

# **Proiect**

## **Prelucrare grafică**

Universitatea Tehnică din Cluj-Napoca  
Brînzoi Ion-Robert  
Grupa 30231

# Cuprins

<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	Functionalități . . . . .	3
<b>3</b>	<b>Detalii de implementare</b>	<b>3</b>
3.1	Funcții și algoritmi . . . . .	3
3.2	Modelul grafic . . . . .	9
3.3	Structuri de date . . . . .	9
3.4	Ierarhia de clase . . . . .	9
3.5	Manual de utilizare . . . . .	10
<b>4</b>	<b>Concluzii și dezvoltări ulterioare</b>	<b>11</b>

# 1 Prezentarea temei

Scopul proiectului constă în realizarea unei scene alcătuite din obiecte 3D folosind librăria OpenGL. Totodată, pentru poziționarea și aranjarea obiectelor a fost folosit programul Blender, ulterior exportând scena și anumite obiecte separat pentru a fi încărcată în Visual Studio și afișată pe ecran.

Am ales să implementez un mic sat înconjurat de dealuri, separat de un castel printre-un râu ce se varsă într-un lac. Pentru vizualizarea scenei complete, se pot folosi mouse-ul și tastatura pentru navigarea printre obiecte. Sunt prezente surse de iluminare diferite, umbre, animații și efecte pentru sporirea realismului.

## 2 Scenariul

### 2.1 Descrierea scenei și a obiectelor

La deschiderea scenei se poate observa castelul care se află peste un râu ușor transparent, fiind înconjurat de un zid prevăzut cu turnuri. În dreapta acestuia se află o pădure, iar lângă pădure am inclus un dragon așezat pe un deal. În fața pădurii se află un lac în care înăoată rațe, iar peste râu este localizat satul. Acesta are două case, o moară, o biserică, un grajd și un turn cu apă. În curtea caselor se pot observa berbeci, cai, un câine și o pisică, doi copaci, o masă și două scaune. În centrul satului se află un monument reflectorizant alcătuit din trei coloane. În spatele satului se poate observa sursa de lumină direcțională, iar pe dealul de lângă se află un alt tip de locuință, mai mare decât cele prezentate anterior.



Figura 1: Imagine de ansamblu asupra scenei.

## 2.2 Funcționalități

La încărcarea programului, utilizatorul se poate mișca prin scenă folosind tastatura și mouse-ul, permitându-se mișările camerei în sus, jos, stânga, dreapta, înainte și înapoi, precum și rotațiile acesteia. Pentru o prezentare de ansamblu, se poate apăsa un buton care va determina mișcarea automată a camerei prin scenă. Vizualizarea scenei poate fi realizată în trei moduri: solid, wireframe și punctiform.

Există șase surse de lumină: una direcțională, asupra satului, patru punctiforme, care iluminează turnurile castelului, și una de tip spotlight, aplicată asupra casei de pe deal. Exemplificarea generării umbrelor se face prin intermediul luminii direcționale.

Referitor la animații, elicea morii se învârte automat la deschiderea scenei, iar rațele de pe lac se mișcă în jurul unor curbe bezier generate aleator la fiecare rulare. Sunt și animații care pot fi comandate de la tastură, și anume coborârea și ridicarea podului castelului peste râu, deschiderea unor porți din sat, rotația sursei de lumină.

Am adăugat și efect de ceată în toată scena, al cărei densitate poate fi incrementată sau decrementată cu un buton, precum și ploaie deasupra satului.

## 3 Detalii de implementare

### 3.1 Funcții și algoritmi

#### Inițializarea scenei

Pentru rasterizarea obiectelor și maparea texturilor am utilizat funcțiile prezente în OpenGL, al căror funcționalitate a fost studiată pe parcursul semestrului. Totodată, în programul principal sunt definite metode pentru modularizarea codului, precum

- **mouseCallback()** - pentru implementarea mișărilor camerei cu ajutorul mouse-ului[1]
- **initObjects()**, **initShaders()**, **initUniforms()** - pentru inițializarea obiectelor, a shader-elor și a datelor trimise la shader din aplicație
- **initSkybox()** - inițializează fețele skybox-ului și trimite variabilele uniforme shader-ului acestuia
- **initPointlights()**, **initSpotLight()** - definirea surselor de lumină
- **sendPointLight(int index)** - trimite o lumină punctiformă la shader în funcție de poziția din vectorul de lumini, și este apelată de mai multe ori în **initUniforms()**

- **drawObjects(gps::Shader shader, bool depthPass)** - folosită pentru desenarea obiectelor cu ajutorul unui shader dat ca și parametru, și a unei variabile pentru setarea desenării umbrelor
- **drawDuck(), drawDroplet()** - funcții separate pentru desenarea obiectelor ce vor fi apelate de mai multe ori în funcția **drawObjects()**
- **renderScene()** - realizează randarea scenei finale
- **processMovement()** - folosită pentru preluarea comenzilor de la tastatură și executarea funcționalităților corespunzătoare

### **Generarea curberilor Bezier[2]**

La fiecare rulare a programului, se generează câte o curbă pentru fiecare rață ce va fi desenată, cu ajutorul următoarelor funcții:

- **generateBetween(float min, float max)** - returnează un float într-un interval dat
- **getRandomBezierCurve()** - returnează o structură de date corespunzătoare unei curbe bezier, având 4 puncte de control. Acestea sunt create folosind funcția de mai sus
- **initBezierCurves()** - inițializează toate curbele
- **getBezierPoint(GLfloat t, bezierCurve curve)** - folosind ecuația parametrică a curbei, returnează punctul corespunzător
- **getBezierDirectionVector(GLfloat t, bezierCurve curve)** - returnează vectorul tangent la curbă corespunzător parametrului t, folosind ecuația derivatei
- **changeBezierExtremities(bezierCurve\* curve)** - atunci când rața ajunge dintr-un capăt la celălalt se inversează extremitățile curbei, iar parametrul t devine nul, considerându-se astfel începutul unui nou traseu

Pentru orientarea rațelor de-a lungul curbei, am preluat mai întâi vectorul de direcție, l-am normalizat, iar apoi am calculat unghiul de rotație necesar, ca fiind arctangenta raportului dintre componenta z și componenta x a acestuia.

```

1 void drawDuck(gps::Shader shader, bool depthPass, int duckIndex) {
2     shader.useShaderProgram();
3     glm::mat4 modelAux = model;
4     modelAux = glm::translate(modelAux, getBezierPoint(t, curves.at(
5         duckIndex)));
6     glm::vec3 directionVector = glm::normalize(
7         getBezierDirectionVector(t, curves.at(duckIndex)));
8     float bezierAngle = glm::atan(directionVector.z, directionVector.
9         x);
10    bezierAngle = (bezierAngle * 180) / 3.14;
11    modelAux = glm::rotate(modelAux, glm::radians(270.0f - bezierAngle
12        ), glm::vec3(0, 1, 0));
13    glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"),
14        1, GL_FALSE, glm::value_ptr(modelAux));
15    if (!depthPass) {
16        normalMatrix = glm::mat3(glm::inverseTranspose(view * modelAux)
17            );
18        glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(
19            (glm::mat3(glm::inverseTranspose(normalMatrix)))));
20    }
21    ducks[duckIndex].Draw(shader);
22 }
```

Listing 1: Orientarea rațelor de-a lungul curbei

## Animații

Am definit și funcții care se ocupă cu mișcarea obiectelor: **bridgeMovement()**, **gateMovement()**, **rainMovement()**, **duckMovement()**. Acestea fie sunt apelate în **processMovement()**, pentru animațiile comandate de la tastatură, fie în **main()**, pentru cele de sine stătătoare, după verificarea unui interval de timp.

```

1 void bridgeMovement(float step) {
2     if (step > 0) {
3         if (bridgeAngle < 0 && bridgeAngle >= -90.0f)
4             bridgeAngle += step;
5     }
6     else {
7         if (bridgeAngle <= 0 && bridgeAngle > -90.0f)
8             bridgeAngle += step;
9     }
10 }
11 void gateMovement(float step) {
12     if (step > 0) {
13         if (gateAngle < 0 && gateAngle >= -90.0f)
14             gateAngle += step;
15     }
16     else {
17         if (gateAngle <= 0 && gateAngle > -90.0f)
18             gateAngle += step;
19     }
20 }
```

```

1 void rainMovement() {
2     for (int i = 0; i < DROPLET_NO; i++) {
3         rainDrops.at(i).moveCounter++;
4         if (rainDrops.at(i).position.y - rainDrops.at(i).moveCounter *
5             rainDrops.at(i).speed < 0.0f)
6             rainDrops.at(i).moveCounter = 0;
7     }
8 }
9 void duckMovement(float step) {
10    if (t < 1) {
11        t += step;
12    }
13    else {
14        for (int i = 0; i < DUCK_NO; i++)
15            changeBezierExtremities(&(curves.at(i)));
16        t = 0;
17    }
18 }
```

Listing 2: Funcțiile de mișcare a obiectelor

### Generarea umbrelor

Umbrele au fost generate prin tehnica hărților de umbre. Mai întâi scena va fi rasterizată din punct de vedere al luminii, ținându-se cont de valorile adâncimii, după care se rasterizează din punct de vedere al camerei prin compararea adâncimilor fragmentelor vizibile cu adâncimea din harta de umbre. Fragmentele cu adâncimea mai mare vor fi în umbră.[3]

### Generarea luminii

Pentru implementarea luminilor direcționale și punctiforme am utilizat resursele de laborator[3], iar cea de tip spotlight a fost implementată din resurse auxiliare.[4]

Pentru lumina punctiformă, calculez distanța de la fragment până la sursa de lumină, și cu ajutorul acesteia și a parametrilor constant, linear și quadratic am calculat atenuarea. Poziția camerei, direcția luminii, direcția de vizualizare, reflectia și coeficientul specular se calculează identic ca și în calculul luminii direcționale. Am definit vectori auxiliari pentru cele trei componente ale luminii, pe care i-am înmulțit cu factorul de atenuare și ulterior i-am compus, returnat și adunat la culoare în main-ul shader-ului doar dacă flag-ul care activează lumina punctiformă este activat.

```

1 vec3 computePointLight(pointLight light){ [...]
2     float dist = length(light.position - fPos.xyz);
3     float att = 1.0f / (light.constant + light.linear * dist + light.
4                           quadratic * dist * dist);
5     ambientAux *= att * texture(diffuseTexture, fTexCoords).rgb;
6     diffuseAux *= att * texture(diffuseTexture, fTexCoords).rgb;
7     specular *= att * texture(specularTexture, fTexCoords).rgb;
8     return (ambientAux + diffuseAux + specularAux); }
```

Listing 3: Calcularea luminii punctiforme

Pentru lumina de tip spot, vom avea nevoie de parametrii theta și epsilon, care vor fi folosiți în calcularea intensității. Theta ține cont de direcția luminii, adică vectorul orientat spre sursa de lumină din perspectiva fragmentului, și direcția luminii spot. Epsilon reprezintă diferența dintre cutOff și outerCutOff. CutOff se referă la raza luminii, și toate obiectele care se află în afara acestui unghi nu sunt iluminate. Parametrul outerCutOff este utilizat pentru ca lumina să scadă în intensitate gradual la margini.

```

1 vec3 computeSpotLight(){
2     vec3 lightDir = normalize(mySpotLight.position - fPos.xyz);
3
4     float theta = dot(lightDir, normalize(-mySpotLight.direction));
5     float epsilon = mySpotLight.cutOff - mySpotLight.outerCutOff;
6     float intensity = clamp((theta - mySpotLight.outerCutOff) /
7         epsilon, 0.0f, 1.0f);
8
9     vec3 ambientAux = spotLightAmbient * texture(diffuseTexture,
10        fTexCoords).rgb;
11    vec3 diffuseAux = spotLightDiffuse * intensity * texture(
12        diffuseTexture, fTexCoords).rgb;
13    vec3 specularAux = spotLightSpecular * intensity * texture(
14        specularTexture, fTexCoords).rgb;
15
16    return (ambientAux + diffuseAux + specularAux);
17 }
```

Listing 4: Calcularea luminii de tip spot



Figura 2: Scena cu luminile punctiforme și de tip spot activate.

## Efectul de ceată

Ceata va fi calculată în fragment shader, conform unei densități trimise ca și variabilă uniform, incrementată sau decrementată de la tastatură. Pentru a obține un rezultat mai bun și mai fotorealistic, am considerat factorul de ceată ca fiind exponentiaș pătratic. Aceasta se calculează ținând cont de distanța fragmentului și de densitatea ceții, iar în raport cu un factor liniar, scăderea factorului de ceată este mai rapidă.

```
1 float computeFog()
2 {
3     float fragmentDistance = length(fPosEye);
4     float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));
5     return clamp(fogFactor, 0.0f, 1.0f);
6 }
```

Listing 5: Calcularea ceții

## Efectul de ploaie

Pentru ploaie, similar cu generarea curbelor am generat puncte intr-un paralelipiped desupra satului, în care am translatat picăturile de ploaie. Pentru fiecare picătură am salvat o distanță ce va fi parcursă la fiecare apelare a funcției de mișcare a acesteia, precum și un contor pentru câte deplasări au avut loc. Fiecare distanță este diferită pentru fiecare picătură pentru accentuarea realismului. Acestea se inițializează cu ajutorul funcției `initDroplets()`. La momentul ajungerii pe pământ, acestea vor fi translatate înapoi la poziția inițială, continuând apoi mișcarea.

```
1 void initDroplets() {
2     for (int i = 0; i < DROPLET_NO; i++) {
3         rainDrop tmp;
4         tmp.position = glm::vec3(generateBetween(xRainMin, xRainMax),
5             generateBetween(yRainMin, yRainMin), generateBetween(zRainMin,
6             zRainMax));
7         tmp.speed = generateBetween(0.03f, 0.10f);
8         tmp.moveCounter = 0;
9         rainDrops.push_back(tmp);
10    }
11 }
```

Listing 6: Generarea picăturilor de ploaie

### 3.2 Modelul grafic

Obiectele și texturile au fost preluate de pe diverse site-uri [5][6], și textura pentru skybox la fel [7]. Obiectele au fost apoi importate în Blender și modificate corespunzător. În cazul unui obiect complex, acesta a fost separat în obiectele sale componente în funcție de materialele acestora. Obiectele care au fost animate, cele reflectorizante și cele transparente au fost exportate separat, iar celelalte obiecte care alcătuiesc scena au fost exportate împreună și încărcate în proiect[8].

### 3.3 Structuri de date

Pe lângă structurile definite în glm (vectori, matrice), am utilizat următoarele structuri de date:

- **bezierCurve** - structura pentru curba Bezier, alcătuită din 4 puncte de control de tip vec3
- **pointLight** - structura pentru lumina punctiformă, alcătuită din doi vectori care specifică poziția și culoarea, precum și cei trei parametrii constant, linear și quadratic
- **spotLight** - structura pentru lumina de tip spot, compusă din doi vectori pentru poziție și direcție, precum și parametrii cutOff și outerCutOff, pentru modificarea aspectului acestora
- **rainDrop** - structura ce cuprinde detaliile pentru o picătură de ploaie, și anume poziția inițială, distanța pe care o parcurge în jos după fiecare mutare, și numărul de mutări.

### 3.4 Ierarhia de clase

- **Camera** - cuprinde metodele de deplasare a camerei, pentru calcularea matricei de vizualizare și pentru prezentarea unei imagini de ansamblu a scenei
- **Model3D** - folosită pentru încărcarea obiectelor în scenă
- **Shader** - cuprinde metodele pentru a inițializa și pentru a activa un shader
- **SkyBox** - conține metodele de inițializare și de desenare a skybox-ului în care se află scena
- **Mesh** - conține informații despre vârfurile, indicii și texturile unui obiect, precum și metodele de rasterizare

### 3.5 Manual de utilizare

- 1 - vizualizare în mod solid
- 2 - vizualizare wireframe
- 3 - vizualizare punctiformă
- 4 - oprirea luminilor punctiforme
- 5 - pornirea luminilor punctiforme
- 6 - oprirea luminii de tip spot
- 7 - pornirea luminii de tip spot
- 9 - oprirea preview-ului camerei
- 0 - pornirea preview-ului camerei
- W - mișcarea camerei înainte
- A - mișcarea camerei la stânga
- S - mișcarea camerei înapoi
- D - mișcarea camerei la dreapta
- SPACE - mișcarea camerei în sus
- SHIFT - mișcarea camerei în jos
- E - rotirea camerei la dreapta
- Q - rotirea camerei la stânga
- B - coborârea podului
- N - urcarea podului
- K - mutarea luminii direcționale la stânga
- L - mutarea luminii direcționale la dreapta
- H - închiderea porților
- J - deschiderea porților
- F - creșterea densității cetei
- G - scăderea densității cetei
- R - pornirea ploii
- T - oprirea ploii

## 4 Concluzii și dezvoltări ulterioare

În concluzie, am reușit îndeplinirea cerințelor prezentate, implementând mai multe surse de lumină și exemplificând generarea umbrelor, a animațiilor și efectelor, dobândind astfel o înțelegere mai bună asupra librăriei OpenGL și a programului Blender. Partea care mi-a plăcut cel mai mult a fost implementarea curbelor Bezier, ilustrând astfel o aplicare a formulelor matematice în realitate.

Ca și dezvoltări ulterioare, se pot menționa:

- Adăugarea mai multor obiecte și animații
- Când rațele ajung la finalul traseului, să se rotească gradual înspre noua direcție în locul rotației imediate
- Adăugarea unui efect de vânt sau chiar o tornadă
- Generarea umbrelor generate de celelalte surse de lumină (punctiforme și de tip spot)

## Bibliografie

- [1] <https://learnopengl.com/Getting-started/Camera>.
- [2] <https://www.gamedeveloper.com/business/how-to-work-with-bezier-curve-in-games-with-unity>.
- [3] Lucrări de laborator.
- [4] <https://learnopengl.com/Lighting/Light-casters>.
- [5] <https://free3d.com/>.
- [6] <https://www.cgtrader.com/>.
- [7] <https://www.humus.name/index.php?page=Textures>.
- [8] [https://www.youtube.com/playlist?list=PLrgcDEgRZ\\_kndoWmRkAK4Y7ToJdOf-OSM](https://www.youtube.com/playlist?list=PLrgcDEgRZ_kndoWmRkAK4Y7ToJdOf-OSM).