OpenGL Project

Prelucrare Grafica

Student: Voicu Laura Luisa

Grupa: 30236

Cuprins

1. Prezentarea temei

Scopul proiectului este acela de a simula navigarea print-o scena complexa alcatuita din elemente din zona urbana, dar si din cea rurala.

Utilizatorul va identifica o diversitate de cladiri, cu forme, texturi si iluminari diferite, elemente naturale ca ploaie, ceata, lac transparent, texturi de plante, munti detaliati. In scena sunt prezente si animatii precum miscarea unui OZN, a unui elicopter si miscarea efectiva a camerei.

De asemenea, utilizatorul poate sa interactioneze cu aplicatia prin intermediul mouse-ului si a tastaturii pentru miscarea in camera, deplasarea sau schimbarea sursei de lumina, cresterea intensitatii cetii, activarea ploii, etc.

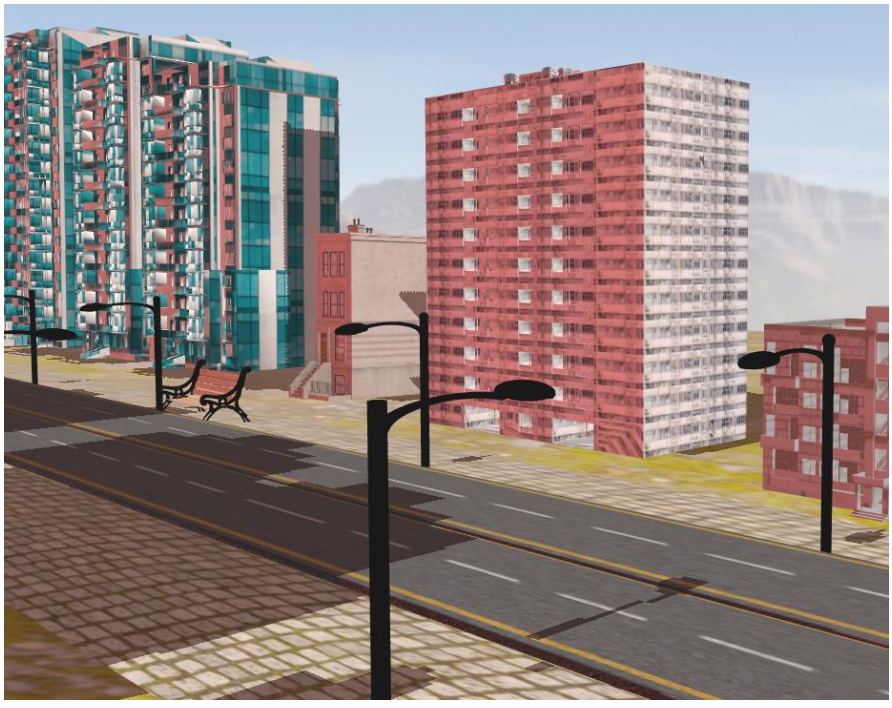
2. Scenariu

2.1 Descrierea scenei si a obiectelor

Scena aplicatiei se imparte in doua subscene:

- Scena urbana, alcatuita din cladiri de diverse forme si texturi, felinare, banci, personaje, stada si trotuar ce se finalizeaza cu intrarea intr-un tunel din interiorul unui munte.
    - Elemente create cu shader-ul **basic.vert , basic.frag**:
        - Cladiri de diverse forme si texturi
        - Strada si trotuar ce se finalizeaza cu intrarea intr-un tunel
        - Munti cu texturi si shape-uri complexe
        - Stalpi de iluminat si bancute

      Dupa cum se observa in imaginile de mai jos, prin acest shader se obtin umbre ale obiectelor in functie de pozitia lor fata de lumina.



- Elemente create cu shader-ul **animationShader.vert, animationShader.frag**:

- Elicopter ale carui elice se misca

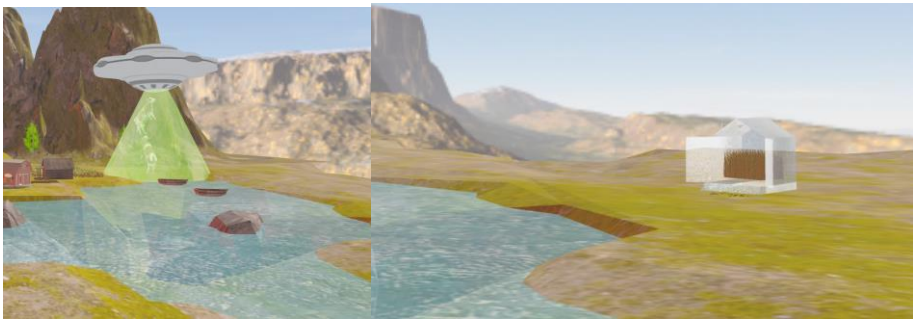
- Elemente create cu shader-ul **twoLightShader.vert, twoLightShader.frag:**
  - Extraterestru, vacuta, gaina

  Acest shader foloseste doua lumini punctiforme care cad pe obiecte si care pot fi activate si dezactivate din butoanele 1 si 2.

- Scena rurala
  - Elemente create cu shader-ul **basic.vert, basic.frag**:
    - Siloz, cabana, hambar, baloti de fan
      - La fel ca si in subscena urbana, cu acest shader se creeaza si umbre pentru obiectele create

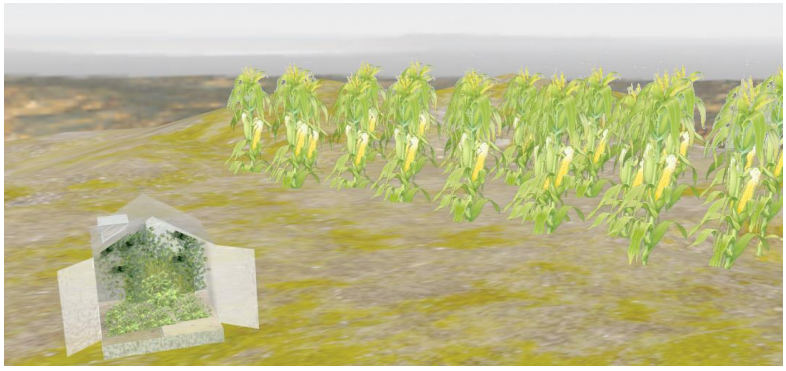
- Elemente create cu shader-ul **transparent.vert, transparent.frag**:
  - Lacul, raza de la OZN, Sera de Flori

  Prin aceste shadere se va obtine o textura transparenta (va scadea opacitatea obiectului)



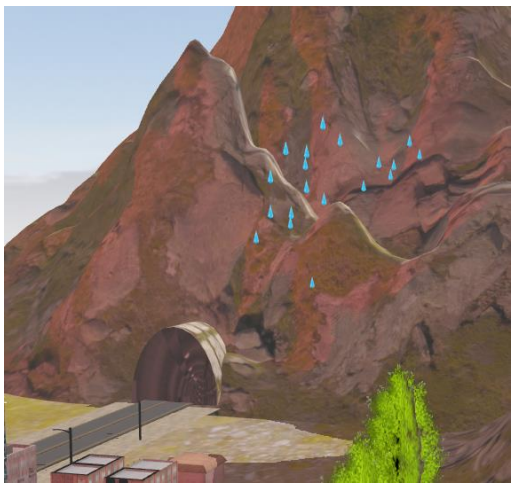
- Elemente create cu shader-ul **animationShader.vert, animationShader.frag**: vacutele si OZN-ul

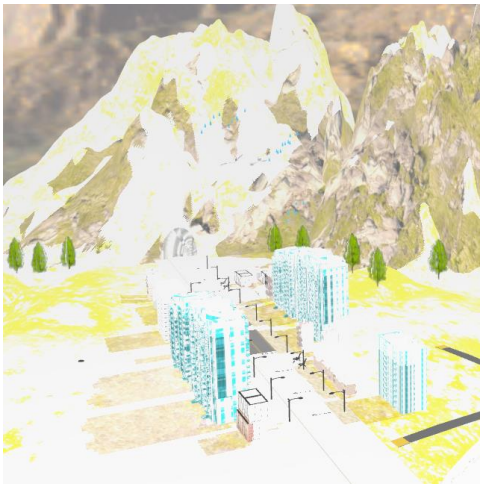
- Elemente create cu shader-ul **transparent2.vert, transparent2.frag**: shadere folosite la eliminarea fragmentelor
  - Copaci, plante (cele din sera), lan de porumb



- Alte elemente speciale
    - Lumina punctiforma si cea directionala: pentru lumina directionala se poate oberva diferenta in functie de unghiul de rotatie in primele 2 imagini, iar in cea de-a treia este aplicata lumina punctiforma

- Ploaie, ceata, fulger





2.2 Functionalitati

- Vizualizarea scenei prin mouse si tastatura si a animatiei de prezentare
- Prezenta a doua surse de lumina distincte (directionala si punctiforma)
  - Se observa prezenta luminii directionale apasand tastele J si L
  - Lumina punctiforma este activata prin tasta O
- Prezenta umbrelor in aplicatie
- Animatii : rotatie in jurul unui punct diferit de origine (centrul bazei elicopterlui pentru elice si centrul razei de la OZN pentru vacute si OZN)
- Elemente de ceata , tunet , ploaie
  - Pentru schimbarea intensitatii cetei: F, G
  - Pentru activarea/dezactivarea ploii: R
  - Elementul de tunet se trigger-uieste automat dupa un interval de timp calculat cu ajutorul functiei `glfwGetTime()`, DOAR daca ploaia este activata, sau manual , folosind tasta H
- Obiecte luminate cu 2 lumini punctiforme
  - Activare/Dezactivare lumina: 1, 2
- Obiecte transparente si fragmentate
- Skybox

3. Detalii de implementare

3.1 Functii si algoritmi

3.1.1 Functii si algoritmi

- `keyboardCallback(GLFWwindow* window, int key, int scancode, int action, int mode)`
  - folosita pentru a receptiona butoanele la o singura apasare a lor
- `mouseCallback(GLFWwindow* window, double xpos, double ypos)`
  - folosita la miscarea in scena folosind mouse-ul
- `movementForAllShaders(glm::mat4 view)`
  - folosita la mutarea in scena a obiectelor pentru fiecare shader
- `processMovement()`
  - folosita la receptionarea tastelor si (in cazul miscarii in scena), trimise mai departe la Camera

- `initModels()`
- `initShaders()`
- `initSkyBox()`
- `initFBO()`
  - `folosita la initializarea hartii de adancime a scenei`
- `renderAnimation()`
  - `initRain()`
  - `rotationAroundCenter`
  - `rotationAroundCenter(glm::mat4 anim_model, glm::vec3 rotationCenter, float rotAngle, glm::vec3 rotationAxis)`

Functii folosite la adaugarea obiectelor pe care se aplica animatie in program.

- `getRandomRainDropPositionAndSpeed(int index)`
  - `folosite la obtinerea pozitiilor random in scena si viteze random pentru generarea ploii\`
- `Alte functii ce adauga obiecte in scena, gata texturate:`
  - `renderBasicObjects(gps::Shader shader)`
  - `renderTransparentObjects(gps::Shader shader)`
  - `renderTreesLikeObjects(gps::Shader shader)`
- `drawObjects(gps::Shader shader, bool depthPass)`
  - `Functie folosita la desenarea obiectelor pentru care vreau sa am umbra`
- `renderScene()`
  - `functie folosita la randerizarea obiectelor`
- `scenePresentation()`
  - `functie cu ajutorul careia scena se plimba singura in camera`

3.1.1 .1 Functii si algoritmi

- Un algoritm interesant implementat in cadrul acestui laborator este cel de generat picaturile de ploaie
  - Abordarea aleasa consta in alegerea coordonatelor in mod aleator (atata timp cat nu depaseste un obiect mai mare(cub) a noilor coordonate, precum si viteza de cadere a acestor picaturi.
- Algoritmul Shadow Mapping
  - Tehnica multi-trecere ce utilizeaza texturi de adancime pentru a decide daca un obiect se afla sau nu in umbra
  - Sursa de lumina e considerata locatia camerei; daca un obiect din scena nu poate fi vazut din perspectiva camerei, atunci el se afla in umbra
  - Etape
    - Rasterizarea scenei din punct de vedere al luminii: valorile de adancime ale unui obiect sunt pastrate in harta de adancime; ulterior sunt adaugate la un framebuffer
    - Rasterizarea scenei dpdv al observatorului

- Se compara adancimea fiecarui fragment cu valorile din harta de adancime obtinuta anterior

- Algoritmii de animatie
  - Se bazeaza pe rotatii in jurul unui punct diferit de origine, deci e nevoie de translatia noului punct in punctul de origine, urmata de rotatie cu numarul de grade dorit, si ulterior translatia inapoi in punctul initial.
    - Intrucat exista multiple obiecte ce fac astfel de rotatii (in punctie diferite, cu grade diferite) am implementat metoda urmatoare:
      - `rotationAroundCenter(glm::mat4 anim_model, glm::vec3 rotationCenter, float rotAngle, glm::vec3 rotationAxis)`
- `Algoritmii pentru doua lumini punctiforme simultane`
  - In **twoLightShader** se va calcula pentru fiecare lumina punctiforma in parte lumina ambientala, difuza si speculara
    - `ambient[index] = att * ambientStrength * lightColor[index];`
    - `diffuse[index] = att * max(dot(normalEye, lightDirN), 0.0f) * lightColor[index];`
    - `specular[index] = att * specularStrength * specCoeff * lightColor[index];`
    - Variabila uniforma activeLight[index] va determina carte dintre cele 2 lumini e „aprinsa”
    - La final, lumina de pe obiect va fi calculata ca si combinatie dintre cele doua lumini

```
vec3 color = min((ambient[0] + diffuse[0]) *
texture(diffuseTexture, fTexCoords).rgb +
                        specular[0] *
texture(specularTexture, fTexCoords).rgb +
                        (ambient[1] + diffuse[1]) *
texture(diffuseTexture, fTexCoords).rgb +
                        specular[1] *
texture(specularTexture, fTexCoords).rgb,
                    1.0f);
```

- Algoritmul de generare a fulgerului
  - Este bazat pe valoarea atenuarii cu care se calculeaza lumina directionala si punctiforma pentru shaderul **basic.** Cu cat valoarea respectiva e mai mare, cu atat scena este mai luminoasa, deci se poate simula efectul de fulger.
  - Ca fulgerul/tunetul sa se activeze este necesar ca **rainMode** sa fie activ.
  - Ulterior, calculam cu functia `glfwGetTime()` `intervalul de timp atat pentru perioada` in care fulgerul este activ (adica atenuarea va avea valoarea 10.0f) cat si perioada pentru care acesta nu e activ (atenuarea are valoarea 1.0f)

```
    float lastTime = glfwGetTime();
    float interval = 3.0f;
    float interval2 = 1.0f;
// [...]
while (!glfwWindowShouldClose(myWindow.getWindow())) {
//[…]
        if (glfwGetTime() - lastTime > interval) {
            //animateThunder();
            thunder = true;

            lastTime = glfwGetTime();


        }
        else  if (glfwGetTime() - lastTime > interval2) {
            thunder = false;
        }
//[...] }
```

3.1.1 .2 Motivatia abordarii alese

Generarea picaturilor de ploaie la coordonate si viteze diferite sporesc ideea de miscare continua a ploii si ofera astfel scenei un aspect mai realist.

Generarea de umbre cu Shadow Mapping: mai simplu de implementat si integrat in aplicetie.

Animatia de rotatie in jurul unui punct diferit de origine implica un grad de complexitate mai mare si confera realism scenei.

3.2 Structuri de date folosite

In cadrul proiectului nu s-au definit noi structuri de date customizate, ci s-au folosit deja cele existente precum:

- Valori primitive: Gluint
- Matrice si vectori: glm::mat2, glm::vec3;

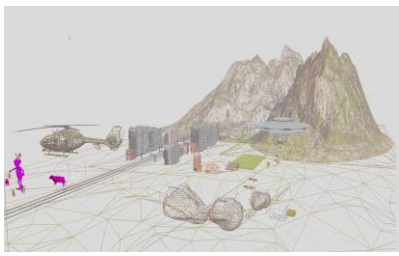
3.3 Ierarhia de clase

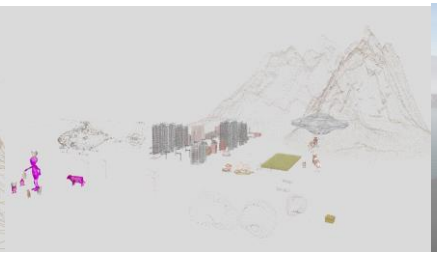
Clasele din cadrul proiectului sunt:

- Main – unde are loc implementare si executia programului principal (incarcare obiecte in scena prin intermediul shaderelor, prelucrarea coordonatelor obiectelor pentru producerea efectelor.

- Camera – clasa unde se initializeaza cameraPosition, cameraTagret, cameraUP si se proceseaza astfel incat sa sa faca posibila explorarea scenei prin tastatura sau mouse
- Model3D , Mesh, Shader, SkyBox

3.4 Scena – overview





4. Manual de utilizare

- Mouse: miscari de rotatie in camera
- W, S, D, S – sus, jos, in spate
- Q, E – rotatie stanga, dreapta in jurul axei Y
- M – afisare depth map
- T -- afisare wireframe a scenei
- Y – afisare polygon a scenei
- R – activare ploaie
- H – activare manuala fulger
- P – mod de prezentare automata al camerei
- 1,2 -- activare surse de lumina punctiforma
- E,F – modificare intensitate ceata
- J,L – rotirea sursei de lumina directionala

5. Dezvoltari ulteriora

O posibila si seminificativa imbunatatire a programului este aceea de mutare a sursei principale de lumina in directii mai specifice, spre exemplu in interiorul stalpilor de pe strada. De asemenea, aplicatia poate fi imbunatatita si prin crearea unor animatii mai complexe (rotirea elicelor atunci cand acestea sunt oblice de exemplu).

6. Concluzii

Elaborarea scenei complexe in OpenGL, ce ilustreaza subscenele din zona urbana si cea rurala, reprezinta o experienta semnificativa si formative. Prin implicsarea in acest proiect am avut oportunitatea de a imbunatati si consolida cunostintele mele in cadrul Prelucrarii Grafice.

Procesul de proiectare si dezvoltare al scenei m-a determinat sa explorez si sa aplic concepte avansate in OpenGL. Am invatat de altfel sa gestionez mai eficient resursele grafice pentru a obtine performante optime, sa manipulez iluminarea si umbrele pentru a conferi realism scenei si integrarea texturilor si obiectelor intr-un mod relevant.