PROIECT PRELUCRARE GRAFICĂ

Student: Vlad Darius-Stefan

Grupa: 30236

Profesor îndrumător: Adrian Sabou

An 2024-2025

I. CUPRINS

II. PREZENTAREA TEMEI

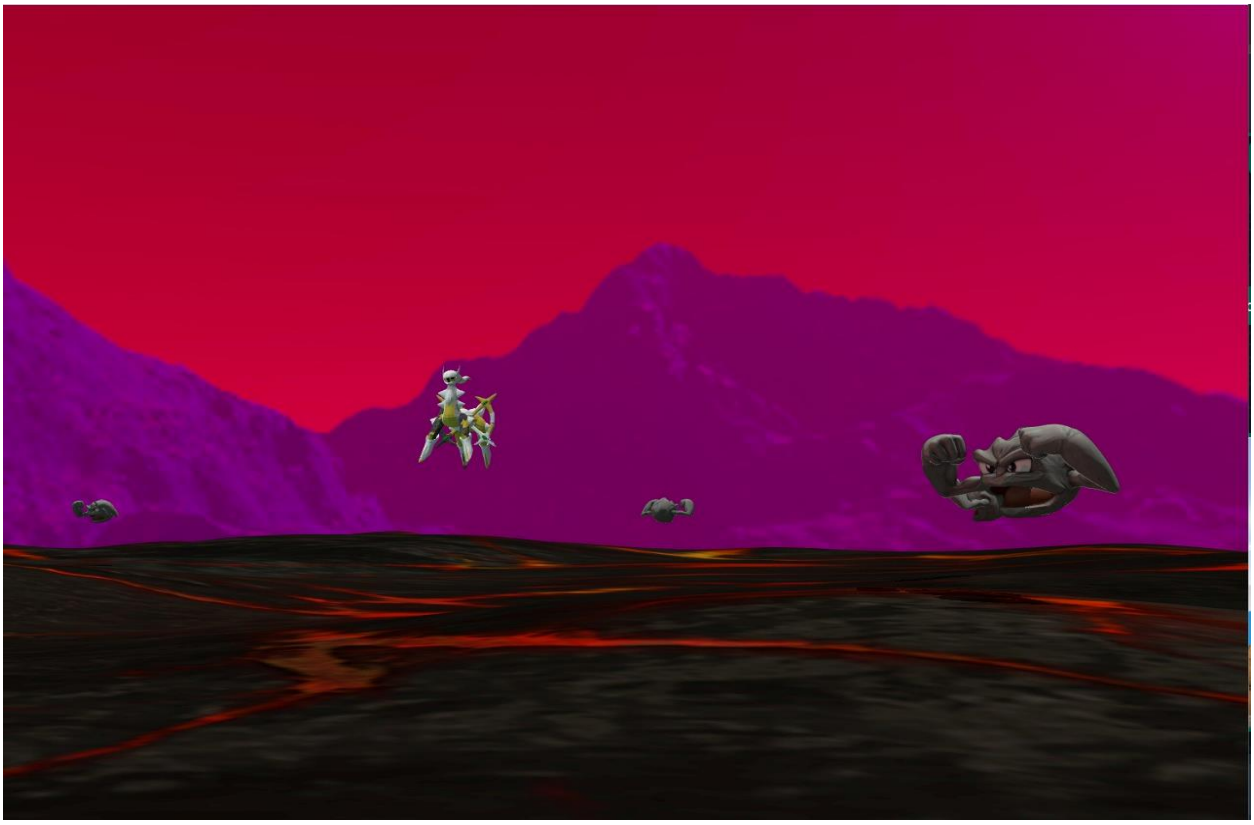
Proiectul are ca și scop realizarea unei prezentări fotorealiste a unor scene de obiecte 3D utilizând librăriile prezentate la laborator (OpenGl, GLFW, GLM, etc.). Utilizatorul are posibilitatea de a interacționa scena prin intermediul mouse-ul și tastaturii.

Tema aleasa pentru acest proiect este lumea Pokemon.

III. SCENARIUL

III.1. Descrierea scenei și a obiectelor

Proiectul contine 4 scene fiecare reprezentand un habitat diferit. In fiecare zona exista tipuri diferite de pokemoni iar scopul jocului este „sa ii prinzi pe toti”. Jucatorul incepe intr-o zona pasnica , unde va gasi si pokeball-ul , cu ajutorul caruia va putea prinde pokemoni. Deoarece nu ar fi corect sa prindem toti pokemonii , vor putea fi prinsi doar cativa dintre cei din scena , cei corupti. Dupa ce termina primul nivel utilizatorul poate trece prin portal ajungand in urmatoarea zona: cimitirul. Aceasta zona este menita pentru a aduce un aer infricosator jocului , selectia de pokemoni din aceasta zona ajutand de asemenea la acest afect. Dupa trecerea prin portal din aceasta zona , jucatorul va ajunge intr-o zona de plaja langa care se afla si apa. Ultima zona reprezinta o lumea diferita de cea normala , unde va da de cel mai puternic pokemon: Arceus.



III.2. Funționalități

Scena are mai multe funcționalități precum:

- Tastele **W**, **A**, **S**, **D –** permite mișcarea camerei
- Tastele **1,2,3,4 –** pentru teleportarea directa in celelalte zone
- Tasta **C** – pentru a incepe derularea unui cutscene
- Tasta **F** – pentru a activa modul de “fly” (miscarea pe toate axele)
- Tasta **L** – pentru a ridica pokeball-ul de pe jos

IV. DETALII DE IMPLEMENTARE

IV.1. Funcții și algoritmi

În implementarea proiectului am folosit o serie de algoritmi pentru implementarea anumitor funcționalități precum:

- **Mișcarea camerei** – funcția de mișcare în funcție de tasta apăsată și cea de rotație a camerei in functie de miscarea mouse-ului
- **Efectul de ceață** – dezvoltat pe baza laboratorului

$$fogFactor = e^{-(fragmentDistance * fogDensity)^2}$$

- **Efectul de transparenta** – am adaugat in fragment shader o variabila uniform ce reprezinta opacitatea , de asemenea trebuie sa avem grija la ordinea de desenare a obiectelor
- **Efectul de valuri** – folosind "wave displacement"
- **Efectul de highlight** – m-am folosit de stencil buffer
- **Miscarea obiectelor pe o traiectorie predefinita** – am creat o traiectorie sub forma unui triunghi si am translatat obiectul in functie de timp pe aceasta axa

Algoritmi:

- **Luminile punctiforme** sunt surse de lumină cu o poziție specifică, care ilumineză în mod uniform în toate direcțiile. Spre deosebire de luminile direcționale, razele emise de luminile punctiforme se estompează pe măsură ce distanța față de sursa de lumină crește, astfel încât obiectele mai apropiate de sursă sunt mai bine iluminate decât cele aflate la distanță. Atunci când utilizăm luminile punctiforme, direcția luminii variază între fragmente și trebuie calculată separat pentru fiecare fragment. Realismul este semnificativ îmbunătățit atunci când se aplică atenuarea patratică în funcție de distanță. Pentru a implementa această atenuare, trebuie să definim coeficienții de atenuare în shader sau să îi transmitem din aplicație prin variabile uniforme.

- **Luminile spotlight** - sursă de lumină situată într-un anumit loc din mediu care, spre deosebire de lumina omnidirecțională, nu emite raze de lumină în toate direcțiile, ci doar într-o direcție specifică. Rezultatul este că doar obiectele aflate într-un anumit raion, în direcția luminii spotlight, sunt iluminate, iar restul rămân în umbră. Un exemplu bun de spotlight ar fi un felinar de pe stradă sau o lanternă. O lumină spotlight în OpenGL este reprezentată de o poziție în spațiul lumii, o direcție și un unghi de tăiere (cutoff angle) care specifică raza luminii spotlight. Pentru fiecare fragment, calculăm dacă fragmentul se află între direcțiile de tăiere ale luminii spotlight (adică în conul său), iar dacă da, iluminăm fragmentul corespunzător.

- **Umbrele** - Shadow mapping este o tehnică care folosește texturi de adâncime pentru a determina dacă un punct se află în umbră sau nu. Este important să observăm scena din punctul de vedere al sursei de lumină, nu din locația camerei. Orice parte a scenei care nu este direct vizibilă din perspectiva luminii va fi considerată în umbră. Tehnica se desfășoară în două etape principale: prima etapă constă în rasterizarea scenei din punctul de vedere al luminii, iar valorile de adâncime sunt stocate într-o hartă de

adâncime. A doua etapă presupune rasterizarea scenei din perspectiva observatorului, unde adâncimea fiecărui fragment vizibil este comparată cu valorile de adâncime stocate în harta umbrelor.

- **Valuri** – [1] in cazul unei unde progresive, relația de deplasare descrie modul în care poziția unei particule se schimbă în timp pe măsură ce unda o traversează. Aceasta arată relația dintre deplasarea particulei față de poziția sa de echilibru și atât poziția de-a lungul undei (reprezentată de x), cât și timpul (reprezentat de t).

IV. 2. Modelul grafic

Scenele au fost realizate in blender iar apoi au fost incarcate in Visual Studio. Fiecare obiect dinamic a fost incarcat direct in Visual Studio pentru a avea control asupra pozitiei sale individuale.

IV. 3. Structuri de date

In realizarea proiectului am folosit diferite structuri de date din librariile **GLM si OpenGl** , precum:

- **glm::mat3 si glm::mat4** - pentru declararea matricilor de model si matricilor pentru normale

```
glm::mat4 model;
glm::mat4 view;
glm::mat4 projection;
glm::mat3 normalMatrix;

glm::mat4 orcModel;
glm::mat3 orcNormalMatrix;



glm::mat4 weaponModel;
glm::mat3 weaponNormalMatrix;

glm::mat4 arceusModel;
glm::mat3 arceusNormalMatrix;

glm::mat4 pokemonGroundModel;
glm::mat3 pokemonGroundNormalMatrix;

glm::mat4 snorlaxModel;
glm::mat3 snorlaxNormalMatrix;
```

- **GLint** – structura de date pentru stocarea locatiei din shadere a variabilelor uniforme

```
GLint modelLoc;
GLint viewLoc;
GLint projectionLoc;
GLint normalMatrixLoc;
GLint lightDirLoc;
GLint lightColorLoc;

GLint particleModelLoc;
GLint particleViewLoc;
GLint particleProjectionLoc;
GLint particleTintLoc;

GLint modelOutliningLoc;
GLint normalMatrixOutliningLoc;
GLint viewOutliningLoc;
GLint projectionOutliningLoc;
GLuint transparencyLoc;
```

- **gps::Model3D** – structura de date pentru modelele 3D utilizate

```
gps::Model3D arceus;
gps::Model3D snorlax;
gps::Model3D magnemite;
gps::Model3D evee;
gps::Model3D dragonite;
gps::Model3D gengar;
gps::Model3D lucario;
gps::Model3D gyarados;
gps::Model3D lapras;
gps::Model3D geodude;

gps::Model3D water;
```

- **gps::Shader** – structura de date pentru shadere

```
// shaders
gps::Shader myBasicShader;
gps::Shader outliningShader;
gps::Shader simpleDepthShader;
gps::Shader particleShader;
gps::Shader waterShader;
```

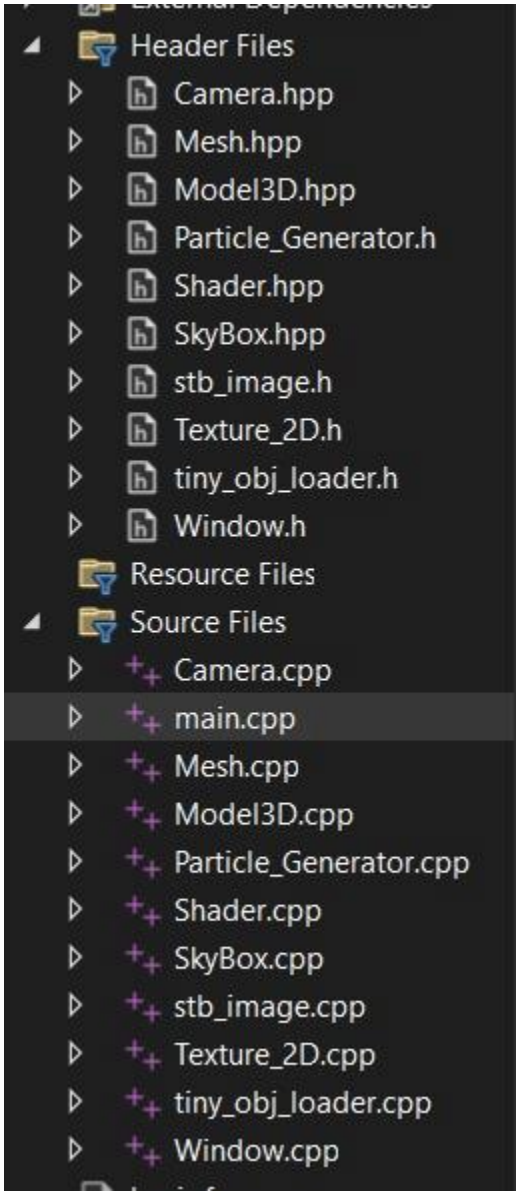
- **gps::Camera** – structura de date creata de catre noi pentru a simula o camera

```
// camera
gps::Camera myCamera(
    glm::vec3(0.0f, 2.0f, 35.0f),
    glm::vec3(0.0f, 5.0f, -10.0f),
    glm::vec3(0.0f, 1.0f, 0.0f));
```

- **gps::SkyBox** - pentru implementarea skybox-ului

```
//skybox
gps::SkyBox mySkyBox;
gps::SkyBox nightSkyBox;
gps::SkyBox redSkyBox;
gps::SkyBox hillSkyBox;
gps::Shader skyboxShader;
```

IV. 4. Ierarhia de clase



- **Clasa Camera** este responsabilă pentru gestionarea poziției, direcției și mișcării camerei într-o scenă 3D. De asemenea, include funcționalități pentru calcularea matricelor de vizualizare și proiecție, utilizate în randarea grafică.

- **Clasa Mesh** reprezintă structura geometrică a unui obiect 3D, incluzând informații despre vertecși, normale, coordonate de textură și indecși. Este esențială pentru definirea formelor obiectelor în grafică 3D.

- **Clasa Model3D** este folosită pentru a încărca și gestiona modele 3D complexe. Aceasta poate include mai multe obiecte de tip Mesh și metode pentru încărcarea fișierelor de modele externe, cum ar fi .obj sau .fbx.

- **Clasa Texture_2D** este utilizată pentru a încărca și gestiona texturi 2D aplicate pe obiecte 3D. Include metode pentru configurarea parametrilor texturilor și încărcarea acestora în GPU.

- **Clasa SkyBox** se ocupă de crearea și randarea unui skybox, o tehnică folosită pentru a simula un orizont sau un fundal infinit într-o scenă 3D.

- **Clasa Shader** gestionează programele de shader utilizate în OpenGL. Aceasta include metode pentru încărcarea, compilarea și utilizarea shaderelor de tip vertex și fragment.

V. PREZENTAREA INTERFEȚEI GRAFICE UTILIZATOR / MANUAL DE UTILIZARE

Pentru a putea vizualiza scena utilizatorul va rula PG-Project.exe. Prima oara se va deschide primul nivel. Utilizatorul poate utiliza “cheat code-uri” pentru a ajunge in alte zone de pe taste , sau poate parcurge fiecare nivel , trecand de la unul la altul prin portal.

Pentru deplasare se vor folosi tastele**: W , A , S , D.**

Pentru miscarea camerei se va folosi mouse-ul.

Pentru atacul cu pokeball-ul se va folosi **LMB** (left mouse button).

VI. CONCLUZII ȘI DEZVOLTĂRI ULTERIOARE

Scena include deja câteva optimizări esențiale, însă există posibilități de îmbunătățire, cum ar fi implementarea detecției coliziunilor, simularea efectului vântului sau adăugarea animațiilor pentru elementele scenei. De asemenea, scena ar putea fi extinsă pentru a include mai multe obiecte, sporind astfel gradul de complexitate și realism.

Pe viitor, intenționez să dezvolt o interfață grafică prin care utilizatorul să poată vizualiza Pokémonii capturați și să inițieze lupte între aceștia. În plus, vreau să creez un simulator de particule care să permită generarea unor efecte vizuale noi și captivante, precum explozii, fum sau efecte de magie, oferind astfel o experiență interactivă și dinamică utilizatorului.

VII. REFERINȚE

- Laboratoare si Cursuri
- https://learnopengl.com/
- [1] https://www.geeksforgeeks.org/displacement-relation-in-a-progressive-wave/