UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Scena OpenGL

Proiectare Grafica

Autor: Paucean Ioana
Grupa: 30231

Facultatea de Automatica
si Calculatoare

22 Ianuarie 2026

Cuprins

1 Prezentarea temei

Scopul acestui proiect este realizarea unei aplicații grafice tridimensionale interactive, utilizând biblioteca OpenGL și conceptele prezentate în cadrul lucrărilor de laborator. Aplicația permite randarea unei scene 3D complexe și oferă utilizatorului posibilitatea de a explora mediul virtual prin intermediul camerei, folosind dispozitive de input standard, precum tastatura și mausul.

Pe lângă controlul manual al camerei, proiectul include și o animație de prezentare automată a scenei. Aceasta constă într-o deplasare lină a camerei printr-o serie de puncte prestabilite (waypoint-uri), având rolul de a evidenția structura și organizarea spațiului tridimensional.

Scena este iluminată utilizând mai multe surse de lumină, iar obiectele sunt texturate pentru a obține un grad mai ridicat de realism vizual. Implementarea urmărește respectarea principiilor pipeline-ului programabil OpenGL și utilizarea transformărilor de modelare, vizualizare și proiecție, precum și a shader-elor pentru calculul iluminării și randării finale.

2 Scenariul

2.1 Descrierea scenei și a obiectelor

Scena 3D este alcătuită dintr-un ansamblu de obiecte statice, importate din fișiere externe și poziționate într-un sistem de coordonate comun. Aceste obiecte formează un mediu coerent, care poate fi explorat din mai multe perspective. Modelele sunt scalate și poziționate astfel încât să păstreze proporții realiste și să ofere puncte de interes vizual pe parcursul explorării.

Fiecare obiect este asociat cu materiale și texturi specifice, care definesc aspectul suprafețelor în condiții diferite de iluminare. Normalele obiectelor sunt utilizate în calculele de iluminare pentru a evidenția volumul și forma acestora. Dispunerea elementelor din scenă este realizată astfel încât să fie compatibilă atât cu explorarea liberă, cât și cu animația automată a camerei.

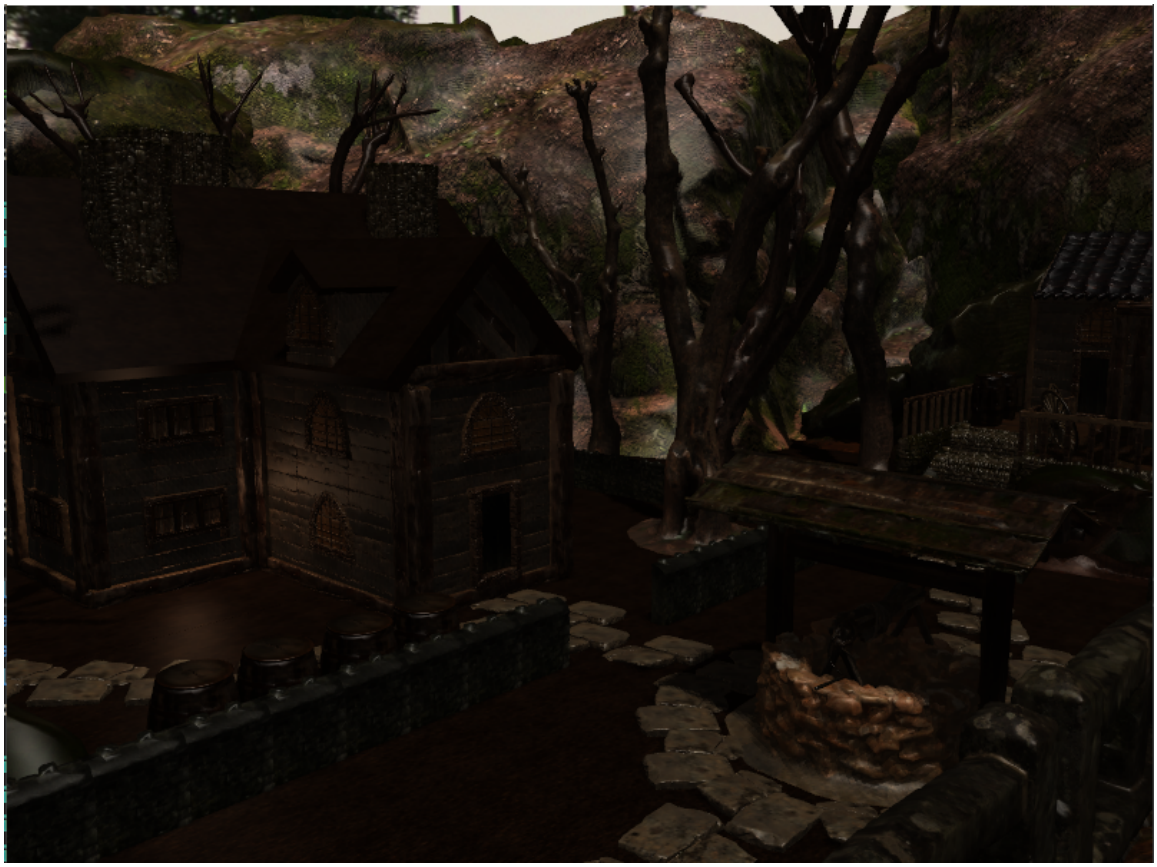

Figura 1: Vedere generală a scenei randate în OpenGL

2.2 Funcționalități

Aplicația oferă următoarele funcționalități principale:

- deplasarea camerei în scenă pe cele trei axe principale;
- rotirea camerei în jurul axelor orizontală și verticală folosind mausul (yaw/pitch);
- modificarea câmpului vizual prin funcția de zoom (FOV);
- animație automată de prezentare a scenei, cu tranziții line între puncte (tour);
- iluminare realizată cu mai multe surse de lumină (direcțională și punctuală);
- aplicarea de texturi și materiale asupra obiectelor;
- randarea umbrelor pentru creșterea realismului vizual;
- randare skybox (cubemap) pentru fundal.

3 Detalii de implementare

3.1 Funcții și algoritmi

3.1.1 Soluții posibile

Pentru implementarea mișcării camerei și a animației de prezentare au fost analizate mai multe variante. O abordare simplă constă în schimbarea instantanee a poziției camerei între puncte prestabilite, însă aceasta conduce la mișcări bruște și neplăcute vizual.

O altă soluție este interpolarea poziției camerei între două puncte consecutive, utilizând o funcție dependentă de timp. Această metodă permite obținerea unei mișcări fluide și uniforme, indiferent de numărul de cadre randate pe secundă.

3.1.2 Motivarea abordării alese

Abordarea implementată se bazează pe interpolarea poziției camerei în funcție de timp, folosind diferența dintre cadre succesive (`deltaTime`). Această soluție oferă o experiență vizuală naturală și permite controlul precis al vitezei de deplasare. De asemenea, structura aleasă permite extinderea facilă a traseului camerei prin adăugarea de noi puncte în vectorul de poziții.

În ceea ce privește iluminarea, utilizarea mai multor surse de lumină contribuie la o mai bună percepție a adâncimii scenei și la evidențierea detaliilor geometrice ale obiectelor.

3.1.3 Algoritmi implementați efectiv (din cod)

1) Control cameră (WASD + mouse look) Camera este controlată cu tastatura și mouse-ul:

- **Deplasare** pe direcțiile forward/back/left/right, calculată din vectorii camerei. Pentru mișcare naturală, înainte/înapoi și stânga/dreapta sunt proiectate pe planul XZ (se ignoră componenta Y), iar urcarea/coborârea se face separat pe axa Y (SPACE / SHIFT).
- **Rotire** cu mouse-ul prin actualizarea incrementală a unghiurilor `yaw` și `pitch`. Pitch este limitat (clamped) în intervalul aproximativ $[-89°, 89°]$ pentru a evita inversarea camerei.

Figura 2: Explorarea scenei: cameră controlată manual (WASD + mouse)

2) Zoom (scroll) prin modificarea FOV Zoom-ul este implementat prin modificarea câmpului vizual (`Camera::Zoom`), actualizat la scroll și limitat între valori rezonabile (1..90 grade). Această abordare păstrează o tranziție rapidă și intuitivă pentru utilizator, fără a modifica poziția camerei.

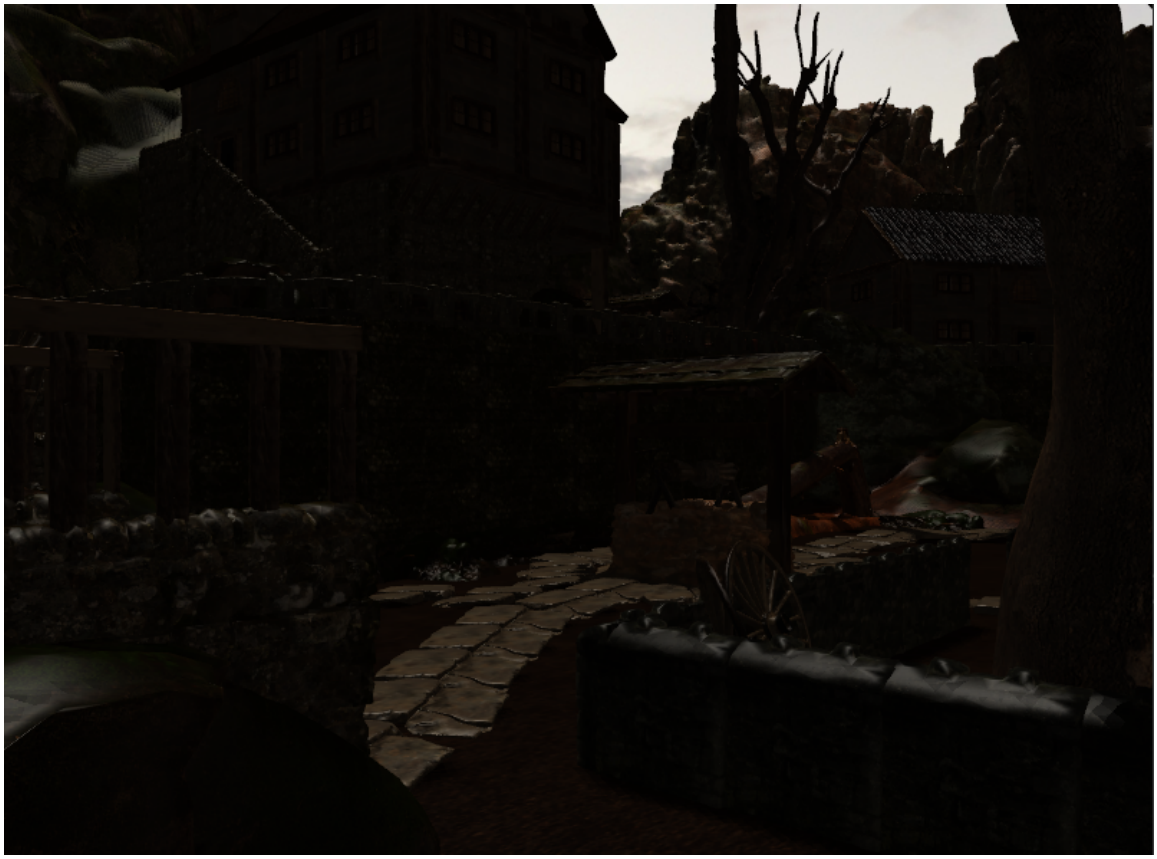

Figura 3: Modificarea câmpului vizual (FOV) prin scroll

3) Animația automată a camerei (tasta P) – tour cu waypoint-uri Pentru prezentarea automată, există o listă de waypoint-uri, fiecare având: `pos` (poziția camerei), `lookAt` (ținta) și `seconds` (durata până la următorul punct).

Mișcarea se face între waypoint-ul curent și următorul astfel:

- se acumulează progresul `tourT` cu `deltaTime / duration`;
- se aplică un easing de tip **smoothstep** pentru accelerație/decelerație lină:

$$s(t) = t^2(3 - 2t), \quad t \in [0, 1]$$

- se interpolează liniar (`glm::mix`) poziția și ținta:

$$pos = mix(pos_i, pos_{i+1}, s), \quad look = mix(look_i, look_{i+1}, s)$$

- camera este actualizată cu noua poziție, țintă și direcție front (look - pos).

Această metodă produce o animație **stabilă în timp** (frame-rate independentă) și ușor de extins prin adăugarea de noi puncte în vector.

4) Iluminare: lumină direcțională + lumină punctiformă Sunt folosite două tipuri de lumină:

- **Lumină direcțională** (simulează „soarele”), definită printr-un vector direcție și o culoare.
- **Lumină punctiformă**, definită prin poziție și culoare (de tip „lampă/torță” în scenă).

Pentru a evidenția vizual poziția luminii punctiforme, proiectul desenează un „lamp cube” (un cub mic randat cu un shader simplu care folosește `lampColor`).

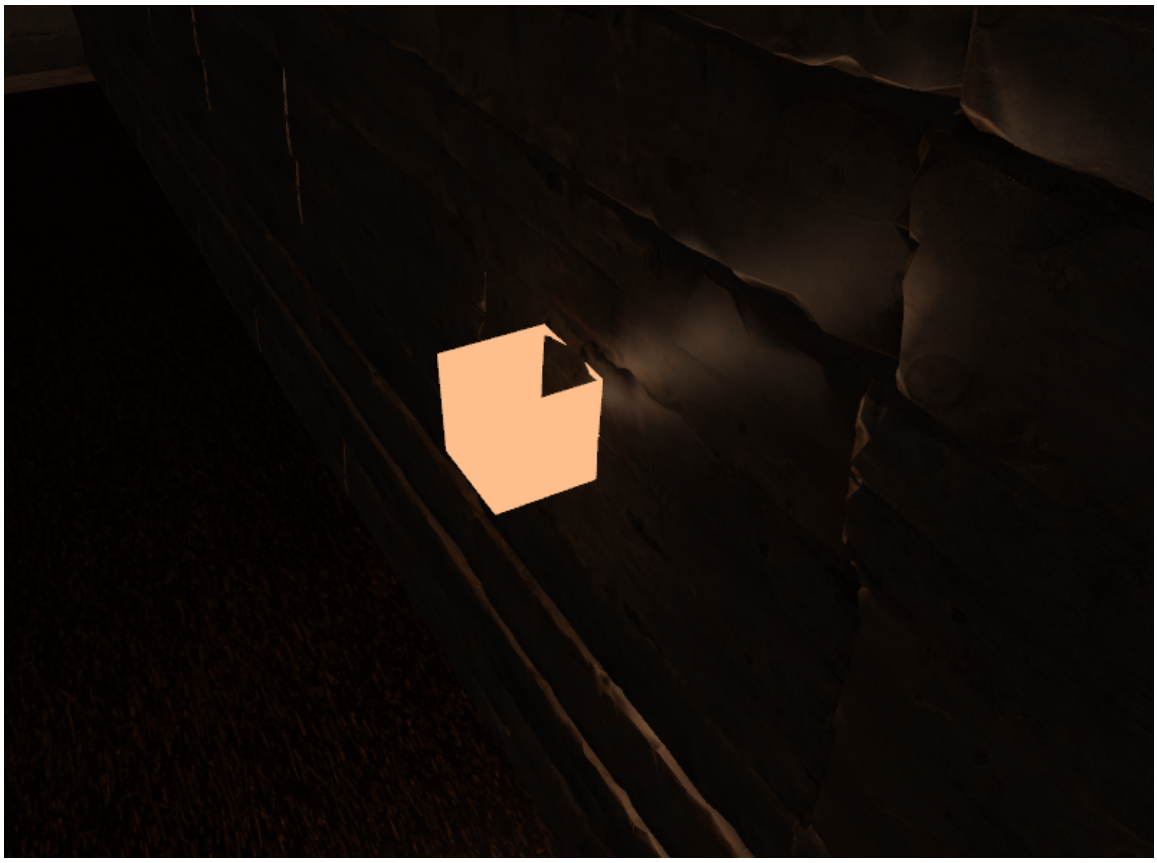

Figura 4: Iluminare în scenă: lumină direcțională + lumină punctiformă (marker vizual)

5) Rotirea luminii direcționale (tasta R) Tasta **R** activează/dezactivează rotația luminii direcționale (toggle). Cât timp rotația este activă, unghiul crește cu `sunSpeed * deltaTime`, iar direcția luminii este rotită în jurul axei Y cu o matrice de rotație. Rezultatul este o modificare graduală a direcției luminii, care schimbă dinamica umbrelor și iluminarea scenei.

6) Umbre (shadow mapping) – depth pass + main pass Umbrele sunt implementate printr-o tehnică standard de **shadow mapping**:

- **Depth pass**: scena este randată din perspectiva sursei de lumină într-un framebuffer (`depthMapFBO`) care conține o **depth texture** (`depthMapTex`). Vertex shader-ul folosește `lightSpaceMatrix` și `model` pentru a proiecta vertecșii.
- **Main pass**: scena este randată normal din perspectiva camerei, iar shader-ul primește `shadowMap` + `lightSpaceMatrix`. Fiecare fragment își calculează coordonata în spațiul luminii și compară adâncimea cu valoarea din depth map pentru a decide dacă este în umbră.

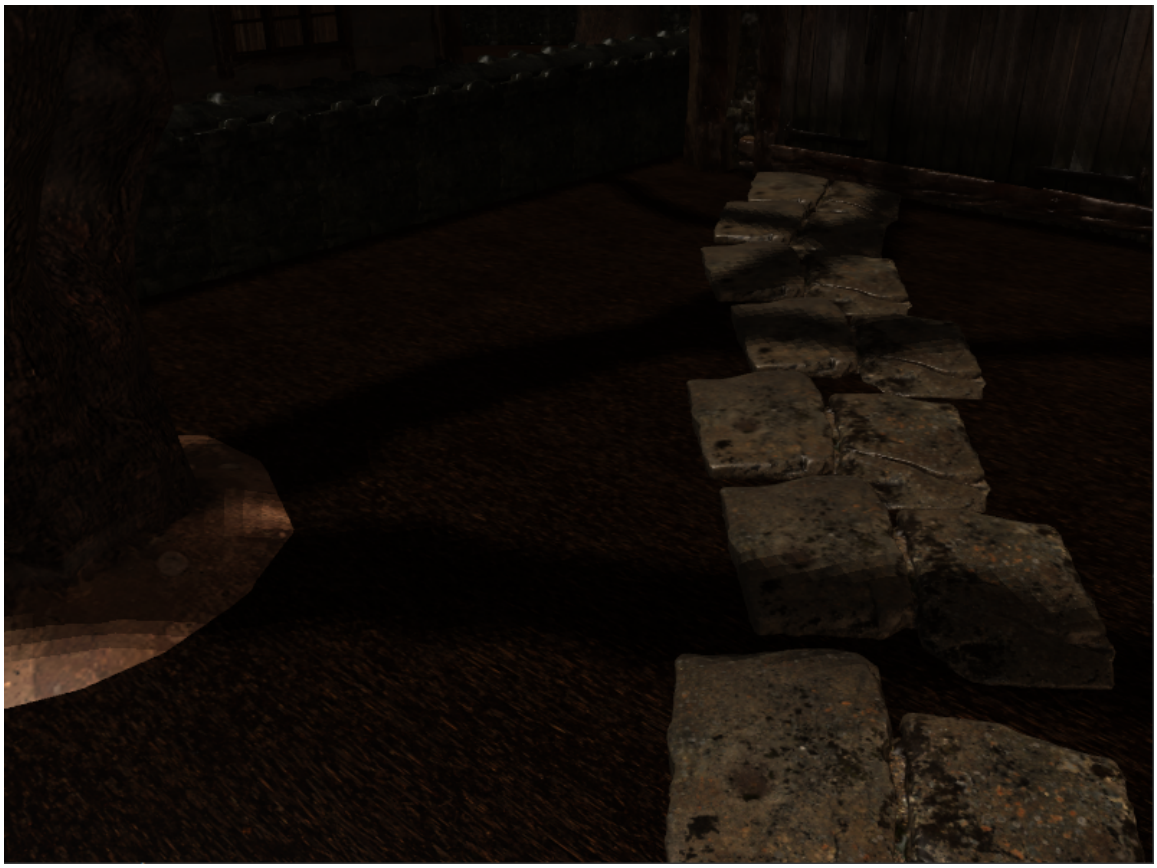

Figura 5: Umbre în scenă (shadow mapping) – efect vizual de profunzime

7) Texturare și materiale Texturile sunt aplicate obiectelor din scenă prin binding pe unitatea de textură (de exemplu `GL_TEXTURE0` pentru textura difuză). În shader-ul principal se folosesc coordonatele UV și normalele pentru a obține un rezultat realist: suprafețele își schimbă aspectul în funcție de lumină și orientare.

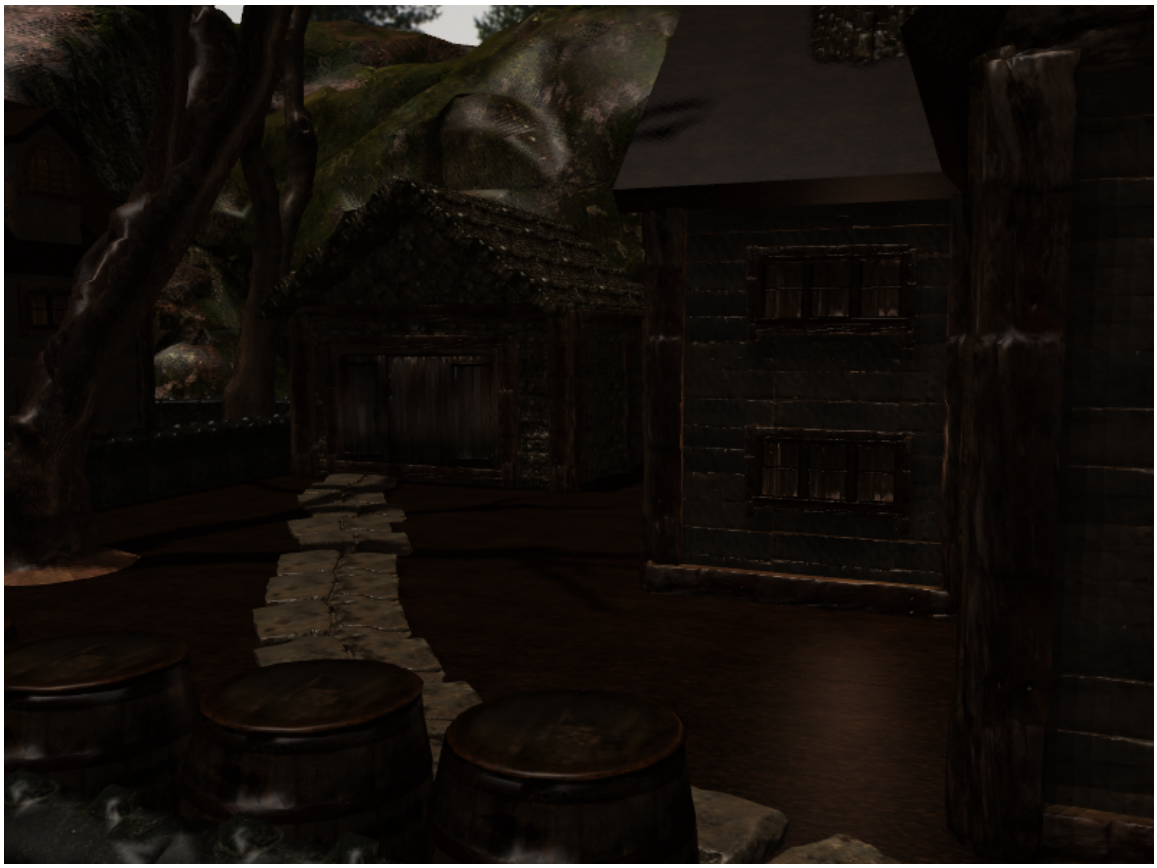

Figura 6: Texturarea obiectelor și evidențierea materialelor

8) Skybox (cubemap) randat la final Skybox-ul este randat la finalul frame-ului, cu `glDepthFunc(GL_LEQUAL)` și fără a scrie în depth buffer (`glDepthMask(GL_FALSE)`), pentru a evita acoperirea obiectelor. Matricea `view` este „tăiată" la partea de rotație (se elimină translația), astfel încât skybox-ul să pară infinit și stabil în jurul camerei.

3.2 Modelul grafic

Aplicația urmează modelul clasic de randare OpenGL, bazat pe transformările de **model**, **view** și **proiecție**. Aceste transformări sunt aplicate în vertex shader, iar calculul iluminării este realizat în shader-ele grafice pe baza parametrilor transmiși din aplicația principală.

Bucla de randare este organizată în pași clari:

1. calcularea matricelor `projection` și `view` din cameră;
2. calcularea `lightSpaceMatrix` pentru pass-ul de umbre;
3. **depth pass** către `depthMapFBO`;
4. **main pass** cu iluminare + texturi + umbrire;
5. randarea marker-ului luminii (cubul mic);
6. randarea skybox-ului.

3.3 Structuri de date

Pentru implementare sunt utilizate următoarele structuri de date:

- vectori de tip `glm::vec3` pentru pozițiile camerei, direcții și waypoint-uri;
- variabile pentru orientarea camerei (unghiuri yaw și pitch) și sensibilitate mouse;

- structură `Waypoint` (poziție, lookAt, timp) și un vector `cameraPath`;
- parametri pentru lumini: direcțională (direcție/culoare) și punctuală (poziție/culoare);
- resurse OpenGL: FBO/texture pentru shadow map, VAO/VBO pentru markerul luminii, cubemap pentru skybox;
- variabile de timp: `deltaTime`, `lastFrame` pentru mișcări independente de FPS.

3.4 Ierarhia de clase

Structura aplicației este modulară, separând logica de randare de cea de control al camerei și de gestionarea shader-elor. În mod concret:

- `gps::Camera`: calculează matricea `View`, gestionează mișcare, rotație și zoom;
- `gps::Shader`: compilează/folosește shader-ele (principal, depth pentru umbre, lamp pentru marker);
- `gps::Model3D`: încarcă și desenează modelul scenei (ex: `MedievalCity.obj`);

Această organizare contribuie la creșterea lizibilității codului și facilitează întreținerea și extinderea proiectului.

4 Prezentarea interfeței grafice utilizator / Manual de utilizare

Interacțiunea cu aplicația se realizează exclusiv prin tastatură și maus, după cum urmează:

- **W, A, S, D** – deplasarea camerei în scenă;
- **SPACE, LEFT SHIFT** – urcare/coborâre pe axa Y;
- **Mișcarea mausului** – rotirea camerei (yaw/pitch);
- **Scroll maus** – zoom (modificare FOV);
- **P** – activarea animației automate a camerei;
- **R** – rotirea (toggle) luminii direcționale.

5 Concluzii și dezvoltări ulterioare

Proiectul demonstrează utilizarea conceptelor fundamentale de grafică 3D și OpenGL pentru realizarea unei scene interactive. Implementarea camerei (mișcare + rotație + zoom), a animației automate, a iluminării (direcțională + punctuală), a texturilor și a umbrelor contribuie la obținerea unui rezultat coerent și realist din punct de vedere vizual.

Ca direcții de dezvoltare ulterioară, aplicația poate fi extinsă prin:

- adăugarea de materiale PBR (roughness/metallic/normal maps) pentru realism sporit;
- îmbunătățirea umbrelor (PCF, bias adaptiv, cascaded shadow maps);
- efecte de post-procesare (gamma correction, tone mapping, bloom);
- optimizări (frustum culling, LOD) pentru performanță mai bună;
- animații suplimentare și interacțiuni cu obiectele din scenă.

6 Referințe

- Khronos Group, OpenGL Documentation
- Joey de Vries, LearnOpenGL