

Proiect Prelucrare Grafică

Student: Călăcean Titus
Universitatea Tehnică din Cluj-Napoca

1 Cuprins:

1. Cuprins
2. Prezentarea temei
3. Scenariul
 - 3.1. descrierea scenei și a obiectelor
 - 3.2. funcționalități
4. Detalii de implementare
 - 4.1. funcții și algoritmi
 - 4.1.1. soluții posibile
 - 4.1.2. motivarea abordării alese
 - 4.2. modelul grafic
 - 4.3. structuri de date
 - 4.4. ierarhia de clase
5. Prezentarea interfeței grafice utilizator / manual de utilizare
6. Concluzii și dezvoltări ulterioare
7. Referințe

1. Prezentarea temei

Tema proiectului constă în realizarea unei scene 3D interactive, randată în timp real folosind OpenGL. Scena de obiecte reprezintă un mic sat de munte, cu două case, un turn de observație din lemn, copaci, un lac și un cer de zi. Utilizatorul poate explora liber mediul 3D prin intermediul unei camere controlate de la tastatură.

Scopul proiectului este familiarizarea cu pipeline-ul grafic modern bazat pe shadere, cu încărcarea de modele 3D din fișiere externe, cu aplicarea de texturi și cu controlul camerei în spațiul 3D. Proiectul utilizează biblioteca GLFW pentru crearea ferestrei și gestionarea intrărilor de la tastatură, GLEW pentru încărcarea extensiilor OpenGL și GLM pentru operații pe vectori și matrice. Pentru încărcarea texturilor din fișiere imagine este folosită biblioteca stb_image.

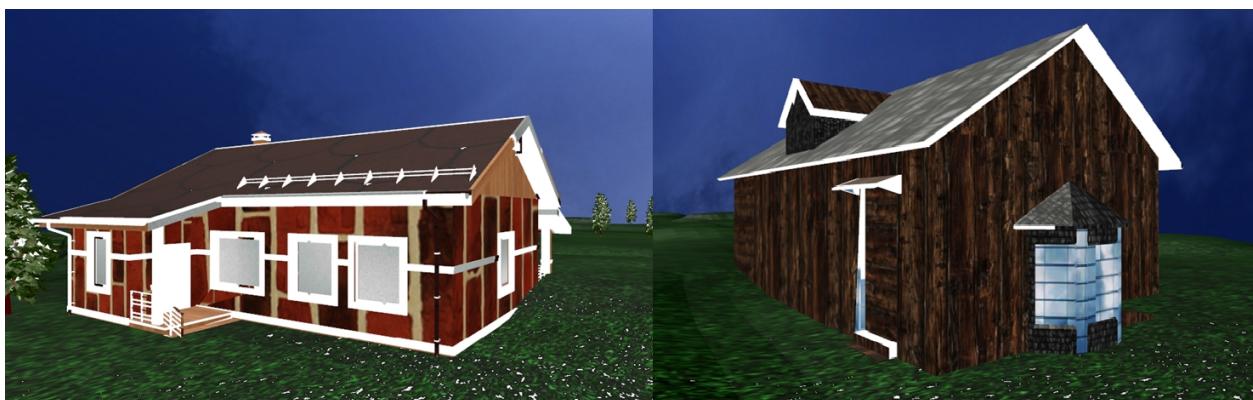


3. Scenariul

3.1. Descrierea scenei și a obiectelor

Scena reprezintă o suprafață de teren verde, ușor reliefată, înconjurată parțial de apă. Pe acest teren sunt plasate:

- **Două case de lemn**, texturate, așezate la distanță una de celalaltă, pentru a sugera un mic cătun.



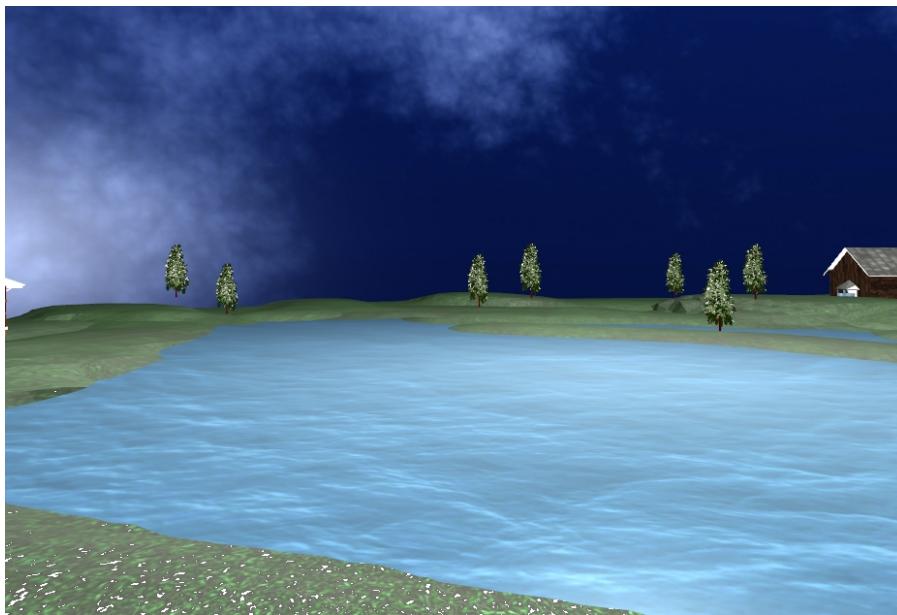
- **Un turn de observație** din lemn, amplasat într-o zonă mai înaltă a terenului, care sugerează un punct de belvedere asupra întregii zone.



- **Copaci** (în special conifere), distribuiți pe suprafața terenului, care oferă senzația unui mic pâlc de pădure.



- **Un lac** cu suprafață albastră, poziționat în partea inferioară a scenei, care reflectă cerul și contribuie la variația cromatică a peisajului.



- **Cerul** este reprezentat prin culoarea de fundal a scenei, un gradient de albastru deschis, care sugerează o zi senină. Lumina soarelui cade de sus, din zona centrală a imaginii, creând efecte de iluminare pe obiecte.



Toate aceste obiecte sunt incluse într-un singur fișier de model 3D, alta_scena.obj, încărcat la rulare prin clasa gps::Model3D. Poziționarea, scara și orientarea obiectelor sunt stabilite în fișierul .obj, iar în cod scena este transformată global printr-o rotație de 180° în jurul axei Oy (pentru a corecta orientarea în raport cu camera).

Camera este inițial poziționată la coordonatele (0, 5, 15) și privește către centrul scenei, astfel încât utilizatorul vede din start satul, lacul și o bună parte din cer.

3.2. Funcționalități

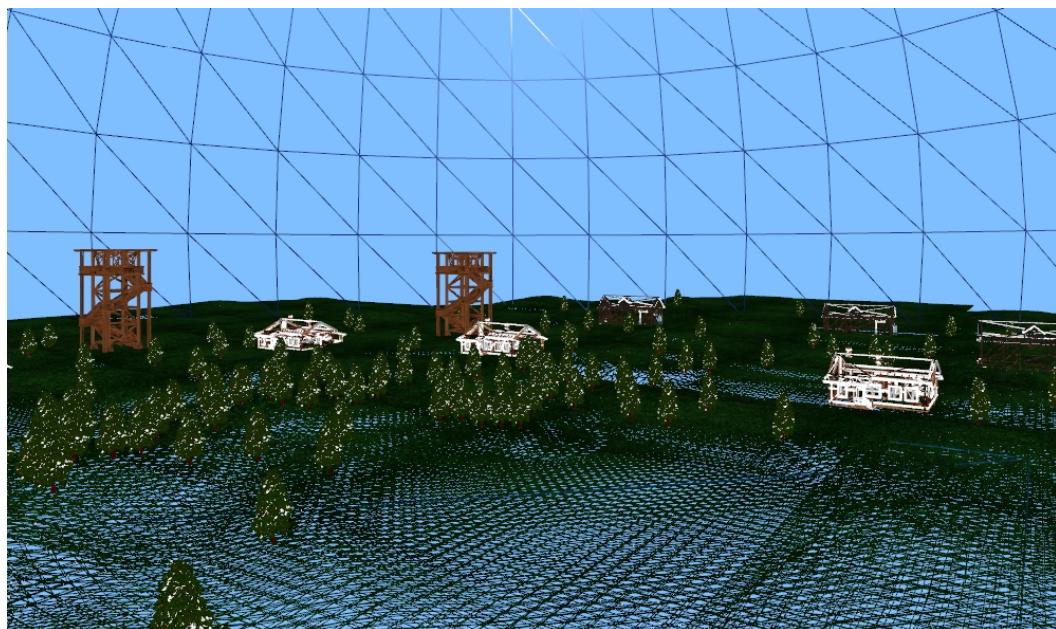
Proiectul oferă următoarele funcționalități principale:

- **Navigare în timp real prin scenă** – utilizatorul controlează camera în spațiul 3D cu tastele:
 - W / S – mișcare înainte / înapoi;
 - A / D – mișcare laterală stânga / dreapta;
 - SPACE – urcare (mișcare în sus);

- LEFT_CONTROL – coborâre (mișcare în jos);
- Q / E – rotire a camerei stânga / dreapta (yaw).
- **Moduri diferite de randare**, care pot fi schimbată din tastatură:
 - 1 – modul **solid**: poligoanele sunt umplute și texturile sunt afișate normal;



- 2 – modul **wireframe**: se văd doar muchiile obiectelor;



- 3 – modul **poligonal**: sunt afișate doar poligoanele obiectelor;



- 4 – modul **smooth**: randare solidă cu iluminare și efecte de reflexie calculate în shader.



- **Iluminare de scenă** – scena este iluminată de 2 surse de lumină poziționate deasupra satului in direcții diferite, la coordonatele (60, 60, 20) și (-40, 20, -20).

Pozitia luminii, culoarea acesteia si pozitia camerei sunt transmise catre shader pentru a calcula iluminarea fragmentelor.

- **Parametru de timp pentru animatii** – variabila time, calculata cu glfwGetTime(), este transmisa shader-ului. Acest parametru poate fi folosit pentru efecte dinamice, precum mici variatii ale luminii sau ale reflexiilor pe suprafata lacului.

Proiectul este interactiv: utilizatorul poate explora scena din mai multe pozitii si cu moduri de vizualizare diferite, ceea ce permite observarea detaliilor modelelor si a efectelor de iluminare.

3.2. Efecte si prezentare

Aplicatia foloseste 2 efecte de fog si wind si o prezentare asupra scenei, prezentarea fiind in 3 directii inainte, stanga si dreapta. Prezentarea si efectele de ceata si vant se folosesc prin urmatoarele taste descrise mai jos in aceasta documentatie dupa cum urmeaza chiar si bineintele mai jos:).

- **Mod de prezentare** - Tasta P
- **Efect de ceata** – Tasta F
- **Efect de vant** - Tasta R

4. Detalii de implementare

4.1. Funcții și algoritmi

Codul principal al aplicației este organizat în jurul câtorva funcții esențiale:

- `initOpenGLWindow()` – se ocupă de inițializarea bibliotecii GLFW, de crearea ferestrei, de asocierea contextului OpenGL și de setarea callback-urilor pentru redimensionare și tastatură. Tot aici se inițializează GLEW (pe platformele non-Apple) și se afișează versiunea de OpenGL suportată de placă.
- `keyboardCallback()` – gestionează evenimentele de la tastatură. La apăsarea unei taste se actualizează vectorul `pressedKeys` și se schimbă modul de randare (tastele 1–4) sau se închide fereastra (tasta ESC).
- `mouseCallback()` – este definită pentru a putea trata în viitor mișcarea mouse-ului; în această versiune nu este folosită.
- `processMovement(float deltaTime)` – interpretează starea vectorului `pressedKeys` și translatează / rotește camera folosind metodele clasei `gps::Camera`. Se ține cont de `deltaTime` pentru a obține mișcare lină, independentă de numărul de FPS-uri.
- `ReadTextureFromFile(const char* file_name)` – încarcă o imagine de pe disc folosind `stbi_load`, inversează liniile imaginii (pentru a corespunde cu sistemul de coordonate OpenGL), creează și configerează un obiect `GL_TEXTURE_2D` și generează mipmap-uri.
- `renderScene()` – pregătește contextul de randare (curățarea bufferelor, setarea `viewport`-ului), activează shader-ul, trimitе uniforme precum `renderMode`, `time`, matricile `view` și `model`, poziția și culoarea luminii și poziția camerei, apoi apeleză metoda `Draw()` a obiectului `myModel` pentru a desena întreaga scenă.
- `cleanup()` – eliberează resursele de pe GPU (VBO, EBO, VAO, texturi) și distrugе fereastra GLFW.

Bucla principală din `main()` calculează `deltaTime`, apeleză `processMovement()` pentru a actualiza camera, apoi `renderScene()` pentru afișarea unui nou frame, urmând să proceseze evenimentele de la sistemul de operare și să schimbe bufferele ferestrei.

4.1.1. Soluții posibile

Pentru implementarea acestei teme existau mai multe variante:

- **Crearea scenei prin cod** – definirea manuală a tuturor vertecșilor și indicilor pentru teren, case, turn, copaci și lac, eventual folosind mai multe VAO-uri și VBO-uri separate pentru fiecare obiect.
- **Încărcarea fiecărui obiect din fișiere .obj distincte** – o casă, un copac, un turn, terenul etc., fiecare instantiat și poziționat din cod prin mai multe matrice model.
- **Încărcarea unei singure scene complete dintr-un fișier .obj** – toate obiectele fiind modelate în prealabil într-un program de modelare 3D (Blender, 3ds Max etc.) și salvate într-un singur fișier.
- **Folosirea unei camere fixe** (doar rotație automată în jurul scenei) versus **o cameră controlată complet de utilizator** (WASD + rotație).
- Randare cu **OpenGL vechi (immediate mode, glBegin/glEnd)** sau cu **OpenGL modern**, bazat pe shadere, VAO/VBO și uniforme.

4.1.2. Motivarea abordării alese

În proiectul de față a fost aleasă varianta:

- **Încărcarea unei scene complete dintr-un fișier .obj** (alta_scena.obj) prin clasa `gps : :Model3D`. Această abordare mută complexitatea modelării 3D în afara codului și permite realizarea unei scene mai detaliate, folosind unelte specializate de modelare.
- **OpenGL modern, core profile**, cu shadere personalizate, ceea ce este aliniat cu cerințele actuale ale dezvoltării grafice și permite extinderea ulterioară cu efecte avansate (iluminare mai complexă, umbre, reflexii).
- **Camerei libere de tip „fly-camera”**, controlată de utilizator, pentru a oferi o experiență interactivă și pentru a putea observa scena din mai multe unghiuri.
- **Mai multe moduri de randare (solid, wireframe, puncte)**, deoarece acestea sunt utile atât pentru debug (verificarea topologiei mesh-urilor), cât și pentru ilustrarea diferenței dintre reprezentările de nivel jos (triunghiuri, vârfuri) și randarea finală.

Această abordare oferă un bun compromis între complexitate și claritate: codul rămâne relativ concis, în timp ce scena este suficient de bogată vizual.

4.2. Modelul grafic

Modelul grafic este bazat pe pipeline-ul standard OpenGL:

- Se utilizează o **proiecție în perspectivă** definită cu `glm::perspective`, cu un unghi de vizualizare de 55° , un raport lățime/înălțime calculat din dimensiunea ferestrei și un plan apropiat la 0.1 și îndepărtat la 1000.
- Matricea de **vizualizare (view)** este generată de clasa `gps::Camera` și descrie poziția și orientarea camerei în scenă.
- Matricea **model** este inițial identitatea, apoi scena este rotită cu 180° în jurul axei `Oy` pentru a corecta orientarea.
- Combinația `projection * view * model` este folosită în shaderul de vârfuri pentru a transforma coordonatele din spațiul obiectului până în spațiul clip.
- Iluminarea este calculată în shaderul de fragmente, folosind poziția luminii (`lightPos`), culoarea ei (`lightColor`) și poziția camerei (`viewPos`). Texturile sunt aplicate pe suprafața obiectelor (iarbă, lemn, apă, acoperișuri) pentru a obține un aspect realist.
- Cerul este reprezentat prin culoarea de fond setată cu `glClearColor(0.5f, 0.75f, 1.0f, 1.0f)`, creând un gradient de albastru deschis specific unei zile senină.

Scenei i se aplică teste de adâncime (`glEnable(GL_DEPTH_TEST)`) pentru a asigura ordonarea corectă a obiectelor în spațiu și culling de fețe din spate (`glCullFace(GL_BACK)`) pentru optimizare.

4.3. Structuri de date

Principalele structuri de date utilizate în aplicație sunt:

- **Vectorul de taste apăsată**: `bool pressedKeys[1024]`; – reține pentru fiecare cod de tastă dacă este apăsat sau nu. Această abordare permite tratarea simultană a mai multor taste (de exemplu, mers înainte și în lateral în același timp).
- **Matrici și vectori 3D** din biblioteca GLM:
 - `glm::mat4 model, view, projection;`
 - `glm::vec3 lightPos, lightColor'`

Aceste structuri reprezintă transformările geometrice și parametrii de iluminare.

- **Identifieri OpenGL** pentru resurse:
 - GLuint verticesVBO, verticesEBO, objectVAO, texture; – ID-uri pentru buffer-ele de vertecși, indici, obiectul VAO și texturi.
- **Structuri interne ale claselor `gps::Model3D`, `gps::Shader` și `gps::Camera`**, care gestionează, respectiv, datele mesh-ului, programul de shadere și parametrii camerei (poziție, direcție, vector up).

Deși în cod există și un exemplu simplu de VAO/VBO (`vertexData`, `vertexIndices` și funcția `loadTriangleData()`), acesta este marcat ca „unused” și are rol demonstrativ, scena reală fiind desenată exclusiv prin `Model3D`.

4.4. Ierarhia de clase

Aplicația folosește mai multe clase definite în fișiere separate, în spațiul de nume `gps`:

- **`gps::Camera`** – reprezintă camera din scenă.
 - Atribute tipice: poziție, direcție de privire, vector up, unghiuri de rotire.
 - Metode importante (folosite în cod):
 - `getViewMatrix()` – întoarce matricea de vizualizare;
 - `move(direction, speed)` – translatează camera în funcție de direcția specificată (forward, backward, left, right, up, down);
 - `rotate(deltaPitch, deltaYaw)` – rotește camera în jurul axelor.
- **`gps::Shader`** – încapsulează un program de shadere OpenGL.
 - Metode importante:
 - `loadShader(vertexPath, fragmentPath)` – încarcă, compilează și link-uește shaderele;
 - `useShaderProgram()` – activează programul de shadere pentru randare.

- ID-ul programului este accesibil prin membrul `shaderProgram`, utilizat pentru setarea uniformelor.
- **gps : :Model3D** – se ocupă de încărcarea și randarea modelelor 3D din fișiere externe (.obj).
 - Metode importante:
 - `LoadModel(modelPath, texturesDir)` – încarcă geometria și texturile asociate;
 - `Draw(shader)` – desenează modelul folosind shaderul specificat.
 - Intern, clasa gestionează mesh-uri, texturi și VAO/VBO-uri necesare pentru randare.

Clasa `main` (fișierul `main.cpp`) acționează ca un orchestrator: initializează fereastra și contextul OpenGL, creează instanțele acestor clase (`myCamera`, `myCustomShader`, `myModel`), apoi, în bucla principală, actualizează camera și apelează funcțiile de randare.

5. Prezentarea interfeței grafice utilizator / Manual de utilizare

Aplicația rulează într-o fereastră OpenGL în care este randată scena 3D. Interfața este minimalistă, fără meniuri sau panouri grafice, deoarece interacțiunea cu scena se realizează exclusiv prin tastatură. În continuare sunt prezentate toate controalele disponibile:

5.1. Controale pentru mișcarea camerei

Tastă	Funcție
W	Merge înainte
A	Merge înapoi
S	Deplasare stânga
D	Deplasare dreapta
SPACE	Urcă
LEFT CONTROL	Coboară
Q	Rotește camera spre stânga
E	Rotește camera spre dreapta

Mișcarea este proporțională cu deltaTime, ceea ce asigură o viteză constantă indiferent de numărul de cadre pe secundă.

5.2. Moduri de randare

Utilizatorul poate modifica modul de vizualizare a obiectelor 3D folosind tastele numerice:

- 1 - Solid
- 2 - Wireframe
- 3 - Points
- 4 - Smooth Shading

5.3. Ieșire din aplicație

- ESC – închide fereastra și oprește aplicația.

5.4. Translatie

1. Tasta I - deplasare în față
2. Tasta K - deplasare înapoi
3. Tasta J - deplasare stânga
4. Tasta L - deplasare dreapta
5. Tasta U - deplasare sus
6. Tasta O - deplasare jos

5.5. Scalare / Rotatie

1. Tasta Z - zoom în scenă (scalare)
2. Tasta x - zoom out scenă (scalare)
3. Tastele T/G - rotatie pe Ox, Tastele Y/H - rotatie pe Oy, Tastele B/N - rotatie pe Oz

5.7. Comportamentul general al aplicației

La lansare, aplicația:

1. Deschide o fereastră OpenGL 4.1 Core Profile.
2. Încarcă modelul 3D complet al scenei (alta_scena.obj).
3. Inițializează shaderul și textura.
4. Plasează camera la coordonatele (0, 5, 15) și privește către centrul satului.
5. Pornește bucla de randare, actualizând la fiecare frame poziția camerei și afișând scena.

6. Concluzii și dezvoltări ulterioare

6.1. Concluzii

Proiectul realizat îndeplinește cerințele de bază ale unei aplicații grafice 3D moderne:

- Folosește **OpenGL modern (core profile)** și shadere personalizate.
- Permite **explorarea interactivă** a unei scene complexe dinamic iluminate.
- Integrează un **model 3D complet**, cu texturi, teren, case, lac, copaci și un turn de observație.
- Oferă mai multe **moduri de randare**, utile atât pentru vizualizare, cât și pentru analiză geometrică.

Pe lângă randarea statică a obiectelor, aplicația include un sistem de iluminare cu o sursă unică de lumină poziționată deasupra scenei, rezultând umbre și reflexii realiste. De asemenea, folosirea deltaTime asigură o mișcare fluidă și independentă de performanța sistemului.

6.2. Dezvoltări ulterioare

Proiectul poate fi extins în multiple direcții, precum:

- **Iluminare avansată**
 - Implementarea unui model de iluminare Phong complet sau PBR (Physically Based Rendering).
 - Adăugarea mai multor surse de lumină (spotlight, point-lights, lumini direcționale).
- **Umbre dinamice**
 - Shadow mapping pentru umbre realiste generate de turn, case și copaci.
- **Elemente animate**
 - Animația apei (valuri, reflexii dinamice).

- Animarea cerului (soare care se mișcă, cer dinamic).

- **Optimizări**

- Frustum culling și instancing pentru randarea eficientă a unui număr mare de copaci.
- Încărcarea scenei în mai multe VAO-uri pentru un control mai bun al fiecărui obiect.

- **Interacțiune extinsă**

- Control cu mouse-ul (pitch & yaw).
- Mod foto (captură de ecran în timp real).
- Coliziuni pentru cameră (nu poate trece prin case sau copaci).

Acste direcții permit transformarea proiectului într-un motor grafic 3D mai complet sau într-o aplicație cu potențial de joc.

7. Referințe

GLFW/GLEW – Graphics Library Framework

<https://www.glfw.org/>

GLM – OpenGL Mathematics

<https://github.com/g-truc/glm>

stb_image.h – Sean Barrett

<https://github.com/nothings/stb>

Tutorial Blender - YouTube

https://www.youtube.com/playlist?list=PLrgcDEgRZ_kndoWmRkAK4Y7ToJdOf-OSM

Laboratoare curs PG - YouTube

<https://moodle.cs.utcluj.ro/course/view.php?id=773>