceții cu aceea a fragmentului, pe baza distanței fragmentului. Luând în considerare reducerea intensității luminii în funcție de distanță. Factorul de atenuare care va fi folosit reprezintă densitatea de ceață, această densitate fiind constantă în toată scena. Rezultatul este o scădere rapidă a factorului de ceață. Valorile factorului de ceață trebuie să fie între 0.0 and 1.0.

Listing 7: Calcularea ceții

```
float computeFog()
{
    float fogDensity = 0.0005f;
    float fragmentDistance = length(fPosEye);
    float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));

    return clamp(fogFactor, 0.0f, 1.0f);
}
```

Listing 8: Amestecare culorii

```
float fogFactor = computeFog();
vec4 fogColor = vec4(0.5f, 0.5f, 0.5f, 1.0f);
fColor = fogColor * (1 - fogFactor) + vec4(color, 1.0f) * fogFactor;
```

### 3.1.6 Eliminarea fragmentelor

Îi spunem programului că va primi imagini RGBA glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, x, y, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, image\_data). Iar acum putem să eliminăm acele fragmente care au valoare alpha mai mică de 0.1. În plus am adăugat un enable pentru a putea dezactiva eliminarea pentru unele obiecte.

Listing 9: Eliminarea fragmentelor

```
if(enableDiscard == 1)
{
    vec4 colorFromTexture = texture(diffuseTexture, fTexCoords);
    if(colorFromTexture.a < 0.1)
        discard;
}
```

### 3.1.7 Animație de prezentare

Pentru animația de prezentare am creat o camera nouă. La apăsarea tastei de animație setez variabila presentation pe 1 care va dezactiva mouseul și tastatura pentru camera inițială și va initializa variabila steps și camera pe prima pozitie. Apoi în funcție de valoare steps deplasesc camera folosind funcția move din cameră. La anumite praguri ale valorii steps voi schimba poziția camerei și voi continua până la terminarea animației. Funcția progress este apelată din render scene și incrementează variabila steps.

Listing 10: Keyboard Callback

```
if (key == GLFW_KEY_4 && action == GLFW_PRESS)
{
    presentation = !presentation;
    startPresentation();
}
```

Listing 11: Animație de prezentare

```
int steps;
void startPresentation()
```

```

{
    steps = 0;
    myCameraPresentation.set(glm::vec3(15.645752f, 2.912375f, 70.467758f), glm::vec3(15.415703f,
        2.912375f, 69.494576f), glm::vec3(0.0f, 1.0f, 0.0f));
}

void progress()
{
    if (steps < 80)
    {
        myCameraPresentation.move(gps::MOVE_BACKWARD, cameraSpeed);
    }

    if(steps == 80) myCameraPresentation.set(glm::vec3(66.805054f, 16.160664f, 165.931976f),
        glm::vec3(65.867821f, 16.059607f, 165.598236f), glm::vec3(0.0f, 1.0f, 0.0f));

    if (steps > 80 && steps < 150)
    {
        myCameraPresentation.move(gps::MOVE_FORWARD, cameraSpeed);
    }

    if (steps == 150) myCameraPresentation.set(glm::vec3(81.684181f, 5.313303f, 34.770737f),
        glm::vec3(81.323334f, 5.198366f, 35.696255f), glm::vec3(0.0f, 1.0f, 0.0f));

    if (steps > 150 && steps < 220)
    {
        myCameraPresentation.move(gps::MOVE_FORWARD, cameraSpeed);
    }

    if (steps == 220) myCameraPresentation.set(glm::vec3(-146.814026f, 6.735712f, 113.727684f),
        glm::vec3(-145.862518f, 6.676405f, 113.425850f), glm::vec3(0.0f, 1.0f, 0.0f));

    if (steps > 220 && steps < 300)
    {
        myCameraPresentation.move(gps::MOVE_FORWARD, cameraSpeed);
    }

    if(steps == 300)
    {
        presentation = 0;
    }
    steps++;
}

```

---

### 3.2 Modelul grafic

Scena a fost realizată în blender apoi exportată în formatul .obj .mtl și încărcată folosind clasa Model3D. Operații de traslație, scalare și rotire au fost folosite pentru pozitionarea lor

### 3.3 Ierarhia de clase

1. Main - clasa principală a proiectului
2. Camera - deplasarea camerei
3. Model3D - incărcarea de obiecte
4. Mesh - desenare obiecte
5. Shader - gestionarea shaderelor

6. Window - crearea și gestionarea ferestrei prin care interacționăm
7. Skybox - incărcarea și desenarea skybox-ului

## 4 Prezentarea intrefetei grafice

nr.	tastă	efect
0	W	deplasare camară în față
1	A	deplasare camară în spate
2	S	deplasare camară la stânga
3	D	deplasare cameră la dreapta
4	LEFT CTRL	deplasare camară în jos
5	SPACE	deplasare camară în jos
6	Q	rotire umbre la stange
7	E	rotire ubmre la dreapta
8	Z	mod de vizualizare solid
9	X	mod de vizualizare poligonal
10	C	mod de vizualizare wireframe
11	1	dezactivare/activare lumina rosie
12	2	rotire la stânga a
13	3	rotire la dreapta a
14	4	prezentare scena
15	mouse	rotirea camerei

Table 1: Tastele folosite pentru operarea aplicației

## 5 Concluzii și dezvoltări ulterioare

Scena realizata ofera o simulare suficienta a realitatii conferind multiple efecte care sa adauge la complexitatea acesteia. Ulterior modificari la scena ar putea fi : crearea unei umbrelor in functie de toate luminile si nu doar de una singura, adaugarea mai multor obiecte cu o textura mai detaliata si realizarea unei ape realiste cu reflexie si valuri.

## References

- [1] <https://glm.g-truc.net/0.9.9/index.html> accesat pe 17 ianuaie, 2023  
Obiecte 3D
- [2] <https://www.cgtrader.com/free-3d-models?author=Enterables> accesat pe 17 ianuaie, 2023
- [3] <https://sketchfab.com/3d-models/japanese-standing-lantern-70aed3dc12274120809c97ee26e78fdb> accesat pe 17 ianuaie, 2023
- [4] <https://sketchfab.com/3d-models/camp-fire-b7967d9c16b64f2790b87e5ff8a80b52> accesat pe 17 ianuaie, 2023
- [5] <https://sketchfab.com/3d-models/rock-b66d5b63deb447299ca3effa904bc789> accesat pe 17 ianuaie, 2023
- [6] <https://sketchfab.com/3d-models/copy-of-aztec-god-quetzalcoatl-or-xolotl-4640a165d390486d9f42332cad2d1365> accesat pe 17 ianuaie, 2023
- [7] <https://sketchfab.com/3d-models/allan-pinkerton-electric-horse-acd1236756de4ef59cca7d84b7781e5e> accesat pe 17 ianuaie, 2023
- [8] <https://sketchfab.com/3d-models/196488-armor-for-man-and-horse-faca2ad557f64809b598c3a646a8804f> accesat pe 17 ianuaie, 2023

- [9] <https://sketchfab.com/3d-models/armored-horse-98c3f1c40a6b422dba76bf5403e0a3d8> accesat pe 17 ianuarie, 2023
- [10] <https://www.cgtrader.com/free-3d-models/exterior/exterior-public/low-poly-modular-old-wooden-fence> accesat pe 17 ianuarie, 2023

## List of Figures

1	Moară . . . . .	2
2	Case . . . . .	3
3	Cavaleri . . . . .	3
4	Cai . . . . .	4
5	Toteme . . . . .	4

## List of Tables

1	Tastele folosite pentru operarea aplicației . . . . .	10
---	---	----

## Listings

1	Calcularea umbrelor . . . . .	5
2	Inițializare picaturi . . . . .	6
3	Animație picături . . . . .	6
4	Calculare delta . . . . .	6
5	Animație Moară . . . . .	6
6	Calcularea luminilor . . . . .	7
7	Calcularea cetei . . . . .	8
8	Amestecare culorii . . . . .	8
9	Eliminarea fragmentelor . . . . .	8
10	Keyboard Callback . . . . .	8
11	Animație de prezentare . . . . .	8