UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Sat Medieval 3D – Aplicație OpenGL

Prelucrare Grafică

Autor: Bozîntan Carla Daniela
Grupa: 30236
Profesor îndrumător: Constantin Nandra
Data:

Facultatea de
Automatică și
Calculatoare

Cuprins

1. Prezentarea temei

Tema proiectului constă în realizarea unei **aplicații grafice 3D interactive** utilizând biblioteca **OpenGL modern (core profile)**, având ca scop construirea și vizualizarea unei scene virtuale complexe. Proiectul urmărește aplicarea practică a noțiunilor studiate în cadrul disciplinei *Prelucrare Grafică*, precum transformările geometrice, iluminarea, texturarea, animația și efectele vizuale.

Scena realizată reprezintă un **sat medieval**, alcătuit din mai multe obiecte 3D texturate și iluminate, integrate într-un mediu coerent din punct de vedere vizual. Aplicația permite utilizatorului să exploreze scena prin deplasarea camerei și să observe efectele grafice implementate în timp real.

2. Scenariul

2.1. Descrierea scenei și obiectelor

Scena virtuală reprezintă un sat medieval amplasat într-un spațiu deschis, având ca element central un teren extins peste care sunt plasate clădiri și obiecte specifice epocii medievale. Mediul este completat de un cer redat printr-un skydome, care oferă impresia unui spațiu infinit.

Printre obiectele componente ale scenei se regăsesc:

- un castel medieval, poziționat la distanță, care oferă profunzime scenei;
- biserică și un turn, elemente arhitecturale dominante;
- mai multe case medievale și un han, grupate într-o zonă de tip sat;
- o taverna utilizata pentru iluminare de punctiforma;
- o statuie, pentru diversitate vizuală;
- un copac animat, utilizat și ca sursă de iluminare de tip spotlight;
- un teren texturat, care susține toate obiectele scenei.

Toate modelele sunt încărcate din fișiere .obj și sunt poziționate, scalate și rotite astfel încât să creeze o compoziție vizuală coerentă.

2.2. Funcționalități

Aplicația oferă următoarele funcționalități:

- deplasarea camerei în scenă (W, A, S, D);
- rotația camerei folosind mouse-ul;

- schimbarea modului de randare (solid, wireframe, puncte);
- iluminare direcțională;
- iluminare de tip spotlight pentru evidențierea copacului si a semnului tavernei;
- animația copacului (rotație);
- efect de ceață atmosferică dependentă de distanță;
- randarea cerului folosind skydome.

3. Detalii de implementare

3.1. Funcții și algoritmi

Aplicația este organizată în funcții clare, fiecare având un rol bine definit în pipeline-ul grafic:

- inițializarea ferestrei și a contextului OpenGL;
- încărcarea modelelor 3D;
- compilarea și utilizarea shaderelor;
- procesarea inputului de la tastatură și mouse;
- actualizarea animațiilor și a particulelor;
- randarea scenei.

3.1.1. Soluţii posibile

Pentru implementarea scenei au fost analizate mai multe variante:

- utilizarea unui skybox sau a unui skydome pentru cer;
- iluminare Phong sau Blinn-Phong;
- ceață liniară sau ceață exponențială;
- animații simple sau sisteme complexe de particule.

3.1.2. Motivarea abordării alese

S-a ales utilizarea skydome-ului deoarece acesta oferă o tranziție mai naturală a texturii cerului. Modelul de iluminare Blinn-Phong a fost preferat datorită eficienței și aspectului vizual plăcut. Ceața exponențială este mai realistă, deoarece intensitatea acesteia crește gradual odată cu distanța. Sistemul de ploaie este implementat simplu, pentru a păstra performanța aplicației.

3.2. Modelul grafic

Modelul grafic utilizat respectă pipeline-ul OpenGL modern:

- transformările sunt realizate prin matricile model, view și projection;
- iluminarea și texturarea sunt calculate în fragment shader;
- testul de adâncime este utilizat pentru afișarea corectă a obiectelor;
- face culling este activ pentru optimizarea randării.

Pentru cer, matricea de view este utilizată fără componenta de translație, astfel încât cerul să rămână fix față de cameră.

3.3. Structuri de date

În implementarea aplicației sunt utilizate mai multe structuri de date, atât la nivel de aplicație C++ (CPU), cât și la nivel de shadere (GPU), pentru a susține randarea corectă a scenei și calculul efectelor grafice.

Vectori și matrici (GLM)

Pentru reprezentarea pozițiilor, direcțiilor și transformărilor geometrice sunt utilizate tipurile oferite de biblioteca GLM:

- glm::vec3 – pentru poziții 3D, direcții de iluminare și normale;
- glm::vec2 – pentru coordonate de textură;
- glm::mat4 – pentru matricile de transformare model, view și projection.

Aceste structuri sunt utilizate atât în aplicația principală, cât și în shadere, unde matricile sunt transmise ca uniforme pentru a calcula poziția fragmentelor în spațiul camerei (eye-space).

Structura pentru surse de lumină (SpotLight)

Pentru implementarea iluminării de tip spotlight a fost definită o structură de date în fragment shader, care grupează toate informațiile necesare unei surse de lumină focalizate:

```
struct SpotLight
{
    vec3 position;
    vec3 direction;
    float cutOff;
```

```
    float outerCutOff;
    vec3 color;
};
```

Această structură permite:

- definirea poziției sursei de lumină;
- specificarea direcției de iluminare;
- controlul unghiului interior și exterior al conului de lumină;
- personalizarea culorii luminii.

Structura este utilizată în algoritmul de calcul al iluminării de tip spotlight, unde intensitatea luminii este determinată în funcție de unghiul dintre direcția luminii și direcția către fragment.

Date pentru calculul ceții atmosferice

Efectul de ceață este implementat în fragment shader, utilizând poziția fragmentului în spațiul camerei (fragPosEye), care este transmisă din vertex shader. Distanța față de cameră este calculată folosind funcția length():

```
float distance = length(fragPosEye);
```

Pe baza acestei distanțe, este aplicată o formulă de ceață exponențială, care determină factorul de amestec între culoarea obiectului și culoarea ceții. Astfel, structurile de date vec3 și vec4 sunt utilizate pentru:

- poziția fragmentului;
- culoarea finală a pixelului;
- culoarea ceții.

Această abordare permite realizarea unui efect de ceață dependent de distanță, care crește gradual odată cu îndepărtarea obiectelor față de cameră.

Variabile globale pentru starea scenei

Aplicația utilizează variabile globale pentru a păstra starea curentă a scenei:

- poziția și orientarea camerei;
- matricile de transformare;
- parametrii de iluminare;
- modul de randare selectat.

Aceste variabile facilitează comunicarea dintre diferitele componente ale aplicației și permit actualizarea coerentă a scenei în fiecare cadru randat.

3.4. Ierarhia de clase

Aplicația este organizată modular, folosind mai multe clase C++, fiecare având un rol bine definit în gestionarea ferestrei, a camerei, a modelelor 3D, a shaderelor și a resurselor grafice. Această structurare permite separarea clară a responsabilităților și facilitează extinderea și întreținerea codului.

Clasa Window

Clasele Window.h și Window.cpp sunt responsabile pentru:

- crearea ferestrei aplicației utilizând GLFW;
- inițializarea contextului OpenGL;
- gestionarea dimensiunilor ferestrei;
- accesul la fereastra GLFW din restul aplicației.

Această clasă abstractizează detaliile legate de fereastră și permite utilizarea simplă a contextului grafic în main.cpp.

Clasa Camera

Clasele Camera.hpp și Camera.cpp gestionează:

- poziția camerei în scenă;
- direcția de vizualizare;
- vectorul „up”;
- calculul matricei view.

Clasa oferă metode pentru deplasarea și rotirea camerei în funcție de input-ul de la tastatură și mouse, permițând explorarea interactivă a scenei 3D.

Clasele Shader

Clasele Shader.hpp și Shader.cpp sunt utilizate pentru:

- încărcarea fișierelor de tip vertex shader și fragment shader;
- compilarea și linkarea programelor de shader;
- activarea shaderelor în timpul randării;
- trimiterea uniformelor către GPU.

Această clasă este esențială pentru implementarea iluminării, a ceții și a efectelor atmosferice, deoarece centralizează utilizarea shaderelor.

Clasele Mesh

Clasele Mesh.hpp și Mesh.cpp definesc structura de bază a unui mesh 3D. Acestea gestionează:

- datele geometrice (vârfuri, normale, coordonate de textură);
- buffer-ele OpenGL (VAO, VBO, EBO);
- desenarea efectivă a mesh-ului.

Clasa Mesh reprezintă unitatea de bază pentru randarea modelelor 3D.

Clasele Model3D

Clasele Model3D.hpp și Model3D.cpp sunt responsabile pentru:

- încărcarea modelelor 3D din fișiere .obj;
- gestionarea mai multor mesh-uri asociate unui model;
- aplicarea texturilor;
- desenarea completă a modelului folosind shaderele active.

Această clasă utilizează clasa Mesh și biblioteca tiny_obj_loader pentru a încărca și procesa modelele 3D.

Fișierul main.cpp

Fișierul main.cpp reprezintă punctul de intrare al aplicației și coordonează:

- inițializarea ferestrei și a contextului OpenGL;
- încărcarea modelelor și a shaderelor;
- inițializarea efectelor vizuale (ploaie, ceață);
- procesarea input-ului utilizatorului;
- randarea scenei în bucla principală a aplicației.

Acesta integrează toate clasele descrise anterior într-un flux coerent de execuție.

Relația dintre clase

Relațiile dintre clase pot fi rezumate astfel:

- main.cpp utilizează Window, Camera, Shader și Model3D;

- Model3D utilizează unul sau mai multe obiecte de tip Mesh;
- Mesh gestionează buffer-ele OpenGL;
- Shader este utilizat de toate clasele care realizează randarea;
- bibliotecile stb_image și tiny_obj_loader sunt utilizate pentru încărcarea resurselor externe.

Această organizare modulară contribuie la claritatea codului și la separarea responsabilităților.

4. Prezentarea interfeței grafice utilizator / Manual de utilizare

Interacțiunea cu aplicația se realizează prin:

- W, A, S, D – deplasarea camerei;
- Mouse – rotirea camerei;
- 1, 2, 3 – schimbarea modului de randare;
- ESC – închiderea aplicației.

Interfața este minimalistă, accentul fiind pus pe explorarea scenei.

5. Concluzii și dezvoltări ulterioare

Proiectul demonstrează aplicarea corectă a conceptelor fundamentale de prelucrare grafică într-o aplicație 3D interactivă. Scena creată oferă un mediu coerent, cu iluminare, animație și efecte atmosferice.

- Posibile dezvoltări ulterioare includ:
- implementarea unui ciclu zi-noapte;
- adăugarea umbrelor dinamice;
- optimizarea randării prin instancing;
- animații mai complexe ale obiectelor.

6. Referințe

[1] OpenGL Programming Guide
[2] https://learnopengl.com
[3] GLFW Documentation
[4] GLM Documentation
[5] Materiale de laborator – Prelucrare Grafică