

Documentație Proiect PG

Peisaj cu zăpadă

Czebely Stefania-Elena

Grupa 2

22.01.2026

Cuprins

1 Prezentarea temei	3
1.1 Obiectivele proiectului	3
1.2 Tehnologii utilizate	3
2 Scenariul	3
2.1 Descrierea scenei și a obiectelor	3
2.2 Sistemul de Iluminare	4
2.3 Funcționalități și Interacțiune	4
3 Detalii de implementare	4
3.1 Matematica din spatele iluminării (Blinn-Phong)	4
3.1.1 Componenta Ambientală	5
3.1.2 Componenta Difuză	5
3.1.3 Componenta Speculară (Blinn-Phong)	5
3.1.4 Atenuarea Luminii Punctiforme	5
3.2 Sistemul de Ceață (Fog)	5
3.3 Sistemul de Particule: Ploaia	6
3.3.1 Inițializare	6
3.3.2 Actualizare (Update Loop)	6
3.4 Animația (Vulturul)	6
4 Codul Sursă al Shaderelor	7
4.1 Vertex Shader (basic.vert)	7
4.2 Fragment Shader (basic.frag)	7
5 Rezultate Vizuale și Scenarii de Test	8
5.1 Scenariul 1: Iluminare Direcțională (Zi Însorită)	8
5.2 Scenariul 2: Iluminare Punctiformă (Point Light)	9
5.3 Scenariul 3: Condiții Atmosferice - Ploaia	9
5.4 Scenariul 4: Condiții Atmosferice - Ceață	10
5.5 Scenariul 5: Vedere de Ansamblu / Mod Prezentare	10
6 Manual de Utilizare	11
7 Concluzii și dezvoltări ulterioare	11
7.1 Concluzii	11
7.2 Dezvoltări ulterioare	12
8 Referințe	12

1 Prezentarea temei

1.1 Obiectivele proiectului

Prezentul proiect are ca scop dezvoltarea unei aplicații grafice tridimensionale complete, utilizând limbajul C++ și API-ul OpenGL (Open Graphics Library). Proiectul urmărește demonstrarea competențelor în utilizarea pipeline-ului grafic programabil, manipularea matricială și implementarea algoritmilor de iluminare și simulare fizică simplificată.

Obiectivul principal este crearea unei scene interactive care combină un mediu virtual detaliat, elemente dinamice (un vultur animat procedural) și efecte de mediu (ploaie, ceață, iluminare complexă). Aplicația trebuie să permită utilizatorului explorarea scenei în timp real și modificarea parametrilor de randare "on-the-fly".

1.2 Tehnologii utilizate

Pentru realizarea acestui proiect s-au utilizat următoarele biblioteci și instrumente:

- **OpenGL (Core Profile):** Pentru randarea grafică hardware-accelerată.
- **GLFW:** Pentru gestionarea ferestrelor, contextului OpenGL și input-ului (tastatură, mouse).
- **GLEW:** Pentru încărcarea extensiilor OpenGL moderne.
- **GLM (OpenGL Mathematics):** Pentru operații matematice vectoriale și matriceale necesare transformărilor 3D.
- **Visual Studio:** Ca mediu de dezvoltare integrat (IDE).

2 Scenariul

2.1 Descrierea scenei și a obiectelor

Universul virtual creat este compus dintr-un spațiu tridimensional definit de un sistem de coordinate dreptunghic. Scena se concentrează pe interacțiunea dintre lumină, atmosferă și elementele de zbor.

Deasupra scenei evoluează un vultur, care execută un zbor circular continuu. Aceasta adaugă dinamism scenei și servește drept reper pentru testarea umbrelor și a transformărilor de rotație/translație compuse, oferind un punct focal mobil pentru observator.

Mediul înconjurător este controlat de un sistem atmosferic dual:

1. **Sistem de ploaie:** Mii de particule care cad gravitațional, simulate prin primitive grafice de tip linie (`GL_LINES`).
2. **Sistem de ceață volumetrică:** O ceață densă care limitează vizibilitatea la distanță, calculată per-pixel în Fragment Shader.

2.2 Sistemul de Iluminare

Scena beneficiază de un sistem de iluminare hibrid, compus din trei tipuri de surse:

- **Lumina Direcțională (Sun Light):** O sursă infinită, plasată la distanță, care emite raze paralele. Aceasta definește iluminarea globală a scenei.
- **Lumina Punctiformă (Point Light):** O sursă locală (poziționată la coordonatele $-13.36, 11.18, -7.78$) care emite lumină omnidirecțională. Intensitatea acesteia scade odată cu pătratul distanței (atenuare pătratică).
- **Lumina Secundară Rotativă:** O sursă colorată (nuante de albastru) care orbitează în jurul centrului scenei, demonstrând calculul dinamic al vectorului de direcție a luminii.

2.3 Funcționalități și Interacțiune

Utilizatorul are control total asupra simulării prin periferice standard:

- **Navigare:** Sistem de cameră *Fly-Through* (WASD + Mouse).
- **Control Mediu:**
 - Tasta F: Activează/Dezactivează ceața.
 - Tastele [/]: Modifică densitatea ceții în timp real.
 - Tasta Y: Pornește/Oprește ploaia.
- **Control Iluminare:**
 - Tasta L: Comută culoarea luminii punctiforme.
 - Tasta N: Pornește/Oprește rotația luminii secundare.
- **Animație:**
 - Tasta P: Activează modul "Prezentare" (tur automat al camerei).

3 Detalii de implementare

Această secțiune detaliază algoritmii și fundamentele matematice utilizate în codul sursă.

3.1 Matematica din spatele iluminării (Blinn-Phong)

Iluminarea este calculată în *Fragment Shader* (`basic.frag`) folosind modelul Blinn-Phong. Acesta este o optimizare a modelului Phong, calculând vectorul median (*halfway vector*) în locul vectorului de reflexie complet.

Ecuatia generală pentru culoarea finală a unui fragment este:

$$I_{final} = I_{ambient} + I_{diffuse} + I_{specular} \quad (1)$$

Componentele sunt calculate astfel:

3.1.1 Componenta Ambientală

Simulează lumina indirectă care este împrăștiată peste tot în scenă.

$$I_{amb} = K_{amb} \cdot Color_{light} \quad (2)$$

3.1.2 Componenta Difuză

Simulează lumina direcțională care lovește suprafața. Depinde de unghiul dintre normala suprafetei (N) și direcția luminii (L).

$$Diff = \max(\dot(N, L), 0.0) \quad (3)$$

$$I_{diff} = Diff \cdot K_{diff} \cdot Color_{light} \quad (4)$$

3.1.3 Componenta Speculară (Blinn-Phong)

Simulează punctul strălucitor de pe obiecte lucioase. Se bazează pe vectorul median H dintre direcția luminii și direcția privirii (V).

$$H = \frac{L + V}{\|L + V\|} \quad (5)$$

$$Spec = (\max(\dot(N, H), 0.0))^{shininess} \quad (6)$$

3.1.4 Atenuarea Luminii Punctiforme

Pentru *Point Light*, intensitatea scade cu distanța (d). Formula implementată în cod (fișier `basic.frag`, funcția `computePointLight`) este:

$$Attenuation = \frac{1.0}{K_c + K_l \cdot d + K_q \cdot d^2} \quad (7)$$

Unde:

- K_c (constant) = 1.0
- K_l (linear) = 0.007
- K_q (quadratic) = 0.0002

3.2 Sistemul de Ceață (Fog)

Ceața este implementată folosind o funcție exponentială pătratică pentru a oferi o tranziție lină și realistă. Calculul se face în shaderul de fragment, pe baza distanței dintre cameră și fragmentul procesat.

Formula matematică:

$$f = e^{-(distance \cdot density)^2} \quad (8)$$

Implementare GLSL (basic.frag):

```
1 if (fogEnabled == 1) {  
2     float dist = length(cameraPos - fPosition);  
3     float fogFactor = exp(-pow((dist * fogDensity), 2.0));  
4     fogFactor = clamp(fogFactor, 0.0, 1.0);  
5     finalColor = mix(fogColor, finalColor, fogFactor);  
6 }
```

Variabila `fogDensity` este controlată din aplicația C++ prin tastele [si], permitând modificarea densității în timp real.

3.3 Sistemul de Particule: Ploaia

Ploaia este simulată printr-un sistem de particule simplificat. Nu se folosesc texturi sau shadere complexe de geometrie, ci primitive de tip linie (GL_LINES) pentru performanță maximă.

3.3.1 Inițializare

Se generează aleatoriu 10.000 de puncte într-un volum deasupra scenei ($X \in [-100, 100]$, $Z \in [-100, 100]$, $Y \in [20, 120]$). Acestea sunt stocate într-un `std::vector<glm::vec3>`.

3.3.2 Actualizare (Update Loop)

La fiecare cadru, în funcția `updateRain()` din C++: 1. Poziția Y a fiecărei picături scade cu o viteză constantă (`rainSpeed`). 2. Pozițiile X și Z sunt modificate în funcție de vectorul vântului. 3. **Reciclarea:** Dacă o picătură atinge solul ($Y < 0$), aceasta este mutată înapoi sus, la o poziție X/Z aleatorie nouă. Acest lucru creează iluzia unei ploi infinite folosind un număr finit de particule.

Secvență de cod C++ pentru actualizare:

```
1 void updateRain() {
2     for (auto& pos : rainPositions) {
3         // Simulare vant
4         float windX = cos(windDirection) * windStrength * deltaTime *
5             10.0f;
6         pos.x += windX;
7         // Gravitatie
8         pos.y -= rainSpeed * deltaTime * 20.0f;
9
10        // Resetare particula
11        if (pos.y < 0.0f) {
12            pos.y = (rand() % 100) + 50.0f;
13            pos.x = (rand() % 200) - 100.0f;
14        }
15    } // Update buffer GPU...
16 }
```

3.4 Animația (Vulturul)

Mișcarea vulturului este descrisă de ecuațiile parametrice ale cercului în planul XOZ. Poziția este recalculate la fiecare frame în funcție de timpul scurs (`deltaTime`).

$$X_{eagle} = R \cdot \cos(\theta) \quad (9)$$

$$Z_{eagle} = R \cdot \sin(\theta) \quad (10)$$

Unde θ (unghiul) crește constant. Pentru ca vulturul să fie orientat corect (tangent la traекторie), se aplică o rotație suplimentară modelului:

```
1 // Rotatia modelului pentru a privi în direcția de zbor
2 eagleModel = glm::rotate(eagleModel, glm::radians(-eagleRotationAngle -
3     90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
```

4 Codul Sursă al Shaderelor

Shaderele reprezintă inima procesării grafice în acest proiect. Iată implementarea completă.

4.1 Vertex Shader (basic.vert)

Acesta transformă coordonatele din spațiul obiectului în spațiul de tăiere (Clip Space) și pregătește datele pentru fragment shader.

```
1 #version 410 core
2
3 layout(location=0) in vec3 vPosition;
4 layout(location=1) in vec3 vNormal;
5 layout(location=2) in vec2 vTexCoords;
6
7 out vec3 fPosition;
8 out vec3 fNormal;
9 out vec2 fTexCoords;
10
11 uniform mat4 model;
12 uniform mat4 view;
13 uniform mat4 projection;
14 uniform mat3 normalMatrix;
15
16 void main()
17 {
18     // Calculam pozitia in World Space
19     vec4 worldPos = model * vec4(vPosition, 1.0f);
20     fPosition = worldPos.xyz;
21
22     // Normala in World Space
23     fNormal = normalize(normalMatrix * vNormal);
24
25     fTexCoords = vTexCoords;
26
27     gl_Position = projection * view * worldPos;
28 }
```

4.2 Fragment Shader (basic.frag)

Aici se calculează iluminarea (Blinn-Phong) și efectul de ceată.

```
1 #version 410 core
2
3 in vec3 fPosition;
4 in vec3 fNormal;
5 in vec2 fTexCoords;
6
7 out vec4 fColor;
8
9 uniform sampler2D diffuseTexture;
10 uniform sampler2D specularTexture;
11
12 // Variabile uniforme pentru lumini si ceata...
13 // (Codul complet pentru computeDirLight, computePointLight
```

```

14 // a fost detaliat în secțiunea 3)
15
16 void main()
17 {
18     vec3 normal = normalize(fNormal);
19     vec3 viewDir = normalize(cameraPos - fPosition);
20
21     vec3 light1 = computeDirLight(lightDir, lightColor, normal, viewDir);
22     vec3 light2 = computeDirLight(secondLightDir, secondLightColor,
23                                   normal, viewDir);
24     vec3 light3 = computePointLight(normal, viewDir);
25
26     vec3 finalColor = light1 + light2 + light3;
27
28     // Aplicare ceata
29     if (fogEnabled == 1) {
30         float dist = length(cameraPos - fPosition);
31         float fogFactor = exp(-pow((dist * fogDensity), 2.0));
32         fogFactor = clamp(fogFactor, 0.0, 1.0);
33         finalColor = mix(fogColor, finalColor, fogFactor);
34     }
35
36     fColor = vec4(finalColor, 1.0f);
37 }
```

5 Rezultate Vizuale și Scenarii de Test

În această secțiune sunt prezentate capturi de ecran din aplicație care demonstrează funcționalitățile implementate în diferite scenarii.

5.1 Scenariul 1: Iluminare Direcțională (Zi Însorită)

În acest scenariu, doar lumina principală (Soarele) este activă. Se observă umbrele proprii ale obiectelor și reflexiile speculare puternice pe suprafetele luminate.



Figura 1: Scena iluminată preponderent de lumina direcțională.

5.2 Scenariul 2: Iluminare Punctiformă (Point Light)

Aici este evidențiată lumina locală (becul virtual). Se observă cum intensitatea luminii scade pe suprafața solului pe măsură ce distanța față de sursă crește (atenuare pătratică).



Figura 2: Efectul luminii punctiforme asupra geometriei.

5.3 Scenariul 3: Condiții Atmosferice - Ploaia

Sistemul de particule este activat. Liniile albastre semi-transparente simulează căderea precipitațiilor. Vulturul este surprins în zbor în timp ce plouă.



Figura 3: Simularea ploii folosind primitive GL_LINES.

5.4 Scenariul 4: Condiții Atmosferice - Ceață

Densitatea ceții a fost crescută. Se observă cum obiectele îndepărtațe devin greu vizibile, culoarea lor contopindu-se cu culoarea de fundal a ceții. Acest efect adaugă profunzime și atmosferă scenei.



Figura 4: Efectul de ceată densă (Fog Density crescut).

5.5 Scenariul 5: Vedere de Ansamblu / Mod Prezentare

Această captură prezintă o vedere generală asupra scenei, surprinsă în timpul animației automate de tip "Prezentare", evidențiind complexitatea ansamblului și mișcarea vulturului.



Figura 5: Vedere de ansamblu a scenei dinamice.

6 Manual de Utilizare

Aplicația nu dispune de meniuri grafice (GUI) pentru a nu afecta performanța și imersiunea, bazându-se exclusiv pe comenzi rapide de la tastatură.

Categorie	Tastă	Descriere
Cameră	W, A, S, D	Deplasare orizontală (Fly Mode).
	Space	Ascensiune verticală.
	L_Ctrl	Coborâre verticală.
	Mouse	Orientarea privirii.
Atmosferă	F	ON/OFF Ceață.
	[/]	Modificare densitate ceață (- / +).
	Y	ON/OFF Ploaie.
Lumini	L	Schimbă culoarea Point Light (Albastru → Alb → Stins).
	N	Rotește lumina secundară.
Diverse	P	Mod Prezentare (Camera automată).

Tabela 1: Lista completă a comenziilor disponibile.

7 Concluzii și dezvoltări ulterioare

7.1 Concluzii

Proiectul a reprezentat un exercițiu complex de sinteză a cunoștințelor de grafică computerizată. Prin implementarea de la zero (folosind OpenGL și C++) a sistemelor de

iluminare și particule, s-a obținut o înțelegere profundă a modului în care GPU-ul procesează vertecșii și pixelii.

Cele mai importante realizări tehnice sunt:

1. Implementarea corectă a modelului de iluminare Blinn-Phong cu multiple surse de lumină (Direcțională, Punctiformă).
2. Crearea unui sistem de particule (Ploaia) care rulează pe CPU și este randat eficient pe GPU.
3. Integrarea efectului de ceată volumetrică direct în fragment shader, permitând o atmosferă dinamică.

7.2 Dezvoltări ulterioare

Pentru îmbunătățirea realismului vizual, proiectul poate fi extins în următoarele direcții:

- **Shadow Mapping:** Implementarea umbrelor dinamice pentru a oferi indicii vizuale spațiale mai bune.
- **Normal Mapping:** Adăugarea de detalii pe suprafața obiectelor fără a crește numărul de poligoane.
- **Skybox:** Adăugarea unui cub texturat care să simuleze cerul și orizontul îndepărtat.
- **Geometry Shaders:** Mutarea logicii de generare a ploii de pe CPU pe GPU pentru a putea randa milioane de particule simultan.

8 Referințe

Documentarea și implementarea acestui proiect s-au bazat pe următoarele resurse bibliografice și online:

1. **Videoclipurile Profesorului (Playlist YouTube):** Suport video curs și laborator.
https://www.youtube.com/playlist?list=PLrgcDEgRZ_kndoWmRkAK4Y7ToJdOf-0SM
2. **Materiale de curs:** Suportul de curs pentru disciplina Prelucrare Grafică (PG), Universitatea Tehnică din Cluj-Napoca.
3. **OpenGL Documentation:** Documentația oficială Khronos Group.
<https://www.opengl.org/documentation/>
4. **LearnOpenGL:** Tutoriale extensive despre iluminare și shadere.
<https://learnopengl.com/>
5. **GLM Manual:** Manualul de utilizare pentru biblioteca matematică.
<https://glm.g-truc.net/0.9.9/api/index.html>