**Documentatie PG**

Anastasiu Andreea Valentina

Grupa 30237

**Cuprins**

1. **Prezentare tema**
2. **Scenariu**
   1. Descriere scena si obiecte
   2. Functionalitati
3. **Detalii de implementare**
   1. Functii si algoritmi
   2. Model grafic
   3. Structuri de date
   4. Ierarhia de clase
4. **Prezentare interfata grafica utilizator**
5. **Concluzii si dezvoltari ulterioare**
6. **Bibliografie**
7. **Prezentare tema**

Proiectul surprinde o scena cu obiecte 3D care reprezinta un peisaj rural. Am ales tema aceasta deoarece am considerat ca sunt elemente care se pot mapa pe actiuniile si librariile din laborator.

1. **Scenariu**
   1. **Descrierea scenei si a obiectelor**

Scena reprezinta un mic peisaj din mediul rural format din 2 casute, capaci, o pisica, rate, un lac, 2 barcute si un avion ca element dinamic care trece deasupra acestui peisaj. Am ales sa pozitionez obiectele astfel incat sa pot crea un mediu cat mai realist.

* 1. **Functionalitati**

Pentru a vizualiza scena vom folosi camera si mouse touch-ul. Daca apasam tastele w,s,d,a ne vom misca fata-spate, dreapta-stanga si cu mouse touch-ul ne putem invartii. Putem vizualiza scena in 3 moduri: solid( tasta 0), wireframe( tasta 9) si point (tasta 8). Daca apasam tasta C am active/ dezactiva efectul de ceata si daca apasam Y vom active animatia camerei. In plus daca nu vrem sa animam automat obiectul puntem folosii sagetiile fata-spate, stanga-dreapta.

1. **Detalii de implementare**

Am asezat obiectele in scena cu ajutorul transformarilor de scalare, translatie si sclare. Obiectele (care au fost descarcate de pe internet[1]) au fost texturate cu ajutorul Blender[3], iar apoi unele au fost asezate direct in scena.

* 1. **Functii si algoritmi[2] + 3.2. Model graphic[2]**

Pentru inceput codul defineste configuratiile pentru a asigura buna functionare pe orice dipsozitiv. Am folosit functii si hedere din OpenGl, dar si altele noi pentru diferite actiuni. Au mai fost create si diferite shadere pentru desenarea sau incarcarea obiectelor.

Variabile globalea folosite sunt:

* **glWindowWidth, glWindowHeight** unde imi defines dimensiunea ferestrei
* **SHADOW\_WIDTH, SHADOW\_HEIGHT**: Dimensiunea hărții de umbre (shadow map).
* Obiecte pentru matricele de transformare (model, view, projection) și locațiile lor de uniforme.
* **gps::Camera myCamera:** Cameră pentru navigare în scenă.
* **AnimationState:** Enum pentru gestionarea stărilor de animație a avionului

Functiile folosite in proiect au fost:

* **void windowResizeCallback(GLFWwindow\* window, int width, int height);** care va actualize dimensiunea ferestrei si viewportul
* **void keyboardCallback(GLFWwindow\* window, int key, int scancode, int action, int mode);** aceasta gestioneaza controlul de la tastatura a animatiilor, camerei
* **void mouseCallback(GLFWwindow\* window, double xpos, double ypos);** gestioneaza rotatia camrei in functie de mouse
* **void animatieCamera();** imi fac o animatie de prezentare sinusoidala pentru pozitia si orientarea camerei
* **void animateNanosuit()** aici am animatia avionului
* **void initObjects();** incarc obiectele 3D cu gps::Model3D.
* **void initShaders();** incarc shader-urile pentru randare
* **void initUniforms();** seteaza matrici si alte uniforme pentru shader
* **void initSkybox();** pentru skybox
* **void initFBO();** Configurează un buffer pentru hărțile de umbre.
* **glm::mat4 computeLightSpaceTrMatrix();** Creează matricea spațiului luminii pentru calculul umbrelor.
* **void drawObjects(gps::Shader shader, bool depthPass);** Desenează toate obiectele din scenă, aplicând transformările și animațiile.

Un fragment shader foarte important este shaderStart pentru ca acolo avem functii pentru: umbre, lumina si ceata . Functiile implementate sau pe care le-am incercat sa le implementez sunt urmatoarele:

* **computePoleLighting()->** in aceasta functie am incercat sa implementez lumina punctiforma care nu mi-a iesit
* **computeLightComponents() ->** aceasta functie calculeaza contributiile de iluminare pentru o lumina directionala

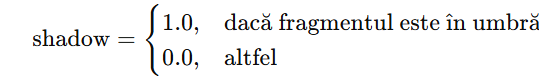
Algoritmul ar fi urmatorul

1. Normalizam: o sa normalizam vectorii de care avem nevoie: normal, directia luminii si directia de vizualizare
2. Calculam lumina ambientala: ambient=ambientStrength⋅lightColor
3. Calculam intensitatea luminii difuze: diffuse=max(dot(normalEye,lightDirN),0)⋅lightColor
4. Se calculează reflecția luminii (folosind funcția reflect) și se determină intensitatea speculară cu formula lui Phong:



* **computeShadow()** -> determina daca fragmentul e sau nu in umbra

Algoritm:

1. **Coordonate normalizate în spațiul luminii**: Transformă poziția fragmentului din coordonate luminoase în coordonate normalizate (între 0 și 1) pentru a putea accesa harta umbrelor.
2. **Determinarea umbrei**: Folosește textura hărții umbrelor (shadowMap) pentru a compara adâncimea fragmentului curent cu cea mai apropiată adâncime înregistrată de lumină. 
3. **Bias pentru artefacte**: Se adaugă un bias pentru a compensa erorile numerice și pentru a reduce artefactele precum "shadow acne".

* **computeFog() ->** Această funcție calculează factorul de ceață în funcție de distanța fragmentului față de cameră

Algoritm:

1. **Densitatea ceții**: Se definește o constantă fogDensity care controlează grosimea ceții.
2. **Distanța fragmentului**: Se calculează distanța fragmentului față de cameră.
3. **Factorul de ceață**: Formula pentru ceață este:



Rezultatul este limitat între 0 și 1.

* 1. **Structuri de date**

**1.Structuri de date pentru geometria scenei:**

* **glm::mat4 model, view, projection:** Sunt utilizate pentru a reprezenta și manipula matricile de transformare în spațiul 3D. Acestea sunt folosite pentru poziționarea, rotația, scalarea obiectelor și pentru proiecția camerei.
* **glm::mat3 normalMatrix:** Matrice utilizată pentru transformările normalei, necesară pentru calculele de iluminare corecte în shader.
* **glm::vec3 lightDir, lightColor:** Vectori pentru a stoca direcția și culoarea luminii în scenă.

**2. Structuri de date pentru gestiunea obiectelor:**

* Clase precum gps::Model3D sunt folosite pentru a încărca și a desena obiecte 3D, cum ar fi avionul, solul sau polul de lumină.
* Obiecte precum gps::Shader sunt utilizate pentru a gestiona programele shader, cum ar fi shaderul personalizat, cel pentru lumină sau cel pentru skybox.

**3. Structuri de date pentru camera și animație:**

* **gps::Camera:** Instanță a unei clase care implementează funcționalități pentru vizualizarea scenei, incluzând mișcarea camerei și rotațiile acesteia.
* **bool cameraAnimationActive:** Variabilă booleană care controlează dacă animația camerei este activă.
* **float animationTime:** Variabilă pentru a ține evidența timpului curent al animației.

**4. Structuri de date pentru interacțiunea utilizatorului:**

* **bool pressedKeys[1024]:** Tablou pentru a reține starea tastelor apăsate de utilizator.
* **float lastX, lastY:** Coordonatele ultimei poziții a mouse-ului, folosite pentru rotația camerei.
* **enum AnimationState:** Enumerație pentru stările animației avionului (TAKE\_OFF, MOVE\_FORWARD, ROTATE, LAND).

**5. Structuri pentru iluminare și umbre:**

* **GLuint shadowMapFBO, depthMapTexture:** Resurse OpenGL pentru implementarea umbrelor bazate pe tehnici de mapping.
* **glm::vec3 poleLightPos, poleLightColor:** Vectori care descriu poziția și culoarea luminii pentru un stâlp de lumină din scenă.

**6. Structuri pentru controlul graficii și randării:**

* **float glWindowWidth, glWindowHeight:** Dimensiunile ferestrei OpenGL.
* **bool fogEnabled:** Variabilă booleană care controlează activarea ceții în scenă.
* **GLuint modelLoc, viewLoc, projectionLoc:** Uniforme utilizate pentru a transmite matricile de transformare către shader.

**7. Structuri adiționale:**

* **glm::mat4 lightSpaceTrMatrix:** Matrice pentru calculul proiecției în spațiul luminii, utilizată în generarea umbrelor.
* *std::vector<const GLchar> faces:*\* Vector utilizat pentru a încărca texturile cubemap necesare skybox-ului.
  1. **Ierarhia de clase**

Proiectul implementează o arhitectură modulară, în care fiecare clasă are o responsabilitate clar definită. Acest lucru facilitează extinderea și întreținerea codului, asigurând totodată separarea preocupărilor. Structura ierarhică a claselor este descrisă mai jos:

**Clase principale**

1. **Shader**  
   Clasa Shader este responsabilă pentru încărcarea, compilarea și utilizarea shaderelor OpenGL. Aceasta abstractizează operațiile legate de gestionarea programelor shader, cum ar fi setarea uniformelor.
2. **Model3D**  
   Clasa Model3D gestionează încărcarea, stocarea și desenarea modelelor 3D utilizând biblioteca OpenGL. Include suport pentru modele complexe, cum ar fi cele importate din fișiere .obj.
3. **Camera**  
   Clasa Camera implementează logica pentru mișcarea și orientarea camerei într-un spațiu 3D. Aceasta suportă mișcări precum deplasarea înainte/înapoi, laterală și rotația pe axele X și Y.
4. **SkyBox**  
   Clasa SkyBox este responsabilă pentru crearea și desenarea unui skybox, folosind texturi cubemap pentru a simula un mediu înconjurător infinit.

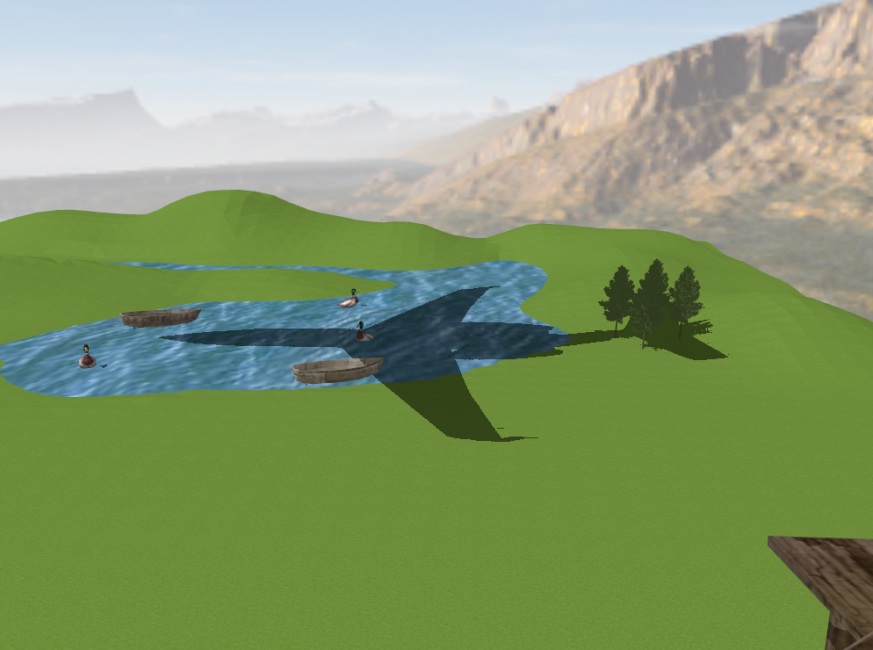
**Relații între clase**

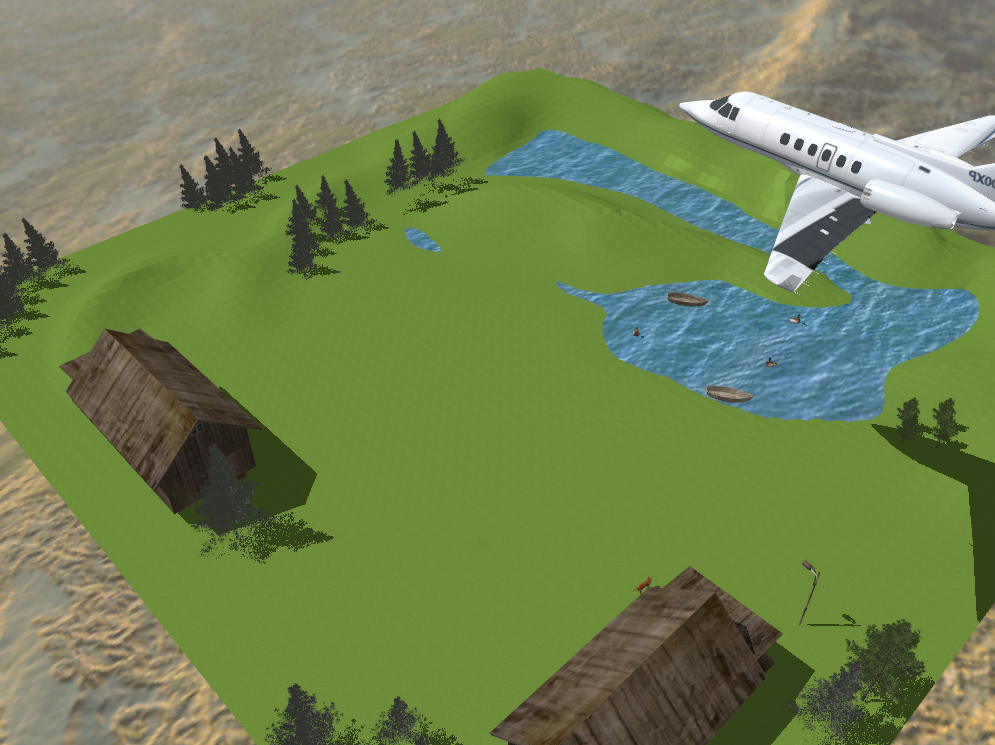
* **Shader** este utilizată de Model3D, SkyBox și alte componente pentru a seta și configura programele shader.
* **Model3D** folosește obiecte Shader pentru a desena modelele cu shader-ele corespunzătoare.
* **Camera** este utilizată pentru a genera matricele de proiecție și vizualizare (view), care sunt transmise shader-elor prin intermediul uniformelor.
* **SkyBox** se bazează pe un shader specific pentru randare și utilizează camera pentru a stabili poziția privitorului.

1. **Prezentare interfata grafica utilizator**

Pentru a deschide proiectul utilizatorul trebuie sa aiba instalat Visual Studio 2020, apo sa ruleze proiectul. Se va deschide scena si pentru a vedea o animatie de prezentare a scenei vom apasa tasta y, iar daca vrem sa incetam aceasta vom apasa din nou y. Pentru a ne misca in scena vom folosi tastele w,s,d,a care reprezinta directiile fata-spate, stanga-dreapta, iar pentru a ne roti vom folosi mouse-ul. Pentru a misca altcumva avionul putem folosi si tastele sus, jos, stanga, dreapta. Pentru a adauga efectul de ceata vom apasa tasta c, iar daca vrem sa-l scoatem vom apasa inca din nou c.







1. **Concluzii si dezvoltari ulterioare**

Proiectul poate fi dezvoltat prin adaugarea de alte efecte precum ploaia sau multiple lumini sau chiar miscarea barcilor pe anumite traiectorii.

1. **Bibliografie**

[1] Free3D. url: https://free3d.com/.

[2] Laborator PG. url: https://moodle.cs.utcluj.ro/course/view.php?id=672

[3] Tutorial Blender. url: https://moodle.cs.utcluj.ro/mod/url/view.php?id=52833