

Universitatea Tehnică din Cluj-Napoca
Facultatea de Automatică și Calculatoare

Documentație

Proiect Prelucrare Grafică

Student: Maria-Magdalena Cret

Grupa 30223

Anul Universitar 2024-2025

Cuprins

1 Introducere	3
1.1 Context	3
1.2 Scenariu	3
2 Modelarea șcenei	3
2.1 Descrierea șcenei	3
2.2 Descrierea obiectelor integrate	5
3 Implementare	5
3.1 Ierarhia de clase	5
3.2 Vizualizarea șcenei	5
3.2.1 Scalare	6
3.2.2 Traslație	7
3.2.3 Rotație	8
3.2.4 Mișcarea în scenă	8
3.2.5 Utilizare mouse, respectiv tastatură	9
3.2.6 Animația de prezentare	10
3.3 Metode de iluminare	11
3.3.1 Lumina ambientală	11
3.4 Metode de vizualizare a șcenei	11
3.4.1 Vizualizare Solid	11
3.4.2 Vizualizare Smooth	11
3.4.3 Vizualizare Wireframe	12
3.4.4 Vizualizare Poligonal	12
4 Realizarea șcenei	13
4.1 Fotorealism	13
4.2 Ceată	15
4.3 SkyBox	16
4.4 Sonorizare	17
5 Animații	17
6 Manual de utilizare	19
7 Concluzii	19
Referințe	21

1 Introducere

1.1 Context

Acest proiect urmărește crearea unei aplicații în API-ul OpenGL, care are ca scop redarea unei scene 3D. Scena trebuie să conțină mai multe obiecte prezentate într-un mod realist. Acestea se vor prezenta în acest mod pe baza unor tehnici precum iluminare și animații. Nu va lipsii nici generarea de efecte precum ceață și/sau ploaia/ninsoarea. Utilizatorul va putea naviga prin scenă cu ajutorul tastaturii și al mouse-ului. Totodată trebuie să existe o prezentare cameră. Contextul scenei este un cartier rezidențial situat în munți care îmbină modernul cu zonele muntoase, oferind senzația de siguranță și confort.

1.2 Scenariu

Într-o dimineață liniștită de primăvară, cartierul rezidențial se trezește încet la viață. Casele moderne, cu arhitectură contemporană, se înșiră elegant de-a lungul aleilor îngrijite, fiecare cu personalitatea sa distinctă. Fațadele clădirilor în tonuri de gri, bej și alb strălucesc în lumina blândă a soarelui. Blocurile sunt impunătoare și moderne, oferind senzația de confort. Pe aleea principală, copaci tineri, plantați la distanțe egale, oferă umbră. Bâncile moderne de lemn, amplasate strategic la intersecții, invită rezidenții la momente de relaxare. În centrul cartierului, un parc modern cu loc de joacă pentru copii așteaptă să prindă viață. Leagănele colorate și toboganele din materiale eco-friendly sunt încă pustii, dar în curând vor răsuna de râsetele copiilor. O mașină roșie se situează pe strada principală și așteaptă să pornească într-o călătorie. Oamenii se grăbesc să-și realizeze activitățile zilnice. Seara, când soarele începe să coboare se creează o atmosferă caldă și primitoare.

2 Modelarea scenei

2.1 Descrierea scenei

Șcena este creată în Blender, s-a urmărit descrierea de mai sus a cartierului rezidențial situat într-o zonă montană. Șcena este creată dintr-o multitudine de obiecte importate ca .obj în Blender și fixate în anumite puncte fixe, estetic. La final, șcena este exportată ca un singur obiect pentru a putea fi pusă în OpenGL, urmând a se realiza operațiile de cameră și celelalte cerințe pe aceasta. Mai jos sunt atașate imagini cu realizarea șcenei în Blender. Fiecare obiect importat a trecut operații de scalare, rotație și traslație. Totodată s-au realizat texturări pe fiecare dintre ele. Un obiect are mai multe texturi. Aceste texturi sunt imagini în format .jpg, .png. Se cunoaște aspectul că un obiect odată importat în OpenGL, nu poate avea mai mult de o singură textură. Astfel că s-a utilizat modul de edit pe fiecare obiect texturat cu mai mult de o textură și la apăsarea tastelor Ctrl + P, obiectul a fost împărțit în numărul de obiecte echivalent numărului de texturi, o textură fiind atribuită fiecărui obiect de acest fel. S-a realizat în acest mod o exportare corectă a șcenei ca .obj și o importare corectă în OpenGL, fiecare obiect fiind texturat corespunzător. Acolo unde nu se realizează texturare corectă, de obicei părțile din obiecte sunt colorate de OpenGL cu negru.

Exportul în format .obj generează două fișiere:

- **Fișierul .obj:** Conține coordonatele geometriei (vârfuri, normale, etc.).
- **Fișierul .mtl:** Conține informații despre materiale.

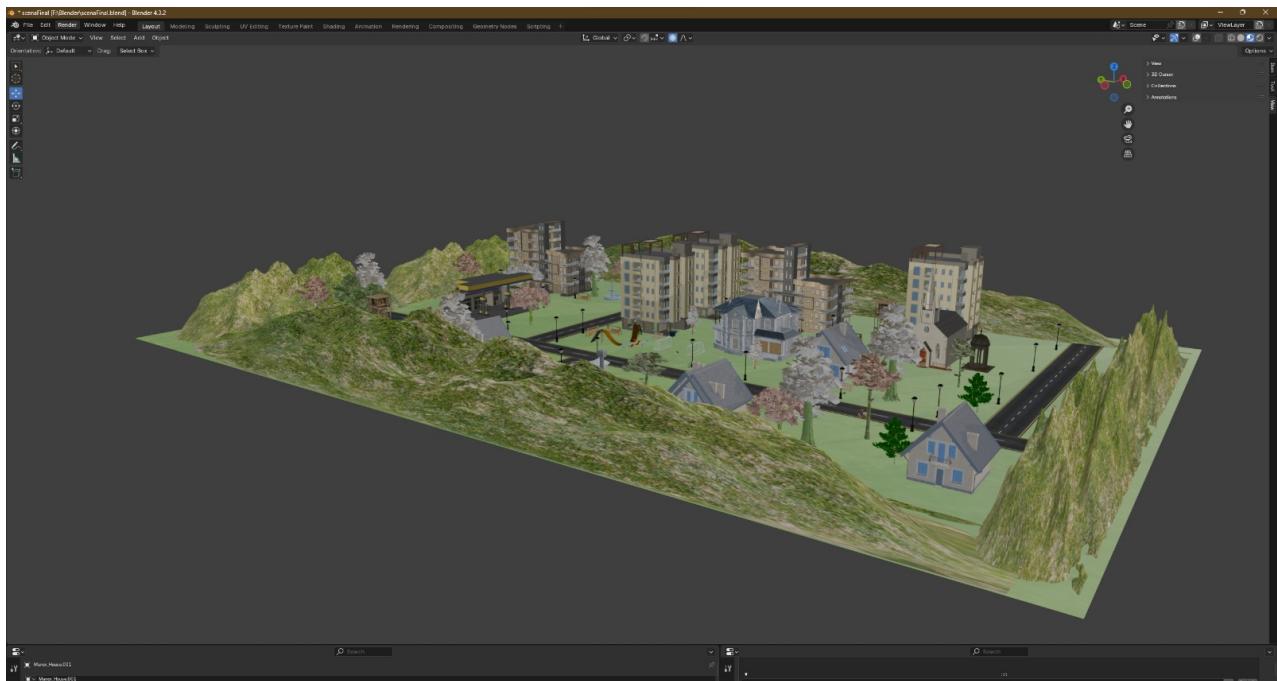


Figura 1: Scena realizată în Blender - prima vizualizare

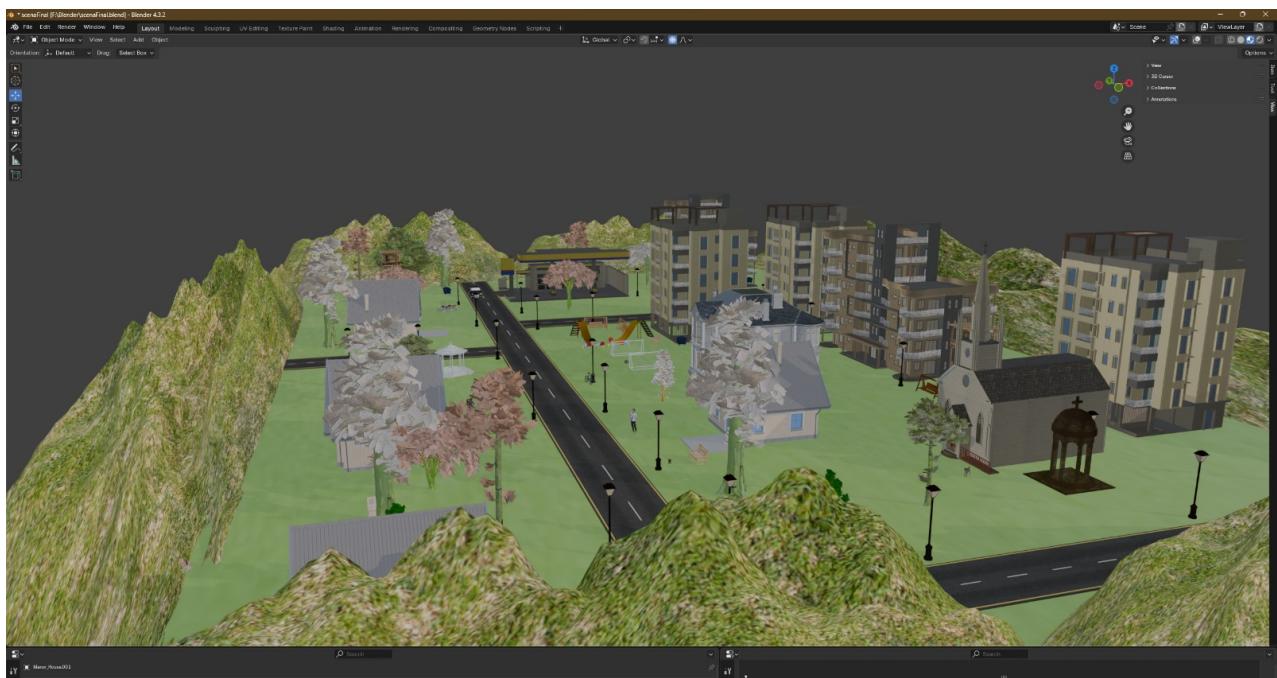


Figura 2: Scena realizată în Blender - a doua vizualizare

2.2 Descrierea obiectelor integrate

Referitor la obiectele auxiliare exportate ca .obj și integrate în OpenGL, mai există două obiecte asupra cărora s-au creat ulterior animații: o mașină și păsări. Acestea sunt puse individual în OpenGL pentru ca să se poată crea animații cu acestea. Pentru a se realiza acest lucru în OpenGL, scenei generale i se va dezactiva vizibilitate, se va păstra vizibilitatea însă obiectului dorit, care se exportă sub format .obj.

3 Implementare

3.1 Ierarhia de clase

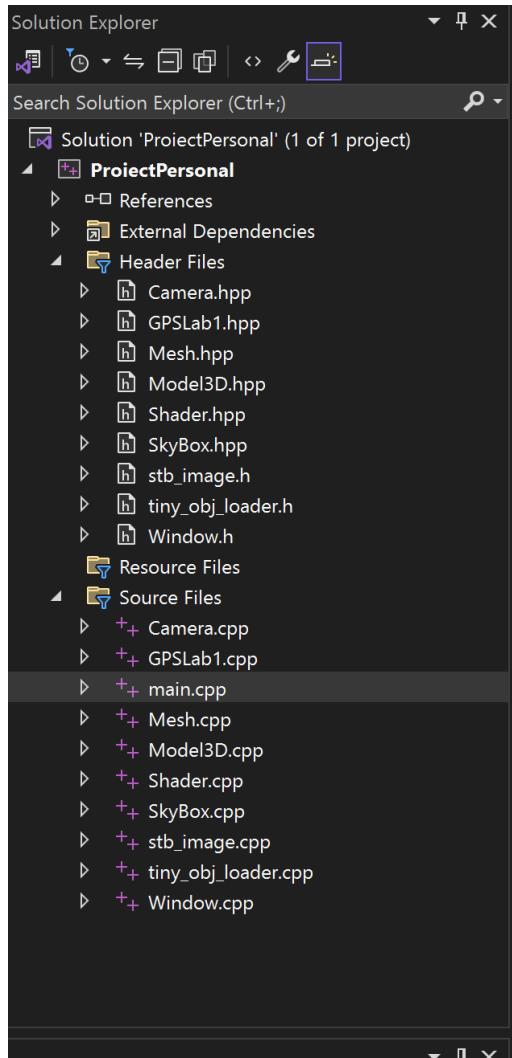


Figura 3: Ierarhia de clase

3.2 Vizualizarea scenei

Pentru implementarea proiectului s-a utilizat Visual Studio cu setup-ul pentru OpenGL. Odată ce s-a exportat scena din Blender în format .obj, a fost introdusă în fișierul corespunzător

obiectelor din template-ul proiectului și a fost pusă în OpenGL în fișierul *main.cpp* din proiect. Acest lucru s-a realizat cu toate obiectele dorite să fie vizibile în proiect, pe baza metodei *LoadModel*, aşa cum se poate observa în codul de mai jos:

```
void initModels() {
    teapot.LoadModel("models/teapot/final.obj");
    lightCube.LoadModel("models/cube/cube.obj");
    screenQuad.LoadModel("models/quad/quad.obj");
    car.LoadModel("models/car/car.obj");
    bird.LoadModel("models/bird/bird.obj");
}
```

Pentru a putea fi văzută scenă din anumite unghiuri se realizează poziționarea camerei. Astfel că se initializează un obiect de tipul **gps::Camera**, utilizând constructorul clasei pentru a seta poziția, direcția de privire și vectorul "up" (sus), care definesc orientarea camerei în spațiu.

```
gps::Camera myCamera(
    glm::vec3(0.0f, 10.0f, 3.0f),
    glm::vec3(0.0f, 0.0f, -20.0f),
    glm::vec3(0.0f, 1.0f, 0.0f)
);
```

Prima linie reprezintă poziția camerei, a doua punctul spre care privește camera, iar a treia este vectorul care reprezintă direcția în sus. Acestea rămânând constante. Dat fiind faptul că poziția camerei nu se modifică, se va modifica poziția obiectelor. Fiecare obiect din scenă este mutat prin actualizarea poziției sale în spațiul 3D. Acest lucru se face prin modificarea matricei de model a fiecărui obiect:

$$M_{\text{final}} = T \cdot R \cdot S$$

unde:

- T : Translație.
- R : Rotire.
- S : Scalare.

3.2.1 Scalare

Scalarea modifică dimensiunea unui obiect prin multiplicarea coordonatelor fiecărui punct al acestuia cu un factor de scalare. Această operație este utilizată pentru:

- A mări sau micșora un obiect.
- A schimba proporțiile unui obiect (non-uniform).

În grafica 3D, scalarea se efectuează utilizând o matrice de transformare. O matrice de scalare pentru un spațiu 3D este de forma:

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

unde s_x, s_y, s_z sunt factorii de scalare pe axele x, y și z .

Dacă $s_x = s_y = s_z$, scalarea este uniformă. Dacă s_x, s_y și s_z au valori diferite, scalarea este non-uniformă.

Pentru a aplica scalarea, coordonatele fiecărui punct al obiectului $(x, y, z, 1)$ sunt măritate cu matricea de scalare S :

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rezultatul (x', y', z') reprezintă noua poziție a punctului după scalare.

Implementarea operației de rotație în proiectul OpenGL are loc în fișierul **Camera.cpp**.

3.2.2 Traslație

Traslația este o transformare geometrică care mută un obiect dintr-o poziție în alta, fără a-i schimba forma sau orientarea. În grafica 3D, traslația se efectuează utilizând o matrice de transformare.

Matricea de traslație pentru un spațiu 3D este de forma:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

unde t_x, t_y , și t_z sunt valorile de traslație pe axele x, y și z .

Pentru a aplica traslația, coordonatele fiecărui punct al obiectului $(x, y, z, 1)$ sunt măritate cu matricea de traslație T :

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rezultatul (x', y', z') reprezintă noua poziție a punctului după aplicarea traslației. Implementarea operației de rotație în proiectul OpenGL are loc în fișierul **Camera.cpp**.

3.2.3 Rotație

Rotăția este o transformare geometrică care rotește un obiect în jurul unui punct fix sau a unei axe. În grafica 3D, rotăția se efectuează utilizând matrice de rotăție. Matricea de rotăție depinde de axa în jurul căreia se face rotăția.

Rotăție în jurul axei x : Matricea de rotăție pentru o rotăție în jurul axei x de un unghi θ este:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotăție în jurul axei y : Matricea de rotăție pentru o rotăție în jurul axei y de un unghi θ este:

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotăție în jurul axei z : Matricea de rotăție pentru o rotăție în jurul axei z de un unghi θ este:

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Aplicarea rotăției: Pentru a aplica rotăția, coordonatele fiecărui punct al obiectului $(x, y, z, 1)$ sunt multiplicate cu matricea de rotăție corespunzătoare axei de rotăție aleasă. De exemplu, pentru o rotăție în jurul axei z :

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rezultatul (x', y', z') reprezintă noua poziție a punctului după rotăție.

Implementarea operației de rotăție în proiectul OpenGL are loc în fișierul **Camera.cpp**. S-a implementat operația de rotăție în jurul axei Y.

3.2.4 Mișcarea în scenă

Mișcarea în scena OpenGL presupune manipularea poziției camerei sau a obiectelor pentru a crea un efect de dinamism.

Astfel că pe baza unor taste de la tastatură se pot realiza mișcări în scenă, precum: în față, în spate, la stânga, la dreapta, respectiv rotății pe axa Y (spre stânga) și pe axa -Y (spre dreapta).

S-a realizat și coliziunea cu scenă în partea de jos, mai precis nu se poate coborî prin scenă atunci când, din punct de vedere al coordonatelor, utilizatorul este situat pe aceasta. Acest lucru s-a implementat pe baza unei constrângeri a tastei **W**, care odată apăsată, realizează mișcarea în față. Constrângerea fiind următoarea:

```
if (pressedKeys[GLFW_KEY_W] && (myCamera.getPosition().y > 1))
```

Constrângerea pe axa *Y*:

- Codul verifică poziția camerei pe axa *Y* folosind:

```
if myCamera.getPosition().y > 1
```

- Această verificare previne mișcarea camerei înainte dacă poziția *Y* scade sub valoarea 1.
- Este o metodă simplă de a evita ca utilizatorul să „cadă” sub un anumit nivel, cum ar fi solul.

3.2.5 Utilizare mouse, respectiv tastatură

S-a realizat mișcarea mouse-ului în fișierul *main.cpp*. Funcția **mouseCallback**, gestionează mișările mouse-ului într-o aplicație OpenGL folosind GLFW. Funcția este folosită pentru a actualiza direcția camerei (yaw și pitch) pe baza mișcărilor mouse-ului, ceea ce permite utilizatorului să ”priviească” în jur în scena 3D. Inițializarea poziției mouse-ului are implementarea următoare:

```
if (firstMouse) {
    lastX = xpos;
    lastY = ypos;
    firstMouse = false; }
```

Funcția **mouseCallback**:

- Determină mișcarea mouse-ului.
- Calculează noile unghiuri de rotație ale camerei.
- Actualizează matricea de vizualizare și alte date importante (matricea normală, direcția luminii) pentru a reda scena corect în funcție de noua orientare a camerei.

Utilizarea tastaturii are loc pe baza funcției **keyboardCallback**. Pe baza acesteia se detectează evenimentele de tastatură: Funcția **keyboardCallback** este apelată de fiecare dată când o tastă este apăsată sau eliberată. Totodată, acesta procesează apăsările de taste și poate iniția acțiuni precum mișcarea camerei, schimbarea modului de vizualizare, schimbarea obiectelor sau chiar închiderea aplicației.

```
if (key >= 0 && key < 1024)
```

Aici, **key** reprezintă codul tastaturii pentru tasta apăsată (de exemplu, **GLFW_KEY_W** pentru tasta ”W”). Această condiție **if** verifică dacă codul tastaturii este valid, presupunând că GLFW folosește un număr întreg pentru identificarea tastelor și că există maximum 1024 taste.

Verificarea acțiunii (apăsare sau eliberare):

Pentru a determina dacă tasta a fost apăsată sau eliberată, verificăm acțiunea efectuată asupra tastei cu următorul cod:

```
if (action == GLFW_PRESS)
```

action reprezintă tipul acțiunii efectuate asupra tastei: - GLFW_PRESS înseamnă că tasta a fost apăsată. - GLFW_RELEASE înseamnă că tasta a fost eliberată.

Actualizarea stării tastelor:

Când o tastă este apăsată (action == GLFW_PRESS), actualizăm starea tastelor astfel:

```
pressedKeys[key] = true;
```

Aici, un array sau vector **pressedKeys** este actualizat pentru a marca tasta corespunzătoare ca fiind apăsată (**true**).

Când o tastă este eliberată (action == GLFW_RELEASE), actualizăm starea tastelor astfel:

```
pressedKeys[key] = false;
```

3.2.6 Animația de prezentare

Animația de prezentare face referire la modul preview al scenei. Codul de mai jos reprezintă implementarea specifică a **previzualizării scenei**, unde camera efectuează o rotație în jurul unui punct central. Aceasta permite utilizatorului să obțină o imagine completă asupra distribuției și a detaliilor din scenă.

```
if (isPreviewActive) {
    angleForRotation += 0.5f;
    myCamera.scenePreview(angleForRotation);
}
```

- **Determinarea stării de previzualizare:** Variabila **isPreviewActive** este un flag care indică dacă previzualizarea scenei este activată. Dacă această variabilă este setată pe **true**, camera începe să se miște, generând o animație continuă.
- **Calcularea rotației:** Variabila **angleForRotation** este utilizată pentru a controla unghiul curent al camerei în rotația sa. Fiecare iterație adaugă un increment (**0.5f**), acest lucru va duce la o mișcare treptată a camerei.
- **Rotația camerei în jurul punctului central:**
`myCamera.scenePreview(angleForRotation)` este metoda pe baza căreia se calculează poziția camerei pe o traiectorie circulară. Camera este rotită folosind funcții trigonometrice:
 - **Sinus (sin)** și **cosinus (cos)** sunt utilizate pentru a genera coordonatele pe axe X și Z.

- Punctul de interes (precum, centrul scenei) rămâne fix.
- **Rezultatul mișării:** Camera oferă o vedere de ansamblu asupra scenei printr-o rotație lină.

3.3 Metode de iluminare

3.3.1 Lumina ambientală

Este o lumină difuză, omniprezentă, care vine din toate direcțiile și iluminează toate suprafețele uniform.

Caracteristici:

- **Uniformitate:** Lumina ambientală este considerată omniprezentă și uniform distribuită în întreaga scenă. Nu depinde de poziția sau direcția sursei de lumină.
- **Fără reflexii direcționale:** Lumina ambientală afectează obiectele într-un mod uniform, fără a genera umbre sau puncte de reflexie speculară.
- **Control simplu:** Este specificată de o singură culoare și o intensitate, aplicată uniform în toate direcțiile.

```
#version 410 core

out vec4 fColor;

void main()
{
    fColor = vec4(1.0f);
}
```

3.4 Metode de vizualizare a scenei

3.4.1 Vizualizare Solid

În acest mod, obiectele sunt afișate complet solide, cu fațetele lor pline. Este cel mai utilizat mod pentru a reprezenta obiectele într-un mod realist sau pentru a aplica efecte de iluminare și texturare.

```
if (pressedKeys[GLFW_KEY_J]) { //SOLID
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
}
```

3.4.2 Vizualizare Smooth

În acest mod, obiectele sunt afișate cu o tranziție lină între culori sau lumini, utilizând interpolarea normalelor. Este folosit pentru a crea un efect mai realist asupra suprafețelor curbe. Este vizualizarea inițială a scenei.

3.4.3 Vizualizare Wireframe

În acest mod, obiectele sunt afișate doar prin muchiile lor, fără a umple fațetele. Este util pentru a analiza structura geometrică a obiectului.

```
if (pressedKeys[GLFW_KEY_K]) { //WIREFRAME  
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);  
}
```



Figura 4: Vizualizare WireFrame

3.4.4 Vizualizare Poligonal

În acest mod, se afișează poligoanele care formează suprafața obiectului, fără a adăuga efecte de iluminare sau netezire. Este util pentru a înțelege structura poligonală brută.

```
if (pressedKeys[GLFW_KEY_L]) { //POLIGONAL  
    glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);  
}
```



Figura 5: Vizualizare Poligonal

4 Realizarea șcenei

4.1 Fotorealism

Fotorealismul în OpenGL se referă la crearea de imagini 3D care sunt extrem de apropiate de realitate din punct de vedere al aspectului vizual, iluminării și texturii. În OpenGL, fotorealismul este atins printr-o combinație de tehnici avansate de iluminare, texturare, și procesare a imaginii. Aceasta include simularea comportamentului luminii, reflexiilor, umbrei și detaliilor de suprafață pentru a obține o imagine cât mai realistă posibil.

Câteva dintre tehniciile care ajută la crearea fotorealismului sunt:

- Texturarea avansată: folosită pentru a adăuga detalii suplimentare pe suprafețele obiectelor.
- Iluminarea și materialele: surse de lumină simple, cum ar fi lumina direcțională sau punctiformă.
- Simularea atmosferei: ex.: fog (ceață), care este o tehnică utilizată pentru a adăuga adâncime scenei sau utilizarea unor Skyboxes.

Proiectul prezentat conține unele dintre aceste tehnici, iar obiectele utilizate se aproprie de realism destul de mult (un exemplu ar putea fi prezența obiectelor care înfățișează oameni).

Referitor la partea de umbre, s-a utilizat tehnica Shadow mapping. Shadow mapping este o tehnică de generare a umbrelor care presupune utilizarea unui "shadow map", care este o

textură ce stochează informații despre distanțele față de sursa de lumină. Aceasta este folosită ulterior pentru a determina dacă un pixel este iluminat sau nu de la sursa de lumină.

Crearea unei Shadow Map

1. Crearea texturii pentru distanțele față de sursa de lumină

Pentru a genera o *shadow map*, se creează o textură care va contine distanțele față de sursa de lumină pentru fiecare fragment din scenă:

- În acest pas, scena este randată din perspectiva sursei de lumină (nu din perspectiva camerei).
- În loc de a salva culoarea fiecărui pixel, se salvează doar distanța față de sursa de lumină.

2. Randarea scenei din perspectiva sursei de lumină

- Se creează o matrice de proiecție a sursei de lumină și o matrice de vizualizare.
- Această matrice permite randarea obiectelor din perspectiva sursei de lumină, astfel încât distanțele față de sursa de lumină să fie captureate într-o textură (*shadow map*).

3. Verificarea umbrelor în etapa de iluminare

După ce *shadow map*-ul a fost creat:

1. Scena este randată din perspectiva camerei.
2. Pentru fiecare fragment, se verifică dacă acesta este în umbră sau nu:
 - Comparând distanța fragmentului față de sursa de lumină cu valoarea stocată în *shadow map*.
 - Dacă distanța fragmentului este mai mică decât distanța stocată în *shadow map*, înseamnă că fragmentul este iluminat.
 - Altfel, fragmentul este în umbră.



Figura 6: Prezentarea fotorealismului

4.2 Ceată

Ceața conferă o anumită impresie de opacitate șenei și creează astfel iluzia de profunzime, ajutând la îmbunătățirea realismului. Acest efect ascunde obiectele aflate la distanțe mari astfel încât acestea să nu dispară într-un mod nerealist.

Pentru realizarea acestui efect, am folosit formula pentru ceată exponențială pătratică din laborator:

$$\text{fogFactor} = e^{-(\text{fragmentDistance} \times \text{fogDensity})^2}$$

unde:

- **fogFactor** este factorul de ceată calculat,
- **fragmentDistance** este distanța față de camera 3D,
- **fogDensity** reprezintă densitatea ceații, un parametru care controlează intensitatea acestui efect.

Crearea efectului de ceată în OpenGL se face, de obicei, printr-un proces care modifică culoarea pixelilor pe măsură ce aceștia devin mai distanți de cameră. Când OpenGL redă fiecare fragment, folosește formula de mai sus pentru a modifica culoarea fragmentului în funcție de distanța față de cameră. Acest lucru este realizat în shader-ul fragmentului.



Figura 7: Prezentarea efectului de ceață

4.3 SkyBox

Pentru a afișa o imagine de fundal, s-au inclus în scenă două skyBox-uri, mai precis două cubemap-uri având câte 6 texture diferite pe fiecare față. Pentru a putea fi adăugate în scenă a fost nevoie de un shader special `skyboxShader.frag`. Funcțiile din `main.cpp`: `initSkyBox()`, respectiv `initSkyBox()` sunt cele care fixează skybox-ul în scenă în funcție de modul dorit. Pentru fiecare, sunt încărcate cele 6 fețe ale cubului, care sunt asignate fețelor corespunzătoare ale skyBox-ului.

În `main.cpp` pe baza codului de mai jos se decide dacă scenă se află în modul de zi sau în modul de noapte. Trecerea se face pe baza tastelor N - modul zi, respectiv M - modul noapte.

```
if (!isDay) {  
    initSkyBox(); }  
else {  
    initSkyBoxDark(); }
```

S-a ales implementarea unui skyBox pentru că oferă posibilitatea de a crea un cer realist într-un mediu 3D și pentru că oferă o performanță mult mai bună decât alternativa de a rasteriza un cer 3D, care poate fi destul de costisitoare.

4.4 Sonorizare

Sunetele pot fi plasate în lumea 3D, iar poziția lor afectează modul în care sunt auzite de utilizator, similar cu modul în care obiectele grafice sunt plasate în OpenGL. Metoda `PlaySoundA` este o funcție disponibilă în API-ul Windows, utilizată pentru a reda fișiere audio. Aceasta face parte din bibliotecile Windows API și este folosită pentru a reda sunete în aplicații Windows.

Parametrii funcției sunt următorii:

- `lpzSound`: reprezintă calea fișierului audio sau numele unui sunet predefinit în sistem.
- `hmod`: este un handle pentru un modul ce conține resursa audio.
- `fdwSound`: reprezintă tipul de redare al sunetului (sincron sau asincron).

```
BOOL PlaySoundA(
    LPCSTR     lpzSound ,
    HMODULE    hmod ,
    DWORD      fdwSound
);
```

Astfel că s-au introdus două sunete în proiect: unul general, care este auzit odată cu pornirea proiectului (aici s-a ales un sunet de păsărele pentru ambianța scenei) și unul care este pornit/oprit pe baza unor taste: 0 sau 9.

```
if (pressedKeys[GLFW_KEY_0]) {
    PlaySoundA("miaw.wav", nullptr, SND_ASYNC | SND_FILENAME);
}

if (pressedKeys[GLFW_KEY_9]) {
    PlaySound(nullptr, nullptr, SND_ASYNC);
}
```

5 Animații

Există două animații în scenă, atât pe obiect cât și pe grupuri de obiecte. S-a realizat animația cu cele două păsări, care realizează mișcarea de rotație într-un punct fix. Cordonatele punctului fix sunt:

$$x = -6.1501, \quad y = -1.9767, \quad z = 2.3055$$

Pentru a se realiza această mișcare de rotație, obiectul s-a traslatat în punctul fix (cel în jurul căreia se dorește mișcarea de rotație), s-a realizat o rotație cu valorile dorite și s-a transltat înapoi în punctul de coordonate a obiectului realizat in Blender. Acest lucru are următoare implementare în OpenGL:

```
birdModel = glm::translate(birdModel, spinAroundPoint);

birdModel = glm::rotate(birdModel, glm::radians(angle), glm
    ::vec3(0.0f, 1.0f, 0.0f));
```

```

birdModel = glm::translate(birdModel, birdInitialPosition -
    spinAroundPoint);

```

Totodată în implementare se găsesc și metode corespunzătoare shadere-lor, precum **shader.useShaderProgram()**, care folosește un program de shader specific pentru procesul de randare al obiectului. Programul de shader este responsabil pentru procesarea vertex-urilor și fragmentelor într-un mod specific (de exemplu, aplicând iluminare, texturare, etc.). Funcția **bird.Draw(shader)** este definită pentru a desena obiectul "bird" folosind programul de shader activizat anterior. Funcția Draw() va folosi transformările și uniformele setate anterior și va desena obiectul utilizând buffer-ele de geometrie și shader-ul specificat. Aceasta este utilizată și pentru a doua animație: cea cu mașina. Pentru ambele obiecte asupra căror se crează animațiile, după ce acestea sunt desenate, matricea de transformare este resetată și skybox-ul este desenat folosind un shader special, cu adâncimea dezactivată pentru a evita ca obiectele să fie acoperite de obiectele din față.

În ceea ce privește animația cu mașine, partea de shadere se păstrează. Pentru ca mașina să aibă efectul de deplasare, se incrementează direcția (se modifică punctul în care se stiuzează obiectul) pe care o urmează mașina și se translatează obiectul în acel punct modificat treptat.

```

static glm::vec3 carPosition = glm::vec3(0.13764f, -0.10738f,
    36.677f);
const glm::vec3 forwardDirection = glm::vec3(0.0f, 0.0f, 0.1f);

if (startCar) {
    carPosition += forwardDirection;
}

model = glm::mat4(1.0f);
model = glm::translate(model, carPosition);

```

Coordonatele punctului care determină poziția mașinii sunt date de vectorul:

$$\text{carPosition} = (0.13764, -0.10738, 36.677)$$

Coordonata Z a vectorului este incrementată treptat cu 1.

În OpenGL, sistemul de coordonate este diferit de cel folosit în mod obișnuit în matematică sau grafică 2D. În OpenGL, axele sunt orientate astfel:

- **X**: Se află de obicei pe orizontală (stânga-dreapta).
- **Y**: Se află pe verticală (sus-jos).
- **Z**: Se află pe adâncime (în față și în spate), ceea ce înseamnă că axa Z este folosită pentru a reprezenta distanța în față și în spatele planului de vizualizare.

6 Manual de utilizare

Mai jos este un tabel care ilustrează manualul de utilizare a testare:

Tastă	Funcționalitate
W	Camera se deplasează în față
A	Camera se deplasează la stânga.
S	Camera se deplasează în spate.
D	Camera se deplasează la dreapta.
Q	Camera se rotește la stânga.
E	Camera se rotește la dreapta.
J	Vizualizare Solid.
K	Vizualizare Wireframe.
L	Vizualizare Poligonal.
N	Skibox de zi.
M	Skibox de noapte.
F	Activarea efectului de ceată.
G	Dezactivarea efectului de ceată.
H	Animație mașina.
Z	Activare preview scenă.
X	Dezactivare preview scenă.
0	Start sunet special.
9	Stop sunet special.

Tabela 1: Taste și Funcționalități

7 Concluzii

Fiind primul contact cu un proiect realizat în OpenGL, dificultatea de implementare a fost mai ridicată. Astfel că îmbunătățiri ulterioare pot exista mai ales asupra implementării codului. Unele dintre realizări sunt: înțelegerea fundamentelor graficii computaționale și a pipeline-ului de randare.

În ceea ce privește experiența de învățare, acest proiect a oferit oportunitatea de a dobândi cunoștințe valoroase în următoarele domenii:

- Înțelegerea profundă a pipeline-ului grafic OpenGL și a etapelor sale principale: vertex

processing, primitive assembly, rasterization și fragment processing.

- b) Înțelegerea transformări geometrice și lucrul cu matrici de transformare.
- c) Sisteme de coordonate și conversia între acestea.
- d) Familiarizarea cu scrierea și optimizarea shader-elor (vertex și fragment shaders).
- e) Manipularea texturilor și coordonatelor UV.
- f) Implementarea tehniciilor de camera și control al perspectivei.
- g) Gestionarea resurselor grafice și optimizarea performanței.

Printre direcțiile potențiale de dezvoltare se numără:

1. Implementarea sistemului de animație pentru obiectele din scenă, permitând mișcări fluide și interacțiuni dinamice între elementele mediului virtual.
2. Optimizarea sistemului de iluminare prin introducerea surselor de tip spotlight, care ar simula mai realist lumina emisă de felinare și ar crea efecte atmosferice mai convingătoare. Lumina punctiformă ar fi o îmbunătățire absolut necesară.
3. Implementarea unui sistem de particule pentru efecte atmosferice, cum ar fi ploaia, zăpada, care ar adăuga un plus de realism și dinamism scenei.
4. Dezvoltarea unui sistem complex de umbre rasterizate, care ar îmbunătăți semnificativ realismul scenei prin adăugarea umbrelor dinamice pentru toate obiectele.
5. Integrarea unui sistem robust de detectare a coliziunilor, care ar permite interacțiuni mai realiste între obiectele din scenă și ar preveni intersectările nefișești dintre acestea. (ex.: coliziunea cu clădirile - să nu mai existe posibilitatea de a intra într-o clădire).

Bibliografie

- [1] *Prelucrare Grafică - suport curs și laborator.* Disponibil la: <https://moodle.cs.utcluj.ro>.
- [2] *Laborator 9 - utilizat ca tamplate.* Disponibil la: <https://moodle.cs.utcluj.ro>.
- [3] <https://free3d.com>.
- [4] <https://www.turbosquid.com/Search/3D-Models/free>.
- [5] <https://www.blender.org/support/>.
- [6] <https://medium.com>.
- [7] <https://www.humus.name/index.php?page=Textures&start=96>.