Documentație Proiect OpenGL

January 14, 2025

Contents

1 Specificația subiectului

Proiectul constă în realizarea unei scene OpenGL care include funcționalități avansate precum navigarea camerei, tururi automate, animații pentru obiecte și efecte grafice. Fiecare cerință specificată este abordată în detaliu, utilizând modele texturate, iluminare avansată și interacțiuni complexe.

2 Scenariu

2.1 Descrierea scenei și a obiectelor

Scena proiectată include următoarele obiecte:

- **Casă (building)**: Reprezentând punctul central al scenei, cu detalii texturate.
- **Mașină (car)**: Un obiect animat care se deplasează pe axa Z.
- **UFO (drona)**: Un obiect animat care zboară circular deasupra scenei.
- **Ceainic (teapot)**: Un obiect animat care cade în mod realist pe sol.
- **Sol (ground)**: Solul scenei, texturat corespunzător.

Imagini ale obiectelor:

Figure 1: Exemplu de casă.

Figure 2: Exemplu de mașină.

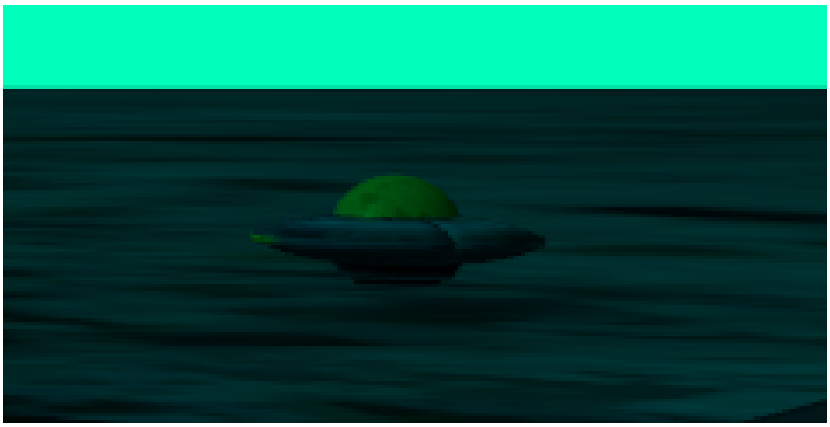

Figure 3: Exemplu de dronă(mini ufo).

2.2 Funcționalități implementate

Funcționalitățile proiectului includ:

- Navigare liberă a camerei.
- Tur automatizat al scenei.
- Iluminare direcțională și punctuală.
- Animații pentru obiecte.

- Moduri de afișare diferite (wireframe, puncte).
- Efect de ceață.

3 Detalii de implementare

Această secțiune descrie în detaliu fiecare funcționalitate implementată.

3.1 Navigarea camerei

Navigarea camerei este gestionată prin funcția `processMovement()` și permite utilizatorului să se deplaseze în scenă folosind tastele `W`, `A`, `S`, `D`, `R`, `F`. Mișcările sunt implementate astfel:

Listing 1: Navigare liberă a camerei

```
if (pressedKeys[GLFW_KEY_W]) {
    myCamera.move(gps::MOVE_FORWARD, cameraSpeed);
}
if (pressedKeys[GLFW_KEY_S]) {
    myCamera.move(gps::MOVE_BACKWARD, cameraSpeed);
}
if (pressedKeys[GLFW_KEY_A]) {
    myCamera.move(gps::MOVE_LEFT, cameraSpeed);
}
if (pressedKeys[GLFW_KEY_D]) {
    myCamera.move(gps::MOVE_RIGHT, cameraSpeed);
}
if (pressedKeys[GLFW_KEY_R]) {
    myCamera.move(gps::MOVE_UP, cameraSpeed);
}
if (pressedKeys[GLFW_KEY_F]) {
    myCamera.move(gps::MOVE_DOWN, cameraSpeed);
}
```

3.2 Turul automatizat al scenei

Turul automatizat permite utilizatorului să vizualizeze scena printr-o serie de puncte predefinite. Camera se deplasează între aceste puncte folosind interpolare liniară, astfel:

Listing 2: Interpolare pentru turul camerei

```
glm::vec3 interpolatedPosition = glm::mix(
    presentationCameraPositions[currentIndex],
    presentationCameraPositions[nextIndex],
```

```
    interpolationFactor
);
glm::vec3 interpolatedTarget = glm::mix(
    presentationCameraTargets[currentIndex],
    presentationCameraTargets[nextIndex],
    interpolationFactor
);
myCamera.setCameraPosition(interpolatedPosition);
myCamera.setCameraTarget(interpolatedTarget);
```

Pentru turul camerei, am implementat o interpolare liniară între punctele predefinite din scenă. Interpolarea este realizată folosind metoda `glm::mix`, care calculează o poziție intermediară între două puncte în funcție de un parametru temporal. Camera se deplasează între aceste puncte, iar ținta este de asemenea interpolată, pentru a crea o tranziție fluidă.

Această metodă a fost aleasă pentru că:

- Este flexibilă și extensibilă – punctele de tur pot fi adăugate sau ajustate cu ușurință.
- Creează o mișcare fluidă, ceea ce este ideal pentru un tur automatizat.

Imagini din timpul turului automatizat:

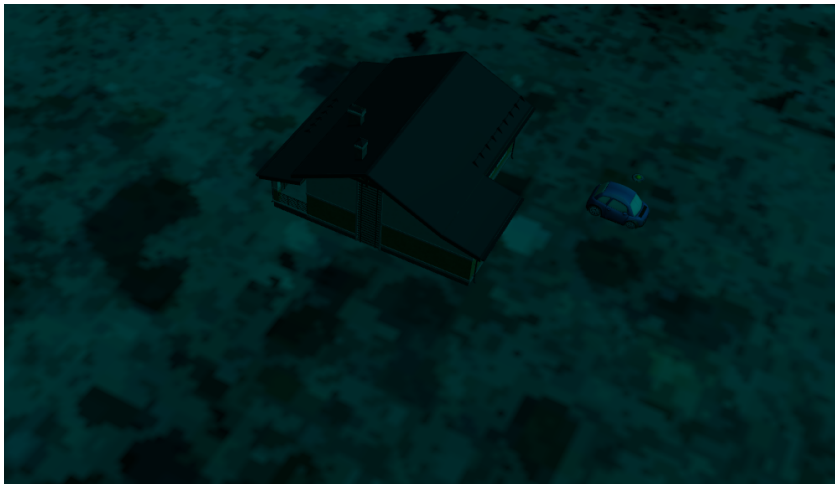

Figure 4: Punctul 1 al turului.

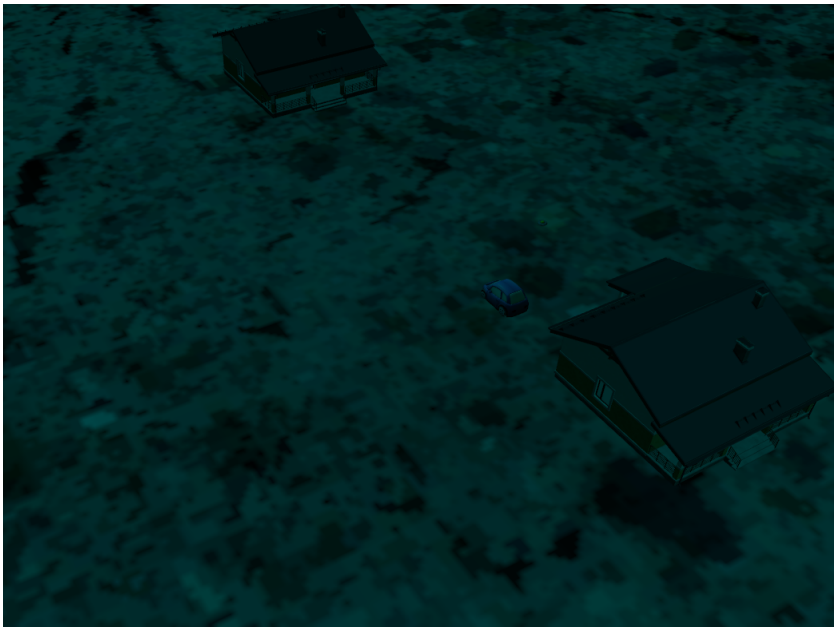

Figure 5: Punctul 2 al turului.

3.3 Iluminarea direcțională și punctuală

Iluminarea este implementată folosind două surse de lumină: direcțională și punctuală. Codul shader pentru acestea este următorul:

Listing 3: Iluminare direcțională și punctuală

```
vec3 lightDirN = normalize(lightDir);
float diff = max(dot(normal, lightDirN), 0.0);
diffuse = diff * lightColor;

vec3 toLight = normalize(pointLightPosition - fragPos);
float distance = length(pointLightPosition - fragPos);
float attenuation = 1.0 / (constantAttenuation +
    linearAttenuation * distance +
    quadraticAttenuation * (distance * distance));
diffuse += attenuation * max(dot(normal, toLight), 0.0) *
   pointLightColor;
```

Imagini cu surse de lumină activă și dezactivată:

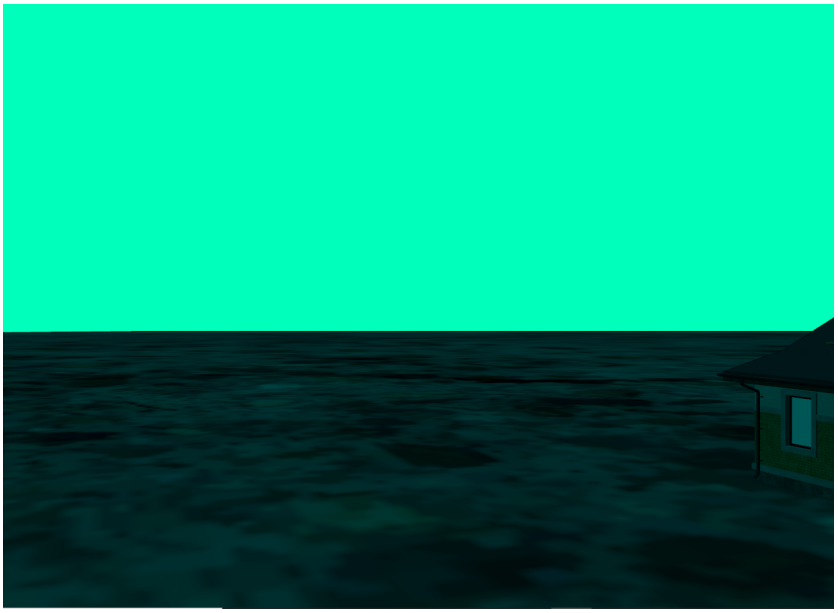

Figure 6: Iluminare direcțională activă.

Obiectul `lightCube` este folosit pentru a vizualiza sursa de lumină punctuală. Acest cub este afișat la poziția luminii și este redimensionat pentru a indica că este un punct de referință, nu un obiect funcțional în scenă. Lumina însăși este implementată folosind coeficienți de atenuare pentru a simula realismul.

Această metodă a fost aleasă pentru că:

- Vizualizarea sursei de lumină ajută la înțelegerea poziționării și a efectului acesteia asupra obiectelor din scenă.
- Cubul este ușor de implementat și oferă claritate vizuală.

Listing 4: Iluminare direcțională și punctuală

```
vec3 lightDirN = normalize(lightDir);
float diff = max(dot(normal, lightDirN), 0.0);
diffuse = diff * lightColor;

vec3 toLight = normalize(pointLightPosition - fragPos);
float distance = length(pointLightPosition - fragPos);
float attenuation = 1.0 / (constantAttenuation +
    linearAttenuation * distance +
    quadraticAttenuation * (distance * distance));
diffuse += attenuation * max(dot(normal, toLight), 0.0) *
   pointLightColor;
```

Figure 7: Iluminare punctuală activă.

3.4 Efectul de ceață

Pentru implementarea ceții, am optat pentru utilizarea shaderului de fragment, deoarece aceasta este metoda cea mai eficientă pentru a calcula efectele bazate pe distanță. Ideea principală este să ajustăm intensitatea culorii unui fragment în funcție de distanța sa față de cameră. Formula folosită se bazează pe un exponent negativ care modelează estomparea graduală, oferind un efect natural.

Această metodă a fost aleasă pentru că:

- Shaderii de fragment permit procesarea fiecărui pixel individual, ceea ce este esențial pentru un efect vizual precum ceața.
- Este flexibilă, permițând reglarea densității ceții și personalizarea culorilor.

Listing 5: Efectul de ceață

```
float fogFactor = exp(-pow(fogDensity * distance, 2.0));
fogFactor = clamp(fogFactor, 0.0, 1.0);
finalColor = mix(fogColor, baseColor, fogFactor);
```

Imagine cu ceața activată:

Figure 8: Efectul de ceață.

3.5 Animațiile obiectelor

Pentru animația dronei(care este defapt un UFO), am ales o traiectorie circulară bazată pe funcții trigonometrice, cum ar fi `sin` și `cos`. Această metodă este simplă și eficientă, permițând controlul complet asupra razei și vitezei de rotație. Coordonatele dronei sunt actualizate la fiecare cadru, folosind un unghi incremental pentru a calcula poziția sa pe cerc.

Această metodă a fost aleasă pentru că:

- Este intuitivă și ușor de controlat, fiind potrivită pentru un zbor ciclic repetabil.
- Permite modificarea simplă a traiectoriei prin ajustarea razei sau a vitezei unghiulare.

Drona: Zborul circular este realizat prin coordonate calculate cu funcții trigonometrice:

Listing 6: Mișcarea circulară a dronei

```
dronaPosition.x = radius * cos(angleDrona);
dronaPosition.z = radius * sin(angleDrona);
dronaPosition.y = 1.5f;
angleDrona += 0.01f;
```

Ceainicul: Căderea este gestionată prin decrementarea coordonatelor Y si se opreste cand atinge solul: Pentru ceainic, am implementat o animație

simplă de cădere liberă, unde poziția pe axa Y este redusă treptat până când ceainicul ajunge pe sol. Aceasta simulează gravitația într-un mod simplu și eficient.

Această metodă a fost aleasă pentru că:

Este suficient de realistă pentru o demonstrație simplă.\n- Poate fi extinsă pentru a include coliziuni mai complexe sau alte efecte.

Listing 7: Căderea ceainicului

```
if (dropTeapot) {
    teapotPosition.y -= 0.1f;
    if (teapotPosition.y <= groundLevel) {
        dropTeapot = false;
    }
}
```

Figure 9: Caderea ceainicului.

4 Prezentarea interfeței grafice / Manual de utilizare s

- **Taste W, A, S, D, R, F**: Navigare liberă a camerei.
- **Tastă T**: Pornirea turului automatizat.
- **Taste X, C, Z**: Schimbarea modurilor de afișare (wireframe, puncte, solid).
- **Tastă 1**: Activare/Dezactivare ceață.
- **Tastă 0**: Activare/Dezactivare lumină punctuală.

- **Taste N, M**: Mișcarea mașinii înainte și înapoi.

5 Concluzii și dezvoltări ulterioare

Proiectul demonstrează utilizarea OpenGL pentru crearea unei scene interactive și realiste. Direcțiile de dezvoltare includ:

- Implementarea umbrelor prin shadow mapping.
- Adăugarea efectelor de ploaie și reflexii.
- Imbunatatirea scenei si adaugarea de obiecte dinamice.
- Creerea scenei in blender si adaugarea a mai multor obiecte acolo.

Desi unele lucruri puteau fi mult mai frumos implementate, prin mai putine linii de cod si mai logic structurate, am avut ocazia sa invat mai multe despre grafica si programarea scenelor, chiar daca anumite functionalitati au fost mai dificil de implementat deoarece a fost ceva nou.

6 Referințe

- Tutoriale OpenGL: `https://learnopengl.com/`
- Documentația oficială OpenGL: `https://www.opengl.org/documentation/`
- Laboratoarele de PG
- OpenGL core project
- Resurse externe pentru modele 3D și texturi: https://www.cgtrader.com/ ; https://free3d.com/
- Tutoriale pe youtube
- ChatGPT