FACULTATEA DE AUTOMATICA SI CALCULATOARE

PROIECT PRELUCRARE GRAFICA

Nume: Branet Tudor-Andrei

Grupa: 30234

CUPRINS

Contents

1) Prezentarea temei

Proiectul are ca si scop realizarea unei prezentari fotorealiste a unei scene de obiecte 3D cu ajutorul specificatiei OpenGL, dar si a librariilor precum GLM si GLFW. Pentru plasarea obiectelor in scena si maparea texturilor am folosit programul Blender.

In realizarea proiectului, am implementat diferite surse de lumina, posibilitatea de a vizualiza scena in modurile wireframe, point si solid, maparea texturilor, animatii, efect de ceata. Utilizatorul poate vizualiza scena prin modificarea pozitiei si orientarii camerei cu ajutorul tastaturii.

2) Scenariul

2.2) Descrierea scenei si a obiectelor

Scena pe care am realizat-o surprinde un peisaj de la ferma, situata intr-o zona montana.

Aceasta este alcatuita din urmatoarele obiecte principale:

Moara

Hambar

Utilaj agricol

Casa

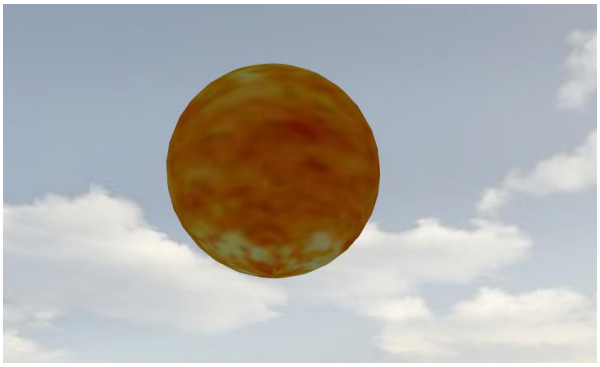
Masina

In scena se afla de asemenea si cateva animale:

2.2) Functionalitati

Ca si functionalitati, am realizat cateva animatii pentru diferite obiecte din scena, precum:

- Miscarea elicelor morii prezentate anterior;
- Deschiderea / inchiderea usii de la casa;
- Rotatia soarelui (obiect ce inglobeaza lumina directionala);



- Rotatia unui nor, precum si animatie de ploaie

3) Detalii de implementare

3.1) Functii si algoritmi

a) Pentru realizarea animatiilor obiectelor:

Pentru a anima anumite parti ale unor obiecte descarcate (cum ar fi elicele de la moara sau usa de la casa) a fost nevoie prima data sa decupez din Blender partea respectiva si sa o exportez ca un obiect separat, realizand cate o functie care sa se ocupa de randarea acestuia in scena. Functiile primesc ca si argumente shaderul cu care desenez obiectul si o variabila booleana care indica daca rasterizarea ale loc din perspectiva luminii sau a camerei.

Apoi, pentru a face ca un obiect sa se roteasca in jurul unei anumite axe raportat la originea sa, a fost nevoie sa realizez o translatie in origine, sa realizez rotatia, iar apoi sa fac o translatie inversa.

Pentru a controla anumite animatii cu ajutorul tastelor, am folosit niste flag-uri pe care le setez cu ajutorul functiei de keyboardCallBack, iar apoi tin cont de acestea in functiile de randare corespunzatoare.

b) Pentru realizarea animatiei de prezentare:

Initial, am creat o functie numita ,,presentation" care se apeleaza in bucla de rulare a aplicatiei in care este realizata prezentarea propriu zisa. Pentru activarea modului de prezentare am folosit tasta P care seteaza pe true variabila booleana ,,presentation_on" si seteaza pozitia si orientarea camerei de la inceputul prezentarii.

In functia ,,presentation" , verific daca flagul pentru prezentare este activ, caz in care reduc viteza si execut diferit miscari ale acesteia (MOVE_FORWARD, MOVE_UP etc) cat timp anumite conditii sunt indeplinite pe rand (de ex., camera nu a ajuns inca la anumite coordonate). La sfarsitul prezentatii, setez variabile ,,presentation_on" pe false.

c) Pentru implementarea luminii:

Pentru implementarea luminii, am folosit modelul de iluminare Phong. Acesta presupune faptul ca lumina este alcatuita din 3 componente: **lumina ambientala** (valoarea poate fi precalculata ca un efect global, iar calculul sau nu depinde de pozitia luminii sau a spectatorului), **lumina difuza** (calculul sau depinde de normala

suprafetei si de directia sursei de lumina, dar nu de pozitia spectatorului) si **lumina speculara** (aceasta componenta da efectul de stralucire, iar calculul sau depinde inclusiv directia de vizualizare).

In proiect am implementat 2 tipuri de lumina, si anume lumina **directionala** (care este situata la infinit, iar razele sale sunt paralele si au aceeasi directie) si lumina **punctiforma** (care are o anumita pozitie in spatiu, ilumineaza radial si este estompata in functie de distanta).

Lumina directionala este cea care genereaza umbrele si se misca odata cu cu un obiect ce reprezinta soarele.

Lumina punctiforma am plasat-o in dreptul unui felinar:

La apasarea tastei K se activeaza o ,,lanterna” ce este implementata tot ca o lumina punctiforma, avand pozitia camerei:

d) Pentru implementarea cetei:

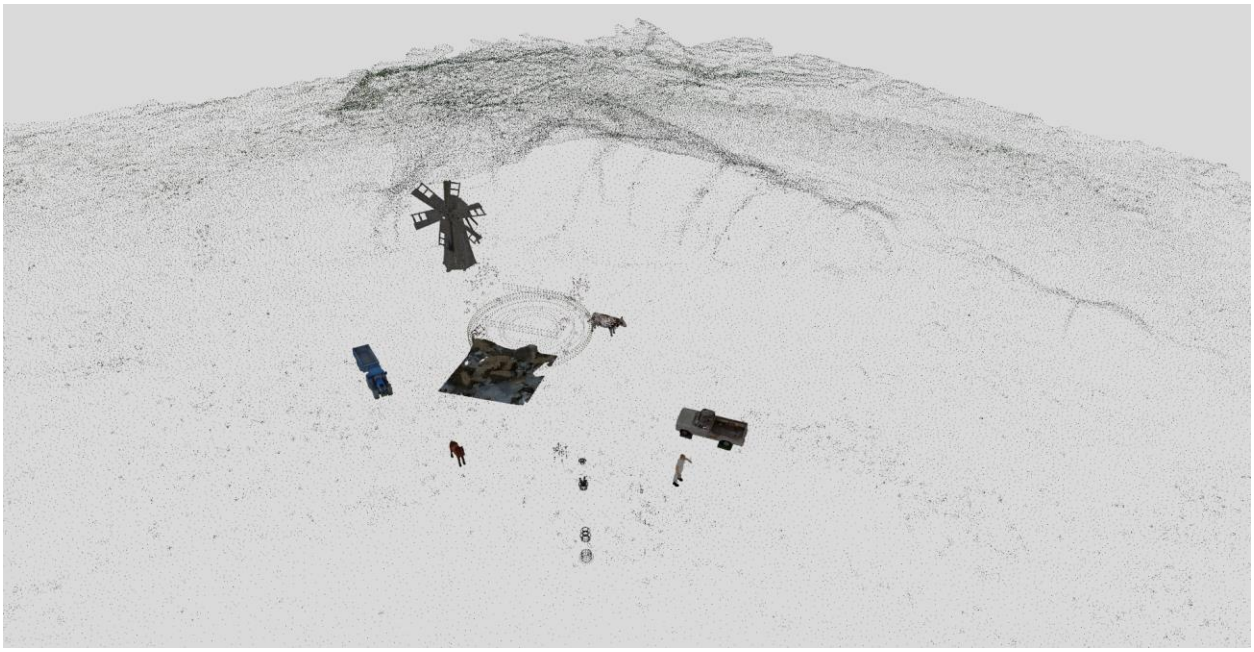
Efectul de ceata este implementat in Fragment shader unde am implementat functia computeFog care are rolul de a calcula factorul de ceata (valoare intre 0 si 1). Acesta depinde de o constanta ce reprezinta densitatea, precum si de distanta fragmentelor (factorul de ceata scade exponential odata cu distanta). Culoarea finala a fragmentului reprezinta o valoare interpolata intre culoarea de baza si culoarea cetii, tinand cont de fogFactor.

e) Pentru implementarea diferitelor moduri de vizualizare:

- vizualizare in modul **solid**: in momentul in care se apasa tasta 2, se apeleaza functia *glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);*

- vizualizare in modul **wireframe**: in momentul in care se apasa tasta 3, se apeleaza functia *glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);*

- vizualizare cu **puncte**: in momentul in care se apasa tasta 4, se apeleaza functia *glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);*



f) Pentru implementarea animatiei / efectului de ploaie:

Am folosit un singur obiect ce reprezinta o picatura si o variabila „translateFactor” ce descreste la fiecare randare si influenteaza coordonata Y la care se deseneaza picaturile. Cand picaturile ajung la sol, variabila se reseteaza la 0. Pentru a simula efectul de ploaie, am folosit functia rand pentru a genera in mod aleatoriu coordonatele X, Y (tinand cont de translateFactor) si Z la care sa desenez o picatura.

3.2) Modelul grafic

Rolul de baza al specificatiei OpenGL este de a primi ca si input primitive grafice, cum ar fi puncte, linii și poligoane, și de a le converti în pixeli prin intermediul unui pipeline grafic programabil. Acesta este alcatuit din diferite stagii care pot fi sau nu implementate de catre programator cu ajutorul unor programe numite **shadere** ce ajung sa fie executate de catre placa grafica. Shaderele sunt scrise in limbajul GLSL. Stagiile pipeline-ului sunt urmatoarele:

- Vertex Shader (obligatoriu);
- Tesselation Shader (optional);
- Geometry Shader (optional);
- Primitive Setup (nu este programabil);
- Clipping (nu este programabil);
- Rasterization (nu este programabil);
- Fragment Shader (obligatoriu);

Vertex Shader proceseaza fiecare varf independent de celelalte, primindu-le prin niste obiecte de tip vertex buffer (VBO). Etapa de rasterizare genereaza un set de fragmente care trebuie sa fie afisate pe ecran si sunt in continuare prelucrate de catre Fragment Shader. Astfel, se obtine culoarea finala si adancimea fragmentului.

3.3) Structuri de date

Structurile de date pe care le-am folosit sunt urmatoarele:

- **Matrici** (obiecte de tipul glm::mat4): folosite pentru stocarea matricilor de modelare, vizualizare si proiectie;
- **Vectori** (obiecte de tipul glm::vec3): folositi pentru stocarea diferitelor tipuri de coordonate sau culori (de ex. pozitia / orientarea camerei, directia luminii etc);
- **Obiect de tipul Camera:** folosit pentru retinerea informatiilor despre camera utilizate in crearea matricii de vizualizare (cameraPosition, cameraTarget, cameraUp);
- **Obiecte de tip gps:**:Model3D – in acest tip de obiecte sunt stocate modele 3D care urmeaza sa fie randate;
- **Obiecte de tip gps:**:SkyBox – obiecte in care se pastreaza cele doua skybox-uri implementate: daySkyBox si nightSkyBox;
- **Obiecte de tip gps::Shader –** cu ajutorul acestora sunt gestionate shader-ele utilizate in proiect;
- **Obiecte de tip std:**:vector – in care se incarca imaginile din care se construiesc skybox-urile

De asemenea, in Basic Fragment Shader am folosit structura urmatoare pentru a pastra caracteristicile luminii punctiforme.

```
struct PointLight {
    vec3 position;

    float constant;
    float linear;
    float quadratic;

};
```

4) Manual de utilizare

Utilizatorul poate interactiona cu scena utilizand urmatoarele taste:

Pentru miscarea camerei:

- **W**: pentru deplasarea inainte a camerei;
- **S**: pentru deplasarea inapoi a camerei;
- **A**: pentru deplasarea la stanga;
- **D**: pentru deplasarea la dreapta;
- **UP**: pentru rotatia in sus;
- **DOWN**: pentru rotatia in jos;
- **LEFT**: pentru rotatia la stanga;
- **RIGHT**: pentru rotatia la dreapta;
- **Q**: rotatia modelului la stanga;
- **E**: rotatia modelului la dreapta;
- **Z**: deplasarea camerei in sus;
- **X**: deplasarea camerei in jos;
- **1**: afisarea coordonatelor camerei

Pentru controlul cetii:

- **C**: activarea cetii;
- **V**: dezactivarea cetii;
- **B**: cresterea densitatii cetii;
- **N**: scaderea densitatii cetii

Pentru controlul luminii:

- **J**: deplasarea la stanga a luminii directionale;
- **L**: deplasarea la dreapta a luminii directionale;
- **R**: activarea modului 1 de iluminare (lumina directionala);
- **T**: activarea modului 2 de iluminare (lumina punctiforma plasata in felinar);
- **K**: activarea modului 3 de iluminare (lumina punctiforma in fata camerei)

Pentru schimbarea modurilor de vizualizare:

- **2**: activarea modului solid;
- **3**: activarea modului wireframe;
- **4**: activarea modului point

Pentru controlul elicelor morii:

- **F**: marirea vitezei de rotatie;
- **G**: scaderea vitezei de rotatie

Pentru controlul usii de la casa:

- **Y**: deschidere usa;
- **U**: inchidere usa

Pentru controlul ploii:

- **5**: activare ploaie:
- **6**: dezactivare ploaie

5) Concluzii si dezvoltari ulterioare

In concluzie, aplicatia pe care am realizat-o ar putea fi dezvoltata ulterior prin adaugarea a mai multor obiecte si animatii, imbunatatirea efectului de umbra, cresterea fotorealismului.

6) Referinte

- https://ro.wikipedia.org/wiki/OpenGL
- Lucrari de laborator