UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

PROIECT

PRELUCRARE GRAFICĂ

BUMBU RĂZVAN-VASILE

GRUPA 30233

AN UNIVERSITAR 2022-2023

Cuprins

1. Prezentarea temei

Proiectul a avut ca scop realizarea unei prezentări fotorealiste a unei scene de obiecte 3D utilizând librăriile prezentate la laborator (OpenGL, GLFW, GLM etc). Utilizatorul trebuie să aibă posibilitatea de a controla scena cu ajutorul tastaturii și/sau a mouse-ului.

Cerințele care au trebuit îndeplinite în cadrul proiectului sunt următoarele:

- **(2p)** vizualizarea scenei: scalare, translație, rotație, mișcarea camerei
  - utilizând tastatura sau mausul
  - utilizând animații de prezentare
- **(1p)** specificarea surselor de lumina (cel puțin două surse de lumină diferite)
- **(0.5p)** vizualizare scenă în modurile solid, wireframe, poligonal și smooth
- **(1p)** maparea texturilor și definirea materialelor
  - calitatea texturilor și nivelul de detaliu al acestora
  - maparea texturilor pe obiecte
- **(1p)** exemplificarea generării umbrelor
- **(0.5p)** exemplificarea animării diferitelor componente ale obiectelor
- **(3p)** fotorealism, complexitatea scenei, nivelul de detaliere al modelării, dezvoltarea diferiților algoritmi și implementarea acestora (generare dinamică de obiecte, detecția coliziunilor, generarea umbrelor, ceață, ploaie, vânt), calitatea animațiilor, utilizarea diferitelor surse de lumină (globală, locală, de tip spot)
- **(1p)** documentația (obligatorie)

2. Scenariul

a. Descrierea scenei și a obiectelor

Scena pe care am ales să o creez reprezintă un mic orășel, care are un lac în centrul său, câteva blocuri, o clădire de garaje, un service auto, câteva mașini, bănci și copaci.

Obiectele au fost editate cu ajutorul programului 3D Blender, după care le-am integrat în peisaj. De asemenea, obiectele au avut nevoie de o texturare, pe care am aplicat-o tot cu ajutorul programului menționat mai sus.

Ca și obiecte individuale avem următoarele:

- 6 blocuri
- Un garaj
- Un service
- 3 mașini (caroserie + roți + interior)

- Un semn de circulație (care se rotește)
- Mai multe bănci
- Mai mulți copaci (coroană + trunchi)
- Un strop de apă (pentru efectul de ploaie)

Ca și obiecte încărcate, avem doar 3 obiecte mari:

- Scena (clădiri + garaj + service + mașini + copaci + bănci)
- Semnul de circulație
- Stropul de ploaie

-Scena completă-

b. Funcționalități

Proiectul are următoarele funcționalități:

- Navigare prin interiorul scenei
- Pornirea animației de prezentare (animație + sunet)
- Pornirea efectului de ceață
- Pornirea efectului de ploaie (cu sunet)
- Pornirea efectului de spotlight
- Vizualizarea scenei în modul solid
- Vizualizarea scenei în modul wireframe

- Vizualizarea scenei în modul punctiform
- Pornirea tranziției zi-noapte
- Pornirea efectului de fulger (cu sunet)
- Rotirea semnului de circulație
- Pornirea și oprirea celor două surse de lumină globale

3. Detalii despre implementare

a. Funcții și algoritmi

i. Soluții posibile

Pentru a implementa proiectul, m-am folosit în primul rând de schița proiectului care ne-a fost pusă la dispoziție. Soluțiile posibile au fost de cele mai multe ori mai multe la număr, eu alegându-le pe cele care mi s-au părut cele mai relevante.

Funcționalitățile librăriei OpenGl au reprezentat un punct important în realizarea proiectului, întrucât adesea am fost nevoit să mă folosesc de acestea pentru generarea obiectelor, dar și pentru crearea diferitelor efecte.

Majoritatea metodelor au fost scrise în fișierul main.cpp, după cum urmează:

- Metoda **glCheckError(...)**
- Metoda **windowResizeCallback(...)**
- Metoda **keyboardCallback(…)**
- Metoda **mouseCallback(…)**
- Metoda **reset(…)**
  - Pentru readucerea camerei în poziția inițială
- Metoda **updateRainMatrix(...)**
  - Pentru crearea efectului de ploaie
- Metoda **updateCameraMatrix(...)**
  - Pentru transmiterea către shader a locației camerei și pentru calcularea noii normale
- Metoda **processMovement(...)**
  - Pentru interpretarea apăsării de taste
- Metoda **initOpenGLWindow(...)**
  - Pentru crearea ferestrei de vizualizare
- Metoda **setWindowCallBacks(...)**
- Metoda **initOpenGLState(...)**
- Metoda **initModels(...)**

  - Pentru încărcarea obiectelor
- Metoda **initShaders(...)**
  - Pentru inițializarea shadere-lor
- Metoda **initRain(...)**
  - Pentru inițializarea picurilor de ploaie
- Metoda **initUniforms(...)**
  - Pentru inițializarea obiectelor
- Metoda **renderRain(...)**
  - Pentru crearea efectului de ploaie
- Metoda **renderSceneObjects(...)**
  - Pentru crearea scenei
- Metoda **renderScene(...)**
  - Pentru afișarea scenei
- Metoda **cleanup(...)**
- Metoda **play(...)**
  - Pentru crearea animației de prezentare
- Metoda **skyBoxAnimation(...)**
  - Pentru crearea animației skyBox-ului
- Metoda **thunderAnimation(...)**
  - Pentru crearea animației fulgerului
- Metoda **main(...)**
  - Metoda principală pentru apelarea celorlalte metode

De asemenea, a fost nevoie de definirea și implementarea mișcărilor de translație și rotație pentru cameră în fișierele Camera.hpp, respectiv Camera.cpp.

Pentru crearea iluminării și a ceței, am modificat shader-ul găsit în fișierul basic.frag.

ii. Realizarea funcționalităților

- **Mișcarea**
  - Mișcarea camerei se realizează pe cele 3 axe: X, Y, Z
  - Camera are 3 componente, care sunt transmise din Main:
    - cameraPosition
    - cameraTarget
    - cameraUp
  - Pe baza acestor componente, se calculează cameraFrontDirection ca și diferența normalizată dintre cameraTarget și cameraPosition

- Tot pe baza acestor componente se calculează cameraRightDiraction, care va fi produsul vectorial dintre cameraFrontDirection și cameraUp
- Mișcarea propriu zisă va fi calculată după apăsarea tastelor de mișcare, astfel încât cameraPosition și cameraTarget vor fi incrementate/decrementate conform direcției de mișcare

- **Rotația**
  - Rotația camerei se face pe axele X, respectiv Y
  - cameraTarget va fi recalculată pe baza următoarelor formule

```
this->cameraTarget.x = this->cameraPosition.x + sin(yaw) * cos(pitch);
this->cameraTarget.y = this->cameraPosition.y + sin(pitch);
this->cameraTarget.z = this->cameraPosition.z - cos(yaw) * cos(pitch);
```

  - cameraFrontDirection și cameraRightDirection vor fi reactualizate conform formulelor de la mișcare
- **Animația de prezentare**
  - Inițial mi-am fixat punctele în care doresc să ajungă camera
  - După ce am fixat punctele, am mișcat și rotit camera astfel încât să treacă prin toate acestea
  - De asemenea, am importat librăriile windowsx.h și mmystem.h pentru a putea adăuga muzică pe fundal folosind funcția PlaySound
- **Efectul de ceață**
  - Efectul de ceață este creat în shader-ul basic.frag
  - Ceața este una exponențial pătratică și este creată astfel:

```
float computeFog()
{
    float fogDensity = 0.05f;
    float fragmentDistance = length(fPosEye);
    float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));
    return clamp(fogFactor, 0.0f, 1.0f);
}
```

- **Efectul de ploaie**
  - Pentru a crea efectul de ploaie am importat un nou obiect care să reprezinte un picure de ploaie
  - După ce am inițializat obiectul, am creat 100 de modele identice cu acesta, pe care le am translatat în diferite poziții din interiorul scenei
  - În momentul în care acest efect este pornit, stropii de ploaie sunt translatați pe direcția Y în jos
  - În momentul în care stropii ating solul, aceștia sunt translatați la înălțimea 15, astfel încât să se creeze efectul de buclă

- Totodată, acest efect beneficiază și de un sunet de fundal adăugat cu ajutorul funcției PlaySound

```
myBasicShader.useShaderProgram();
for (int i = 0; i < noDrops; i++) {
    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
    waterDropY[i] -= 0.1f;
    model3[i] = glm::translate(glm::mat4(1.0f), glm::vec3(waterDropX[i], waterDropY[i], waterDropZ[i]));
    if (waterDropY[i] < 0) {
        waterDropY[i] = 15;
    }
    normalMatrix3[i] = glm::mat3(glm::inverseTranspose(view * model3[i]));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model3[i]));
    glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix3[i]));
}
```

- **Efectul de spotlight**
  - Acest efect înlocuiește lumina globală cu o lumină punctiformă care are originea în originea camerei
  - Astfel, a fost nevoie să trimit către shader poziția camerei, dar și direcția de înainte
  - Diferența majoră față de lumina globală este dată de faptul că trebuie creat un con de lumină, care să lumineze doar între anumite valori ale unghiului de vizualizare

```
float theta = dot(lightDirN, normalize(-lightDirFront));
float epsilon = 0.3f;
float intensity = clamp((theta - 0.2f)/epsilon, 0.0, 1.0);
diffuse *= intensity;
specular *= intensity;

float constant = 1.0f;
float linear = 0.09f;
float quadratic = 0.032f;

float distance = length(lightPos - fPosEye.xyz);
float attenuation = 1.0f / (constant + linear * distance + quadratic * distance * distance);
```

- **Luminile direcționale**
  - Luminile direcționale sunt luminile globale ale scenei
  - Acestea sunt în număr de două și sunt create în shader
  - Pentru ca aceste două lumini să provină din direcții diferite, am creat două direcții înspre care acestea să lumineze
  - Astfel, shader-ul creează luminile în funcție de aceste direcții
  - Pentru a opri oricare dintre aceste două lumini, doar setez valoare culorii respective luminii pe care doresc să o opresc pe 0
  - Fiecare lumină va avea componenta ambientală, difuză și specular proprie

```
//compute eye space coordinates
fPosEye = view * model * vec4(fPosition, 1.0f);
vec3 normalEye = normalize(normalMatrix * fNormal);

//normalize light direction
vec3 lightDirN = vec3(normalize(view * vec4(lightDir, 0.0f)));

//compute view direction (in eye coordinates, the viewer is situated at the origin
vec3 viewDir = normalize(- fPosEye.xyz);

//compute ambient light
ambient = ambientStrength * lightColor;

//compute diffuse light
diffuse = max(dot(normalEye, lightDirN), 0.0f) * lightColor;

//compute specular light
vec3 reflectDir = reflect(-lightDirN, normalEye);
float specCoeff = pow(max(dot(viewDir, reflectDir), 0.0f), 32);
specular = specularStrength * specCoeff * lightColor;

// light 2
//normalize light direction
vec3 lightDirN2 = vec3(normalize(view * vec4(lightDir2, 0.0f)));

ambient2 = ambientStrength * lightColor2;

//compute diffuse light
diffuse2 = max(dot(normalEye, lightDirN2), 0.0f) * lightColor2;

//compute specular light
vec3 reflectDir2 = reflect(-lightDirN2, normalEye);
float specCoeff2 = pow(max(dot(viewDir, reflectDir2), 0.0f), 32);
specular2 = specularStrength * specCoeff2 * lightColor2;
```

- **Efectul de fulger**
  - Acest efect este creat doar prin înmulțirea luminii cu o variabilă pe care o cresc și o scad treptat după anumite repere de timp

```
if (enableThunder == true){
    ambientStrength += extraLight;
}
```

  - Totodată, acest efect are pe fundal și un sunet care să imite trăznetul, adăugat cu ajutorul funcției PlaySound
- **Vizualizarea scenei în modurile solid, wireframe, punctiform**
  - Se realizează cu ajutorul cu ajutorul funcției glPolygonMode

```
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

```
glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
```

iii. Motivarea abordării alese

Soluția a fost aleasă astfel încât să folosesc cât mai multe din informațiile prezentate în cadrul laboratorului, dar și astfel încât scena să arate cât mai bine.

Pentru crearea tuturor efectelor m-am folosit de exemplele găsite în lucrările de laborator.

Totodată, am ales și obiecte care să arate cât mai aproape de realitate.

b. Modelul grafic

Modelul grafic are la bază modelele grafice prezentate în lucrările de laborator, dar și obiectele găsite online pe care le-am introdus în scenă.

Astfel, pentru iluminarea scenei am folosit două surse de lumină care au la bază modelul de iluminare Phong.

În cazul efectului de ceață am ales metoda exponențial pătratică, aceasta reușind să creeze o ceață cât mai realistă.

Pentru crearea efectului de fulger am folosit modelul de iluminare Phong, diferența fiind făcută de faptul că am înmulțit componenta ambientală a luminii cu o variabilă.

Efectul de tranziție de zi-noapte este creat prin micșorarea, respectiv creșterea luminii transmise către shader.

Pentru efectul de spotlight am folosit modelul de iluminare Phong, diferența fiind făcută de conul de umbră care a trebuit creat (pentru a nu ilumina decât pe direcția înainte), dar și de originea luminii care acum se află în originea camerei.

c. Structuri de date

Pentru realizarea operațiilor necesare am folosit structurile de date disponibile în cadrul librăriei OpenGL, precum GLuint, GLFWwindow etc.

d. Ierarhia de clase

Pentru a implementa funcționalitățile descrise anterior, a fost necesară includerea unor fișiere de tip .cpp, dar și includerea unor fișiere de tip .hpp, dar și de includerea unor biblioteci:

- **Camera.hpp + Camera.cpp**
  - Folosite pentru mișcarea camerei

- **Mesh.hpp + Mesh.cpp**
  - Folosite pentru definirea vârfurilor unui obiect
- **Model3D.hpp + Model3D.cpp**
  - Folosite pentru crearea unui model 3D
- **Shader.hpp + Shader.cpp**
  - Folosite pentru încărcarea shadere-lor
- **SkyBox.hpp + SkyBox.cpp**
  - Folosite pentru încărcarea skybox-ului

- **Window.hpp + Window.cpp**
  - Folosite pentru crearea ferestrei
- **Stb_image.hpp + Stb_image.cpp**
- **Tiny_obj_loader.hpp + Tiny_obj_loader.cpp**
- **GLEW.h + GLFW.h**
  - Folosite pentru diferite funcționalități
- **Random.h**
  - Folosite pentru generarea de numere aleatoare
- **Window.h + MMSystem.h**
  - Folosite pentru adăugarea de sunet

De asemenea, a mai fost nevoie și de folosirea unor fișiere de tip shader:

- **Basic.frag + Basic.vert**
  - Pentru generarea luminii scenei
- **SkyBoxShader.frag + SkyBoxShader.vert**
  - Pentru generarea luminii skybox-ului

4. Prezentarea interfeței grafice (manual de utilizare)

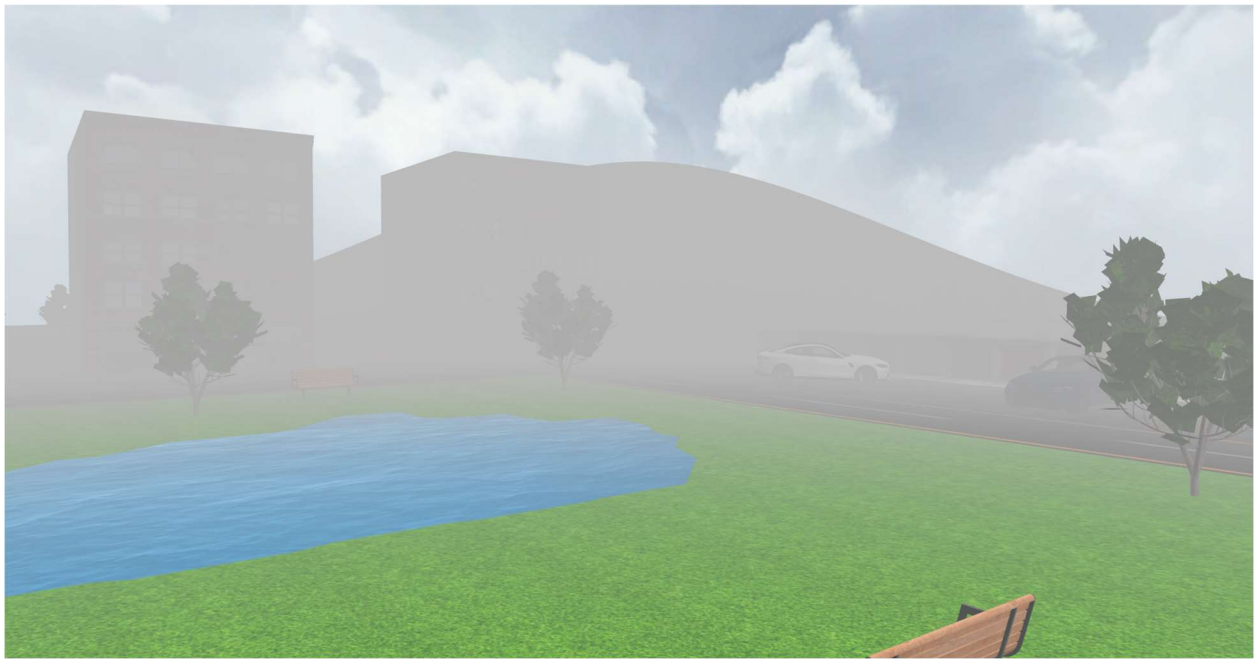
Pentru implementarea diferitelor funcționalități ale aplicației, se vor utiliza următoarele taste:

- W – mișcare înainte
- A – mișcare stânga
- S – mișcare înapoi
- D – mișcare dreapta
- R – mișcare sus
- F – mișcare jos
- ↑ – rotație cameră sus

- ↓ – rotație cameră jos
- → – rotație cameră dreapta
- ← – rotație cameră stânga
- 1 – pornire animație prezentare
- 2 – oprire animație prezentare
- 3 – pornire ceață
- 4 – oprire ceață
- 5 – pornire ploaie
- 6 – oprire ploaie
- 7 – pornire spotlight
- 8 – oprire spotlight
- 0 – readucerea camerei în poziția inițială
- U – pornire lumină 1
- I – oprire lumină 1
- O – pornire lumină 2
- P – oprire lumină 2
- K – pornire efect de fulger
- L – oprire efect de fulger
- Z – vizualizare scenă în mod solid
- X – vizualizare scenă în mod wireframe
- C – vizualizare scenă în mod punctiform
- V – rotație semn de circulație stânga
- B – rotație semn de circulație dreapta
- N – pornire animație zi-noapte
- M – oprire animație zi-noapte

-Scena completă-

-Scena cu efect de ceață-



-Scena cu efect de ploaie-

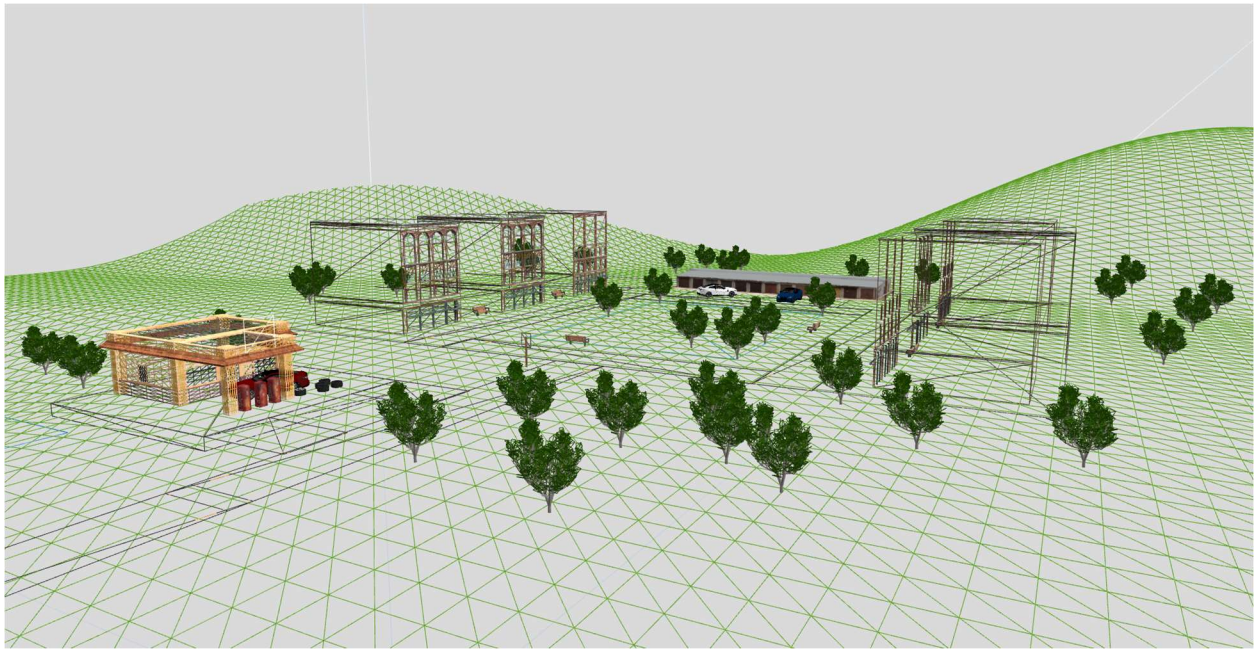
-Scena cu efect de spotlight-

-Scena doar cu lumina 1-

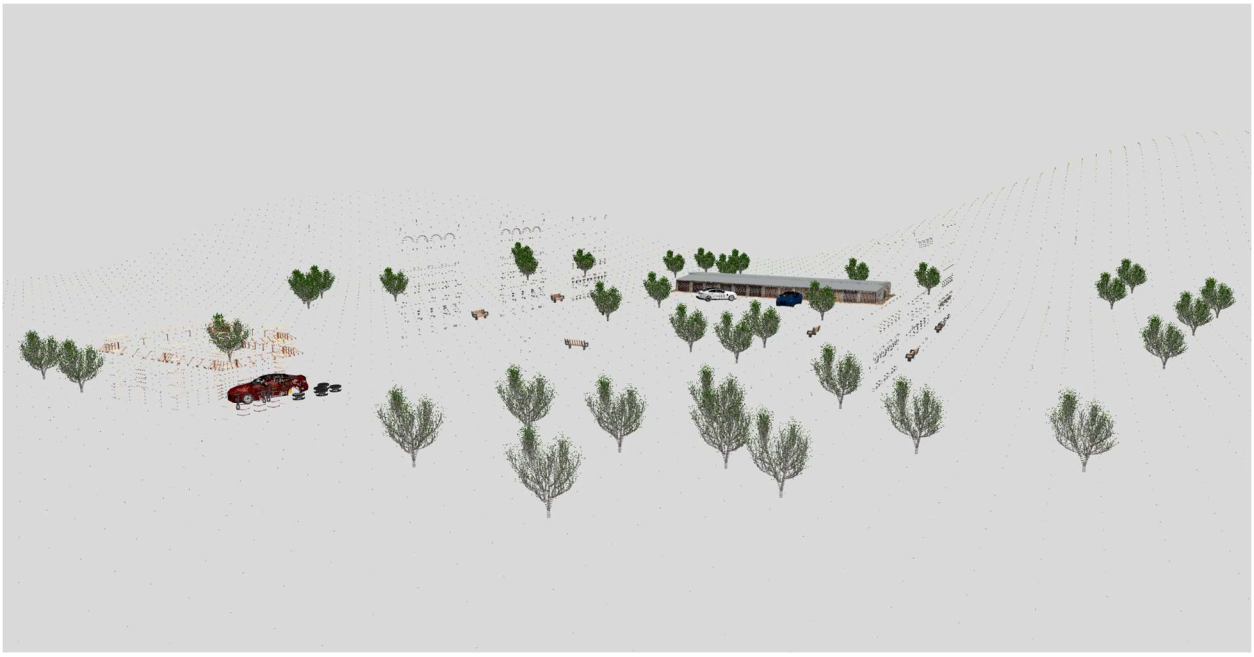
-Scena doar cu lumina 2-

-Scena cu efect de fulger-

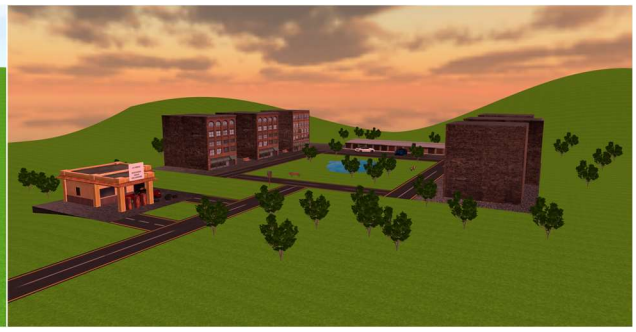
-Scena în mod wireframe-

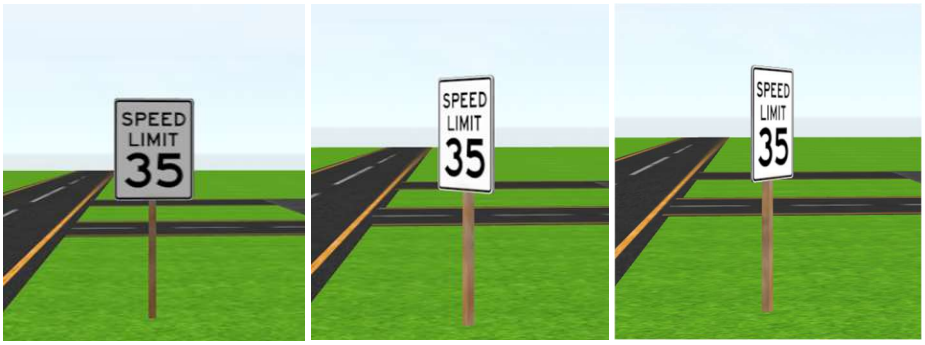


-Scena în mod punctiform-



-Scena cu animație zi-noapte-





5. Concluzii și dezvoltări ulterioare

Aplicația a fost creată pentru a reda scena unui mic orășel, utilizatorul având control asupra direcției de deplasare, dar totodată și asupra diferitelor efecte care pot fi aplicate (ceață, ploaie, fulger etc).

Realizarea acestui proiect mi-a îmbunătățit abilitățile de a lucra în aplicația 3D Blender, întrucât la momentul începerii proiectului nu aveam nicio idee despre cum se folosește această aplicație.

De asemenea, realizarea proiectului m-a ajutat să înțeleg mai bine cum se încarcă obiecte pentru crearea unei scene in OpenGl, cum se operează cu aceste obiecte și totodată cum pot fi create diferite efecte asupra scenei.

Ca și dezvoltări ulterioare asupra aplicației, pot fi luate în considerare următoarele:

- Mărirea suprafeței scenei
- Adăugarea de alte obiecte precum stâlpi de iluminat, animale etc
- Mișcarea mașinilor
- Adăugarea de umbre

6. Referințe

- https://docs.google.com/document/d/1njtWPMmOQNIaD_z9ve8iPRUqQTWdIV_PO-NvPD0nOuM/edit
- https://free3d.com/
- https://www.youtube.com/playlist?list=PLrgcDEgRZ_kndoWmRkAK4Y7ToJdOf-OSM
- https://docs.gl/
- https://www.youtube.com/watch?v=MAJqiDll0a8&ab_channel=OGLDEV
- https://sketchfab.com/3d-models/popular