UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea: Automatica si Calculatoare
Specializare: Calculatoare si Tehnologia Informatiei
Disciplina: Prelucrare grafica

Proiect de semstru: Chernobyl

Student

Cozma Victoria

Grupa 30225

Îndrumător

Prof. Nandra
Constantin Ioan

12 ianuarie 2021

Cuprins

1. Prezentarea temei

Acest proiect de semestru se bazează pe folosirea motorului grafic OpenGL pentru a crea o scenă 3D, folosind pentru a modela şi poziţiona obiectele din această scenă în aplicaţia Blender.

2. Scenariul

2.1. Descrierea scenei şi a obiectelor

Concret, scena prezentată prezintă centrala nucleară de la Cernobîl, locul unde în 1986 în aprilie a avut loc probabil cel mai cunoscut accident nuclear din lume. Reactorul 4 al centralei s-a supraîncălzit şi a explodat, expulzând în împrejurimi şi în atmosferă cantităţi masive de materiale radioactive. Cu toate că au trecut doar 35 de ani de atunci, zona va fi complet sigură şi lipsită de radiaţie abia peste 20.000 de ani. Bazându-mă pe acest motiv, am decis să creez zona Cernobîlului şi a oraşului Pripyat pentru a putea prezenta evenimentele întâmplate într-o manieră 100% sigură.

Astfel, scena se învârte în jurul centralei nucleare, care este poziţionată în mijlocul acesteia. Iniţial, plecăm de la intrarea în Pripyat şi trecem printr-o pădure în care întâlnim o multitudine de semne de avertisment. Ajungând la centrală, pe lângă aceasta se găsesc containere şi butoaie cu material radioactiv, toate marcate corespunzător cu acelaşi semn de avertisment. Se poate observa care este reactorul 4, fiind marcat cu fum. Pe lângă pădure, mai putem observa linii de înaltă tensiune, care fac scena astfel creată să aducă mai mult a locul real din Ucraina. Pe lângă acestea, mai există un avion Mustang care a fost trimis de la Moscova ca să încerce să stingă incendiul care a avut loc la reactorul 4.

Toate obiectele au fost preluccrate prima data in aplicatia Blender.

2.2. Funcţionalităţi

Aplicaţia oferă posibilitatea de a observa scena atât dintr-un moment de zi, cât şi dintr-un moment de noapte, astfel făcându-se o animaţie de trecere de la zi la noapte. Se poate observa fum care apare şi se expandează într-un ritm încet deasupra reactorului care a explodat. Pentru a face atmosfera mai veridică, se foloseşte un efect de ceaţă uşoară, care poate fi activat sau nu. De asemenea, se pot observa usoare umbre ale obiectelor, care sporesc efectul de fotorealism al scenei. Utilizatorul se poate plimba prin scenă cu ajutorul mouse-ului si a tastelor W,A,S,D. Se poate comuta intre modul wireframe si plane utilizand tastele M si N.

3. Detalii de implementare

3.1. Funcţii şi algoritmi

Pentru crearea acestei scene 3D s-a folosit template-ul oferit la laboratoarele de Prelucrare Grafică. Astfel, pentru a implementa miscarea camerei sus, jos, dreapta, stanga si navigarea acesteia prin scena, am folosit transformarile de translatie, scalare si rotatie a camerei.

Urmatorul pas a fost adaugarea texturilor pe obiecte. Am incercat sa facem texturile cat mai calitative, prin maparea corecta a acestora pe obiecte.

Pentru iluminarea scenei am folosit surse de lumina diferite, cum ar fi (globala, tip spot). Iluminarea a fost implementata prin combinarea celor 3 tipuri de lumina: ambientala, speculara si difuza. Pentru rezultate mai bune, am implementat modelul de iluminare Blinn-Phong, care îmbunătățește performanța, evitând calculul costisitor al vectorului de reflexie. Scopul său este de a îmbunătăți reflexiile speculare în anumite condiții, cum ar fi un coeficient foarte scăzut de strălucire (rezultând astfel o zonă specular mare). Reflexiile speculare ale modelului Phong tind să fie întrerupte imediat ce unghiul dintre direcția de vizionare și reflexia luminii crește peste 90.0 grade (cosinusul devine negativ și astfel este fixat la 0.0). Soluția propusă de Blinn este de a folosi un așa-zis semi-vector în locul vectorului de reflexie pentru a calcula componenta specular. Semi vectorul (H) este un vector unitar la jumătatea distanței dintre direcția de vizionare și direcția luminii. Semi-vectorul este calculat ca:

$$\boldsymbol{H} = \frac{\boldsymbol{L} + \boldsymbol{Eye}}{\|\boldsymbol{L} + \boldsymbol{Eye}\|}$$

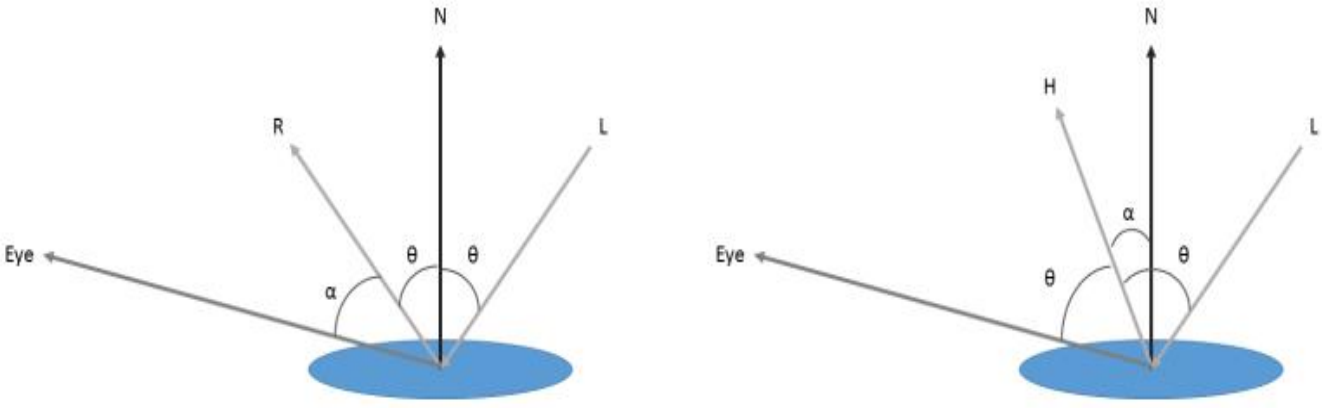


Figura 1 - Modelul specular Phong (stânga) versus modelul specular Blinn (dreapta)

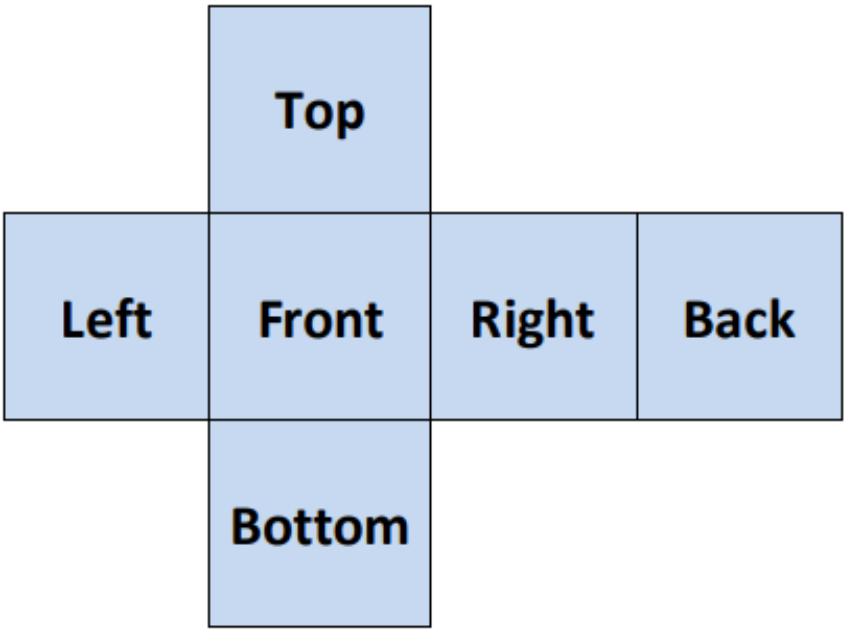
Urmatorul pas a fost sa generam umbrele. Pentru aceasta, am explorat o tehnică ce oferă rezultate decente și este ușor de implementat, și anume shadow mapping (“hărți de umbre”). Rasterizarea cu umbre Shadow mapping este o tehnică multi-trecere care utilizează texturi de adâncime pentru a decide dacă un punct se află în umbră sau nu. Cheia este aceea de a observa scena din punctul de vedere al sursei de lumină în loc de locația finală de vizionare (locația camerei). Orice parte a scenei care nu este direct observabilă din perspectiva luminii va fi în umbră. Etapele principale ale algoritmului sunt descrise mai jos:

1. Rasterizarea scenei din punctul de vedere al luminii. Nu contează cum arată scena (informații despre culoare); singurele informații relevante în acest moment sunt valorile de adâncime. Aceste valori sunt stocate într-o hartă de umbră (sau hartă de adâncime) și pot fi obținute prin crearea unei texturi de adâncime, atașarea la un obiect framebuffer și rasterizarea întregii scene (așa cum este văzută din poziția luminii) în acest obiect. În acest fel, textura de adâncime este umplută direct cu valorile relevante ale adâncimii.

2. Rasterizarea scenei din punct de vedere al observatorului (poziția camerei). Se compară adâncimea fiecărui fragment vizibil (proiectat în cadrul de referință al luminii) cu valorile de adâncime din harta umbrelor. Fragmentele care au o adâncime mai mare decât cea care a fost stocată anterior în harta de adâncime nu sunt direct vizibile din punctul de vedere al luminii și sunt, prin urmare, în umbră.

Prima trecere a algoritmului de shadow mapping generează o hartă de adâncime a întregii scene privită din perspectiva luminii. Această hartă poate fi salvată direct într-o textura prin atașarea texturii la un obiect framebuffer și rasterizarea directă în acesta. OpenGL face toată rasterizarea (informații despre culoare, adâncime și șablon) într-un framebuffer. Culorile stocate în framebuffer sunt utilizate de ecran atunci când se afișează conținutul vizual generat de aplicațiile noastre. Obiectele Framebuffer ne permit să creăm propriile framebuffere și, în loc să rasterizăm în zone speciale de memorie GPU, putem scrie informații (culori, adâncime și informații șablon) în texturi. Aceste texturi sunt atașate ca obiecte buffer de culoare, adâncime și stencil pentru obiectele noastre framebuffer.

Pentru creare unui fundal, am inclus scena noastra 3D în interiorul unui cub cu texturi diferite aplicate pe fiecare față. Acest cub se numeste skybox și este compus din 6 texturi, urmând acest model:



Am crescut realismul scenei 3D prin adăugarea unui efect de ceață, care creeaza o atmosferă specială. Prin manipularea atributelor asociate efectului de ceață putem să personalizăm atmosfera și, de asemenea, să ameliorăm percepția de adâncime (axa Z). Culoarea de fundal trebuie aleasă în funcție de culoarea efectului de ceață:

glClearColor(0.5, 0.5, 0.5, 1.0);

În versiunile de OpenGL cu pipeline fix existau trei metode de calculare a ceții (liniară, exponențială și exponențială pătratică). Aceeași funcționalitate o putem realiza și în versiunea cu pipeline programabil prin adăugarea calculelor de ceață în shader-ul nostru de fragmente. In proiectul dat, o sa folosim modelul de ceata exponentiala, care se apropie mult de

Ceață exponențială:

Un rezultat mai bun poate fi obținut 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 în comparație cu abordarea liniară. $fogFactor = e^{-fragmentDistance * fogDensity}$.

Eliminarea fragmentelor. Valoarea alfa din texturi poate fi utilizată pentru a elimina un fragment. De exemplu, copacii din scena sunt implementati prin combinarea a 2 plane, pe care este mapata o textura de compac. Am verificat valoarea canalului alfa urmând să eliminam fragmentul dacă valoarea este mai mică de 0,1.

3.2. Modelul grafic

Pipeline-ul de rasterizare implementat în OpenGL este ilustrat în figura de mai jos. Pipeline-ul programabil conține cel puțin stagiile de vertex shading și fragment shading, și înlocuiește pipeline-ul fix, care nu mai este suportat (iluminare și transformări). Faza de vertex shading procesează fiecare vârf independent de celelalte; datele vârfurilor sunt primite de la aplicație prin obiecte de tip vertex buffer (VBO). Faza de rasterizare generează un set de fragmente care sunt procesate în faza de fragment shading; ieșirea este reprezentată de culoarea și adâncimea fragmentului (coordonata Z).

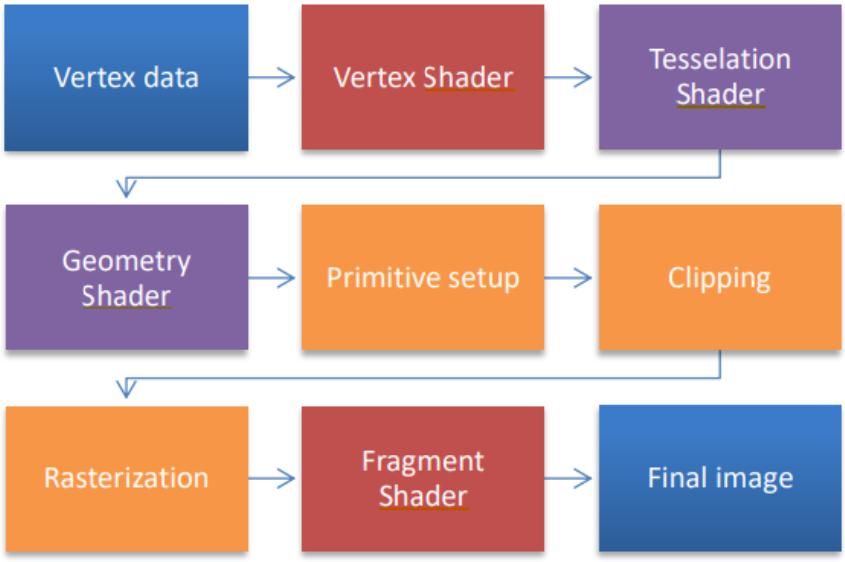


Figure 1 – Pipeline-ul programabil OpenGL

3.3. Structuri de date

Un obiect 3D este reprezentat folosind informații topologice ale vârfurilor. Pentru fiecare vârf avem nevoie de atribute cum ar fi poziția, culoarea, coordonatele de textură și alte informații relevante. Datele de vârf sunt stocate în obiecte Vertex Buffer (VBO). O serie de VBO pot fi stocate într-un obiect Array Vertex data Vertex Shader Tesselation Shader Geometry Shader Primitive setup Clipping Rasterization Fragment Shader Final image Vertex. (VAO). Un obiect Vertex Buffer (VBO) este un buffer folosit pentru a păstra o serie de atribute de vârf (cum ar fi poziția, culoarea, vectorul normală etc.). Un obiect Vertex Array (VAO) grupează mai multe VBO-uri. Pentru a utiliza VBO trebuie să:

- generăm nume pentru VBO (ID-uri): glGenBuffers(…)
- selectăm un anumit VBO pentru inițializare: glBindBuffer(GL_ARRAY_BUFFER, …)
- încărcăm date în VBO: glBufferData(GL_ARRAY_BUFFER, ...)

O procedură similară este necesară pentru obiecte Vertex Array:

- generăm nume pentru VAO (ID-uri): glGenVertexArrays(…)
- selectăm un anumit VAO pentru inițializare: glBindVertexArray(…)
- reîmprospătăm VBO-urile associate cu VAO-ul curent: glBindBuffer(…) și glVertexAttribPointer(…)
- selectăm VAO-ul folosit pentru rasterizare: glDrawArrays(…)

Pentru a putea rasteriza obiecte 3D, trebuie să scriem cel puțin un vertex shader și fragment shader. Pentru aceasta avem nevoie să:

- generăm un obiect shader (identificabil printr-un ID unic): glCreateShader(…)
- atașăm codul sursă al shader-ului la obiectul de tip shader: glShaderSource(…)
- compilăm shader-ul: glCompileShader(…) După ce am compilat programele vertex shader și fragment shader, trebuie să le combinăm într-un așanumit obiect de tip program shader.

Pașii pentru aceasta sunt după cum urmează:

- creăm un program shader: glCreateProgram(…)
- atasăm shader-ele compilate anterior: glAttachShader(…)
- operația de linking a shader-elor: glLinkProgram(…)

3.4. Ierarhia de clase

Proiectul contine urmatoarele clase:
Main.cpp
Camera.cpp
Mesh.cpp
Model.cpp
Shader.cpp
SkyBox.cpp

Shaderele au fost structurate in urmatoarele clase:
ShaderStart.vert
ShaderStart.frag
SkyBoxShader.vert
SkyBoxShader.frag

DepthMap.vert
DepthMap.frag
ScreenQuad.vert
ScreenQuad.frag

4. Prezentarea interfetei grafice utilizator

Pentru a naviga prin scena, utilizatorul va folosi mouse-ul pentru miscari de rotatie a camerei si tastele W,A,S,D pentru a se misca prin scena. Tastele 1 si 2 vor fi folosite pentru activarea/dezactivarea modului automat de prezentare. Tastele 5 si 6 activeaza/dezactiveaza ceata. Tastele M/N comuteaza intre modul wireframe si fill.

Interfata grafica este una prietenoasa si proiectata astfel incat utilizatorul sa poata naviga instinctiv. Pe masura trecerii timpului, luminozitatea scenei va scadea, ceea ce simuleaza un efect de venire a noptii.

De asemenea, cu cat obiectele sunt mai indepartate, cu atat mai observabil va fi efectul de ceata. Cand utilizatorul ce apropie de obiect, ceata se va rari, marind claritatea obiectului.

5. Concluzii si dezvoltari ulterioare

Acest proiect m-a ajutat sa ma familiarizez cu aplicatia OpenGL si Blender. Prin intermediul acestora, am reusit sa texturez obiecte 3D, sa le aranjez in scena prin translatii, scalari, rotatii, sa le iluminez si sa le creez umbre. De asemenea, am invatat sa animez elemente din scena si sa adaug lumini tip spot. Am deprins concepte de baza ale prelucrarii grafice, cum ar fi:

Pipeline programabil
Shadow Map
Lumina difuza, speculara, ambientala
Texturarea obiectelor
Shader
Efecte speciale
Animare scena
VBO/VAO
Shadow Acnee
Scena wireframe
Luminozitate

Conceptul de Shadere m-a familiarizat cu procesul de trimitere si prelucrare a parametrilor intre CPU si GPU.

Ulterior, proiectul poate fi dezvoltat prin adaugarea mai multor obiecte in scena, prin adaugare de mai multe animatii (ploaie, vant, ninsoare) si algoritmi (de coliziune).

6. Referinte

1. https://free3d.com/3d-models/
2. https://www.blender.org/support/tutorials/
3. https://learnopengl.com/Introduction
4. https://community.khronos.org/
5. https://www.blender.org/features/modeling/
6. Indrumator de laborator PG