



UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Scena interactiva in OpenGL

Prelucrare Grafica

Asistent de laborator: Constantin Nandra

Student: Stoica Mihai-Nicușor

Grupa: 30237

FACULTATEA DE AUTOMATICA
SI CALCULATOARE

2 Decembrie 2025

Cuprins

1 Descrierea Proiectului si Originea Datelor	2
2 Detalii de implementare	2
2.1 Modelul de Iluminare Multi-Sursa	2
2.2 Shadow Mapping	3
2.3 Calculul Matricei Normale	4
3 Efecte Vizuale si Post-Procesare	5
3.1 Ceata	5
3.2 Shaderul de Ploaie	5
3.3 Aberatia Cromatica	6
3.4 Skybox	6
3.5 Animatie	7
4 Controlul	7
4.1 Camera si Navigarea	7
4.2 Modul de Camera Orbita	7
4.3 Player	8
4.4 Moduri de Randare (Debugging)	8
4.5 Interfata de Control (Tabel)	8
5 Povestea Scenei	8
6 Concluzii	9
A Cod sursa	10

1 Descrierea Proiectului si Originea Datelor

Proiectul reprezinta o simulare grafica avansata a unei zone urbane reale, intitulata "Bulevardul Oltenia 34", dezvoltata utilizand API-ul OpenGL 4.1. Scena principală a fost construită pornind de la date geografice extrase prin **Google Maps API**, oferind o baza geometrică fidela pentru teren și structurile clădirilor. Pentru a popula acest mediu, au fost integrate modele 3D de înaltă fidelitate importate de pe platforma **Sketchfab** sau create în **Blender**.

Procesul de optimizare a fost unul critic: toate modelele au fost prelucrate în **Blender** pentru reducerea numărului de poligoane și re-maparea UV. Ulterior, s-a utilizat **Material Combiner** pentru a genera atlase de texturi (texture atlases), reducând astfel numărul de apeluri de desenare și reducând numărul de obiecte importante în scena.

2 Detalii de implementare

2.1 Modelul de Iluminare Multi-Sursă

Aplicația implementează un model de iluminare Blinn-Phong capabil să simuleze mai multe surse de lumina simultan. Sursele sunt stocate într-un array de structuri `lights[MAX_LIGHTS]` în fragment shader.

- **Structura PointLight:** Fiecare lumina contine vectori pentru pozitie și culoare, precum și coeficienți pentru atenuare.
- **Calculul Atenuării:** Intensitatea luminii scade pe măsură ce distanța ($dist$) față de sursă crește, utilizând formula:

$$att = \frac{1.0}{constant + linear \cdot dist + quadratic \cdot (dist \cdot dist)} \quad (1)$$

- **Componentele Iluminării:** Pentru fiecare fragment, se calculează suma vectorială a componentelor ambientale, difuze și speculare.
 - Componenta ambientală menține o luminozitate minima a texturii.
 - Componenta difuză utilizează produsul scalar dintre normala fragmentului și direcția luminii.
 - Componenta speculară simulează reflexia stralucitoare pe baza direcției de vizualizare și a hărții de specularitate.

```
1 void Scene::addLight(glm::vec3 pos, glm::vec3 color, float constant, float
2   ↵ linear, float quadratic) {
3   PointLight light;
4   light.position = pos;
5   light.color = color;
6   light.constant = constant;
7   light.linear = linear;
8   light.quadratic = quadratic;
9   lights.push_back(light);
10 }
11 void Scene::InitLightUniforms(gps::Shader& shader) {
12   shader.useShaderProgram();
13 }
```

```

14     lightLocs.resize(lights.size());
15
16     for (size_t i = 0; i < lights.size(); i++) {
17         std::string i_str = std::to_string(i);
18         std::string posName = "lights[" + i_str + "].position";
19         std::string colName = "lights[" + i_str + "].color";
20         std::string constName = "lights[" + i_str + "].constant";
21         std::string linName = "lights[" + i_str + "].linear";
22         std::string quadName = "lights[" + i_str + "].quadratic";
23
24         lightLocs[i].position = glGetUniformLocation(shader.shaderProgram,
25             posName.c_str());
25         lightLocs[i].color = glGetUniformLocation(shader.shaderProgram,
26             colName.c_str());
26         lightLocs[i].constant = glGetUniformLocation(shader.shaderProgram,
27             constName.c_str());
27         lightLocs[i].linear = glGetUniformLocation(shader.shaderProgram,
28             linName.c_str());
28         lightLocs[i].quadratic = glGetUniformLocation(shader.shaderProgram,
29             quadName.c_str());
29
30         glUniform3fv(lightLocs[i].color, 1, glm::value_ptr(lights[i].color));
31         glUniform1f(lightLocs[i].constant, lights[i].constant);
32         glUniform1f(lightLocs[i].linear, lights[i].linear);
33         glUniform1f(lightLocs[i].quadratic, lights[i].quadratic);
34     }
35
36     GLuint numLightsLoc = glGetUniformLocation(shader.shaderProgram,
37         "numLights");
37     glUniform1i(numLightsLoc, static_cast<int>(lights.size()));
38 }
```

2.2 Shadow Mapping

Sistemul de umbre este realizat prin tehnica Shadow Mapping, utilizand o harta de adancime (depth map) generata dintr-o trecere initiala de randare.

- **Framebuffer si Textura:** Se utilizeaza un `shadowMapFBO` si o textura de adancime `depthMapTexture` cu o rezolutie de 2048x2048 pixeli pentru a asigura margini clare ale umbrelor.
- **Transformarea Light Space:** Obiectele sunt randate din perspectiva sursei de lumina folosind o matrice de transformare `lightSpaceTrMatrix`, calculata prin combinarea unei proiectii ortografice cu o matrice de vizualizare orientata spre centrul scenei.

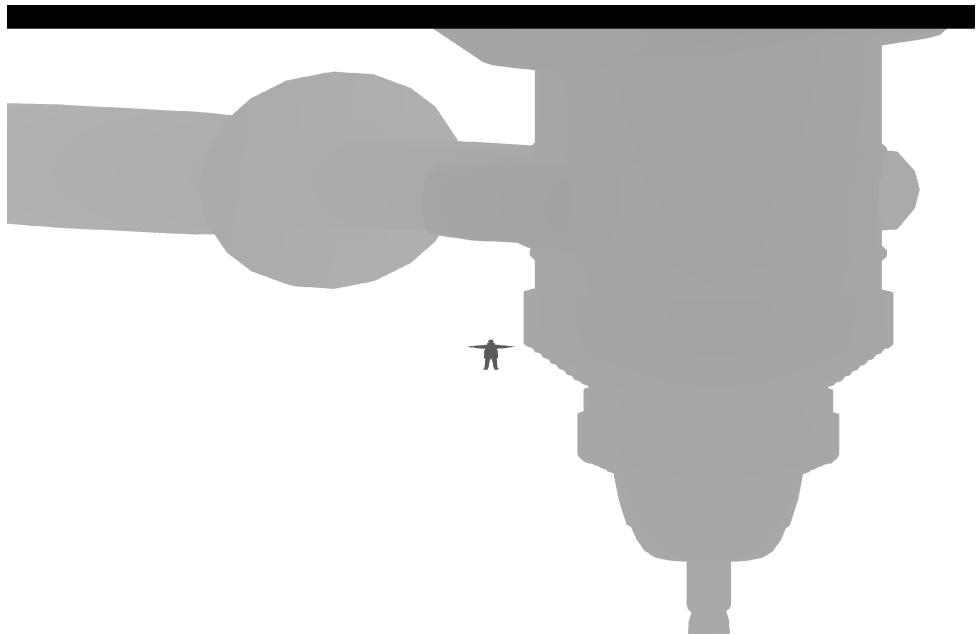


Figura 1: Vizualizare Shadowmap



Figura 2: Umbra playerului cazuta pe ochiul unitati EVA

2.3 Calculul Matricei Normale

Pentru a asigura ca iluminarea ramane corecta chiar si atunci cand obiectele sunt scalate neuniform sau rotite, aplicatia calculeaza Matricea Normala (*Normal Matrix*) la fiecare cadru pentru obiectele dinamice.

$$\text{NormalMatrix} = (\text{ModelView}^{-1})^T \quad (2)$$

Aceasta operatiune, realizata prin `glm::inverseTranspose(view * model)`, elimina efectele nedorite de scalare asupra vectorilor normali, asigurand calcule corecte in modelul Blinn-Phong.

3 Efecte Vizuale si Post-Procesare

3.1 Ceata

Pentru a imbunatati perceptia adancimii si atmosfera scenei nocturne, s-a implementat un efect de ceata calculat in spatiul Eye Space.

- **Formula Matematica:** Factorul de ceata (*fogFactor*) este determinat printr-o functie exponentiala patratica:

$$fogFactor = e^{-(fragmentDistance \cdot fogDensity)^2} \quad (3)$$

- **Parametri:** S-a utilizat o densitate de 0.003f si o culoare personalizata pentru a simula un amurg dens. Amestecul final dintre culoarea obiectului si ceata se realizeaza prin functia `mix`.



Figura 3: Ceata vizibila in jurul unei statui

3.2 Shaderul de Ploaie

Ploaia este simulata ca un efect de ecran utilizand un quad ce acopera intreg viewport-ul.

- **Animatie de Textura:** Shaderul `rain.frag` animeaza coordonatele UV ale texturii de ploaie in functie de variabila `time`, translatandu-le vertical pentru a simula caderea picaturilor.
- **Optimizare Fragment:** Se utilizeaza testul de alpha (alpha discard) pentru a elimina fragmentele transparente ale texturii, permitand vizualizarea scenei din spate.



Figura 4: Ploaie

3.3 Aberatia Cromatica

Pe langa ceata si ploaie, aplicatia include un efect de post-procesare optional: Aberatia Cromatica. Aceasta simuleaza imperfectiunea unei lentile reale, unde lungimile de unda ale luminiilor nu converg in acelasi punct focal.

- **Implementare:** Efectul este controlat prin variabila uniforma `caStrength`. Cand este activat, shader-ul primeste o valoare de `0.0025f`, care este utilizata pentru a decala coordonatele de texturare ale canalelor Rosu si Albastru in directii opuse fata de canalul Verde.
- **Rezultat:** Creeaza un efect de distorsiune a culorilor la marginile obiectelor, amplificand aspectul cinematic si usor haotic al scenei.

3.4 Skybox

Fundalul este reprezentat printr-un Skybox ce utilizeaza 6 texturi (fetele unui cub). Shader-ul dedicat pentru Skybox elimina componenta de translatie din matricea de vizualizare, asigurand ca cerul ramane la infinit indiferent de miscarea camerei.



Figura 5: Scena cu skybox in fundal

3.5 Animatie

Obiectul `drill` sufera o rotatie continua pe axa Y, independenta de restul scenei. Variabila `drillAngle` este incrementata la fiecare cadru si resetata la 360 de grade, matricea de model fiind actualizata constant pentru a reflecta aceasta miscare mecanica.

4 Controlul

4.1 Camera si Navigarea

Sistemul de camera este unul de tip First-Person, controlat prin input-uri de la tastatura si mouse.

- **Matematica Camerei:** Orientarea este definita prin vectorii `cameraFront`, `cameraRight` si `cameraUp`, recalculati la fiecare miscare a mouse-ului folosind functii trigonometrice bazate pe valorile *pitch* si *yaw*.
- **Viteza Variabila:** S-a implementat o functie de sprint care modifica `cameraSpeed` intre `BASE_CAMERA_SPEED` si `SPRINT_CAMERA_SPEED` atunci cand tasta `Left Shift` este apasata.

4.2 Modul de Camera Orbita

Pentru o vizualizare de ansamblu a scenei, a fost implementat un mod de camera automata ("Orbit Mode"), distinct de camera WASD.

- **Logica Matematica:** Camera se roteste automat in jurul unui punct central fixat la coordonatele (0, 10, 0). Pozitia camerei pe cercul de orbita este calculata folosind functii trigonometrice, unde raza (`orbitRadius`) poate fi ajustata dinamic:

$$x = \sin(\text{orbitAngle}) \cdot \text{orbitRadius} \quad (4)$$

$$z = \cos(\text{orbitAngle}) \cdot \text{orbitRadius} \quad (5)$$

- **Control:** Utilizatorul poate mari sau micsora raza orbitei (Zoom In/Out) folosind tastele **W** si **S** in timp ce acest mod este activ, permitand inspectarea detaliilor de la distanta sau de aproape fara a opri rotatia.

4.3 Player

Clasa **Player** gestioneaza randarea unui model 3D care urmareste pozitia camerei, fiind utilizat pentru a oferi un punct de referinta in scena si pentru a genera umbre proprii. Aceasta este influentata de o matrice de model ce include rotatii calculate in functie de unghiul de vizualizare al camerei pe axa Y (yaw).

4.4 Moduri de Randare (Debugging)

Pentru a verifica geometria si topologia modelelor importante, s-a implementat functionalitatea de schimbare a modului de poligon:

- **GL_FILL:** Modul standard, randare solida cu texturi si iluminare.
- **GL_LINE:** Modul Wireframe, util pentru a vedea densitatea poligoanelor si structura vertex-urilor.
- **GL_POINT:** Randeaza doar varfurile, util pentru debug-ul pozitiilor vertex-urilor.

4.5 Interfata de Control (Tabel)

Categorie	Input	Functie
<i>Navigare</i>	W / A / S / D	Deplasare camera (Fata / Spate / Stanga / Dreapta)
<i>Navigare</i>	Mouse Move	Controlul privirii (Pitch / Yaw)
<i>Navigare</i>	Left Shift	Sprint (viteza crescuta)
<i>Obiecte</i>	J / L	Rotire model Player
<i>Scena</i>	Q / E	Rotire globala a intregii scene pe axa Y
<i>Camera</i>	H	Activare/Dezactivare Mod Orbita
<i>Efecte</i>	R	Comutare efect ploaie (On/Off)
<i>Efecte</i>	I	Comutare Aberatie Cromatica (On/Off)
<i>Debug</i>	M	Vizualizare harta de adancime (Shadow Map)
<i>Debug</i>	K	Comtare tip Wireframe/Point/Solid
<i>Informatii</i>	P	Afisare coordonate camera in consola
<i>Sistem</i>	ESC	Inchidere aplicatie si eliberare resurse

Tabela 1: Configuratia completa a controalelor aplicatiei

5 Povestea Scenei

Ne aflam intr-un eveniment apocaliptic unde o strada din municipiul Craiova a fost trasa printre-un portal deasupra unui ocean portocaliu populat de statui Moai, sub o eclipsa. Aceasta teleportare a provocat prabusirea unui star destroyer intr-un bloc turn. Playerul, un om misterios cu masca si geaca de ploaie galbena, vine sa salveze orasul, care este atacat de Godzilla cu sabie laser si bormasina gigantica. Totul este acoperit de o ceata densa si rosie.

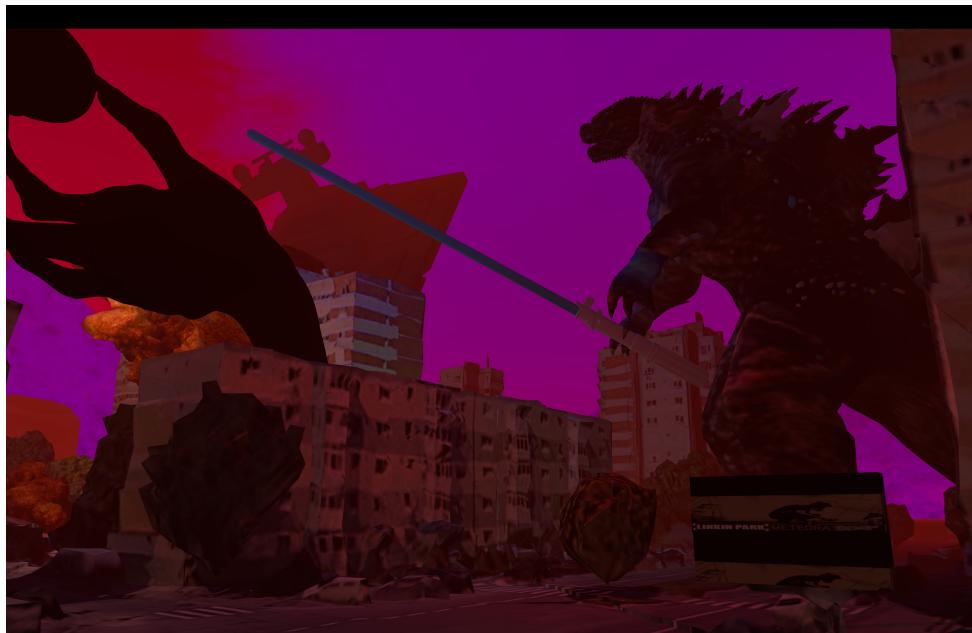


Figura 6: Godzilla in fata blocului



Figura 7: Playerul rotit cu fata spre camera

6 Concluzii

Aplicatia demonstreaza o integrare reusita a tehnicilor de randare in OpenGL. Prin utilizarea unui numar ridicat de surse de lumina si a efectelor de post-procesare, s-a reusit crearea unei atmosfere imersive.

A Cod sursa

<https://github.com/KOTerra/utcn-OpenGL-FirstPersonGoArounder>