UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Documentație proiect : Prelucrare Grafică

Nume student : Ilieșiu Robert-Mircea
Grupa : 30234
Profesor îndrumător : Nandra Cosmin

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Cuprins:

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

1. Prezentarea temei

Scopul proiectului constă în realizarea unei prezentări fotorealiste a unei scene de obiecte 3D.

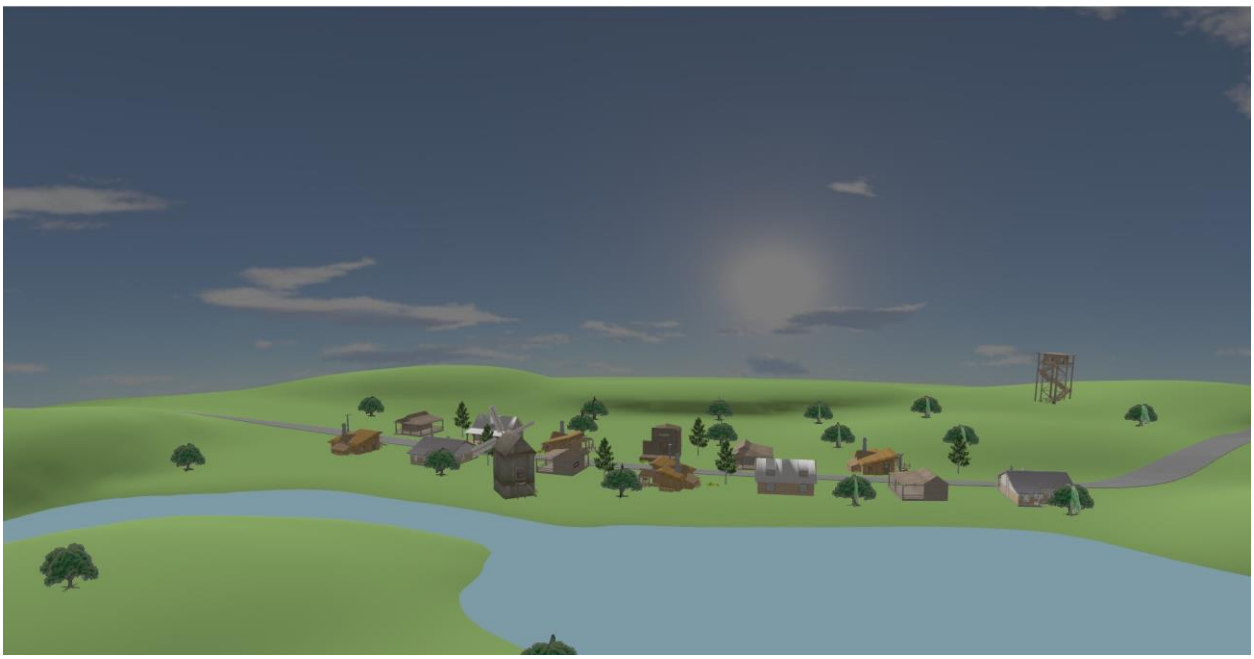
Pentru realizarea scenei se pun la dispozitie librăriile prezentate pe parcursul laboratoarelor. Obiectele au fost modelate și așezate în scena, în aplicatia Blender. Prin intermediul acestei aplicații am setat coordonatele dorite pentru obiectele scenei. Ulterior scena a fost exportată în proiect, în OpenGL.

Tema proiectului este un sat european, având ca obiecte 4 tipuri de case, o bancă, o moara de vând, un turn observator și mai multe obiecte care au ajutat la realizarea decorului scenei.

2. Scenariul

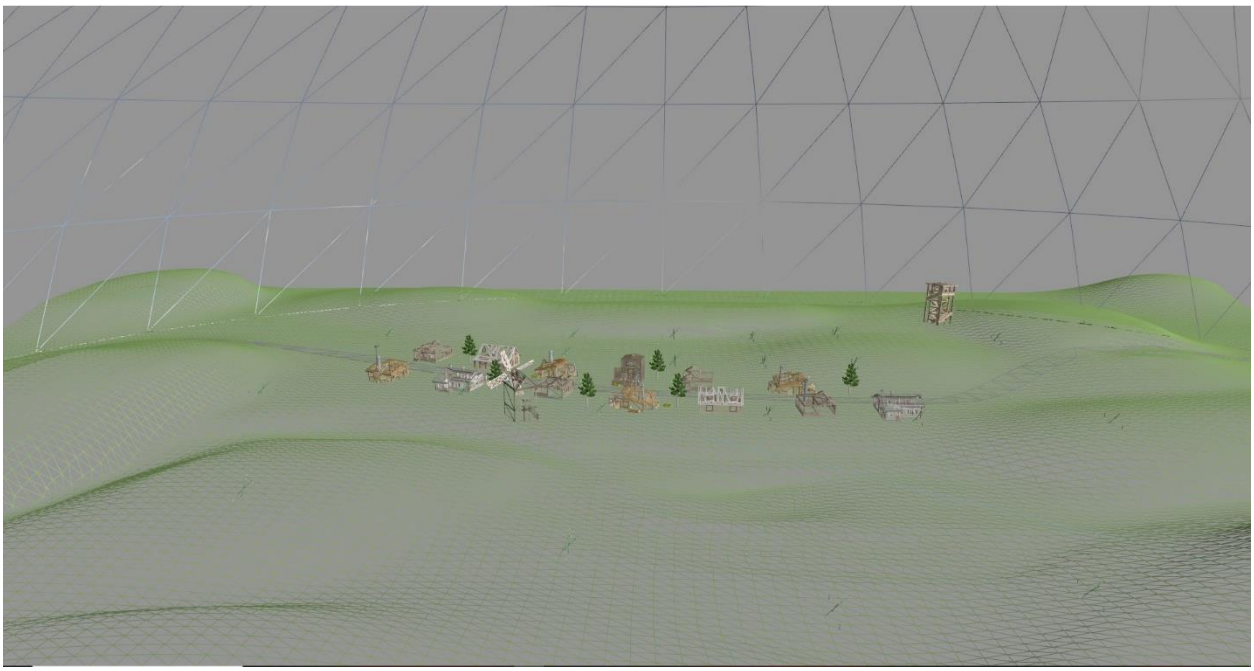
2.1. Descrierea scenei și obiectelor

Deschiderea aplicație OpenGL, va rezulta urmatoarea scenă:



Obiectele din cadrul scenei sunt: 4 tipuri de case, un turn observator, o moară de vând, o bancă și cateva elemente decorative : 2 tipuri de copaci și un tufiș.

Scena poate fi vazută din perspectiva a mai multor moduri de vizualizare:



Modul Wireframe

Modul Poligon

2.2. Funcționalități

Pentru deplasarea în scenă am folosit tastele (W – față, A - stânga, S - spate, D - dreapta) și navigarea cu mouse-ul pentru a urca/coborâ în altitudine. Prin tastele “Z” și “C” se poate roti camera de vizualizare la dreapta/stânga. Deasemenea apăsarea tastei “P” va activa prezentarea automată a scenei.

Totodată, există o sursă de lumină direcțională pentru a reprezenta soarele în scenă și 4 surse de lumină punctiforme pentru a reprezenta iluminatul stradal, ativată cu ajutorul tastei “Y”.

Prin apăsarea tastei “U” se activează vizualizarea umbrelor în scena, prin tasta “N” se activează “Modul noapte”, prin tasta “T” se activează animația pentru rotirea morii de vânt, prin apăsarea tastei “F” se activează ceața în scenă iar prin apăsarea tastei “R” se activează ploaie.

Deasemenea în scenă este implementat efectul de transparență pe 5 sticle și efectul fragment discarding pe un tip de copac des întâlnit în scenă.

3. Detalii de implementare

3.1. Funcții și algortmi

3.1.1. Soluții posibile

Iluminarea: utilizarea shading-ului Phong pentru un efect realist.
Umbrele: generarea unei hărți de adâncime pentru proiectarea umbrelor.
Ploaia: simularea fizicii gravitaționale pentru picăturile de ploaie.
Animația: rotația continuă a morii de vând.
Ceața: culoarea fundalului este calculată în funcție de factorul de ceață (FogFactor).
Noaptea: schimbarea luminozității în scenă.
Transparența: simularea efectului de transparență prin combinarea culorii obiectului și a culorii actuale din framebuffer într-o singură culoare.
Eliminarea fragmentelor: se verifică valoarea canalului alfa al unei texturi urmând să eliminat fragmentul care are valoarea este mai mică de 0,1 obținând astfel doar obiectul decupat(fără fundalul lui) în scenă.

3.1.2. Motivarea abordării alese

Lumina direcțională: pentru realizarea luminii direcționale am folosit modelul de iluminare Phong, având rol principal în crearea realismului din scenă și un rol esențial în evindențierea umbrelor.

Vectorul direcției luminii (**lightDir**) este definit și rotit pentru a simula mișcarea sursei de lumină (soarele). Acest vector este transformat în spațiul vizual folosind matricea **view** și

matricea de rotație **lightRotation**.

```
lightDir = glm::vec3(0.0f, 1.0f, 2.0f);
lightRotation = glm::rotate(glm::mat4(1.0f), glm::radians(lightAngle), glm::vec3(0.0f, 0.0f, 1.0f));
lightDirLoc = glGetUniformLocation(myCustomShader.shaderProgram, "lightDir");
glUniform3fv(lightDirLoc, 1, glm::value_ptr(glm::inverseTranspose(glm::mat3(view * lightRotation)) * lightDir));
```

Pentru înplementarea în shader-ul fragment:

Transformarea normalelor: Normalizăm normalele în coordonate normalEye pentru a asigura calcule pentru iluminare sunt corecte.

Direcția luminii: Vectorul direcției luminii (lightDir) este normalizat,pentru a ne asigura că lungimea vectorului este egală cu 1.

Direcția de vizualizare: Se calculează vectorul care indică direcția din poziția fragmentului (fPosEye) spre camera virtuală, plasată la originea coordonatelor normalEye.

Componenta lumină ambientală: Aceasta reprezintă baza iluminării în scenă și nu depinde de poziția camerei.

Componenta lumină difuză: Aceasta simulează reflexia luminii într-un mod uniform în toate direcțile, indiferent de direcția de vizualizare.

Componenta lumină speculară: Aceasta simulează reflexia luminii pe suprafețe lucioase și crează aceste reflexii în funcție de direcția de vizualizare.

```
void computeLightComponents()
{
	vec3 cameraPosEye = vec3(0.0f);//in eye coordinates, the viewer is situated at the origin
	//transform normal
	vec3 normalEye = normalize(fNormal);
	//compute light direction
	vec3 lightDirN = normalize(lightDir);
	//compute view direction
	vec3 viewDirN = normalize(cameraPosEye - fPosEye.xyz);
	//compute ambient light
	ambient = ambientStrength * lightColor;
	//compute diffuse light
	diffuse = max(dot(normalEye, lightDirN), 0.0f) * lightColor;
	//compute specular light
	vec3 reflection = reflect(-lightDirN, normalEye);
	float specCoeff = pow(max(dot(viewDirN, reflection), 0.0f), shininess);
	specular = specularStrength * specCoeff * lightColor;
}
```

Lumina punctiformă: sunt surse de lumină cu o poziție dată care iluminează radial și uniform în toate direcțiile. Razele generate de luminile punctiforme se estompează în funcție de distanță, făcând astfel obiectele mai apropiate de sursă să pară mai iluminate decât obiectele mai îndepartate.

Am definit vectorul de pozitie a luminii punctiforme și am trimis aceste poziții la shader-ul de fragmente.

```
glm::vec3 lightPos1 = glm::vec3(-1.41646f, -0.661937f, -3.34227f);
lightPos1Loc = glGetUniformLocation(myCustomShader.shaderProgram, "lightPos1");
glUniform3fv(lightPos1Loc, 1, glm::value_ptr(lightPos1));
```

În shader am folosit formula generală pentru iluminarea punctiformă care depinde de componentele ambientală, difuză și specular, ajustate în funcție de distanța fața de sursă.

```
vec3 computePointLight(vec4 lightPosEye)
{
	vec3 cameraPosEye = vec3(0.0f);
	vec3 normalEye = normalize(fNormal);
	vec3 lightDirN = normalize(lightPosEye.xyz - fPosEye.xyz);
	vec3 viewDirN = normalize(cameraPosEye - fPosEye.xyz);
	vec3 ambient = ambient * lightColor;
	vec3 diffuse = max(dot(normalEye, lightDirN), 0.0f) * lightColor;
	vec3 halfVector = normalize(lightDirN + viewDirN);
	float specCoeff = pow(max(dot(normalEye, halfVector), 0.0f), shininess);
	vec3 specular = specularStrength * specCoeff * lightColor;
	float distance = length(lightPosEye.xyz - fPosEye.xyz);
	float att = 1.0f / (constant + linear * distance + quadratic * distance * distance);
	return (ambient + diffuse + specular) * att * vec3(1.0f, 1.0f, 1.0f);
}
```

Umbrele: sunt generate folosind algoritmul **Shadow Mapping**, care constă din două etape principale:

- **Rasterizarea scenei din punctul de vedere al luminii**. Nu contează cum arată scena iar singurele informații de care avem nevoie sunt valorile de adâncime. Aceste valori sunt stocate într-o **hartă de adâncime** și pot fi obținute prin crearea unei texturi de adâncime, atașarea la un obiect framebuffer și rasterizarea întregii scene. În acest fel, textura de adâncime este umplută direct cu valorile relevante ale adâncimii.

- **Rasterizarea scenei din punct de vedere a poziție camerei**. Se compară adâncimea fiecărui fragment vizibil cu valorile de adâncime din **harta de adâncime**. Fragmentele care au o adâncime mai mare decât cea care a fost stocată anterior în **harta de adâncime** nu sunt direct vizibile din punctul de vedere al luminii.

Generarea hărții de adâncime: scena se rendeaza din perspectiva sursei de **lumină direcțională**. Valorile de adâncime (Z-buffer) sunt salvate într-o textură de adâncime (**depth texture**).

```
void initFBO() {
	//generate FBO ID
	glGenFramebuffers(1, &shadowMapFBO);
	//create depth texture for FBO
	glGenTextures(1, &depthMapTexture);
	glBindTexture(GL_TEXTURE_2D, depthMapTexture);
	glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT,
		SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT,
GL_FLOAT, NULL);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
	float borderColor[] = { 1.0f, 1.0f, 1.0f, 1.0f };
	glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
	//attach texture to FBO
	glBindFramebuffer(GL_FRAMEBUFFER, shadowMapFBO);
	glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
GL_TEXTURE_2D, depthMapTexture,
		0);
	glDrawBuffer(GL_NONE);
	glReadBuffer(GL_NONE);
	glBindFramebuffer(GL_FRAMEBUFFER, 0);
}
```

Matricea de transformare în spațiul luminii: se construiește o matrice care combină proiecția perspectivă și vizualizarea din perspectiva sursei de lumină direcționale.

```
glm::mat4 computeLightSpaceTrMatrix() {
	glm::mat4 lightView = glm::lookAt(glm::inverseTranspose(glm::mat3(lightRotation)) * lightDir,
glm::vec3(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	const GLfloat near_plane = -6.0f, far_plane = 20.0f;
	glm::mat4 lightProjection = glm::ortho(-20.0f, 20.0f, -20.0f, 20.0f, near_plane, far_plane);
	glm::mat4 lightSpaceTrMatrix = lightProjection * lightView;
	return lightSpaceTrMatrix;
}
```

Compararea cu harta de adâncime: se compară adâncimea punctului curent din perspectivă luminii direcționale cu valoarea stocată în harta de adâncime.

```
float computeShadow()
{
        // perform perspective divide
        vec3 normalizedCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
        // Transform to [0,1] range
        normalizedCoords = normalizedCoords * 0.5 + 0.5;
        // Get closest depth value from light's perspective
        float closestDepth = texture(shadowMap, normalizedCoords.xy).r;
        // Get depth of current fragment from light's perspective
        float currentDepth = normalizedCoords.z;
        if (normalizedCoords.z > 1.0f)
                return 0.0f;
        // Check whether current frag pos is in shadow
        float bias = 0.005f;
        float shadow = currentDepth - bias > closestDepth ? 1.0f : 0.0f;
        return shadow;
}
```

Ploaia: pentru simularea fizicii de cădere a picăturilor de ploaie, am creat 2 funcții care ajută la crearea acestui efect.

Inițializarea picăturilor de ploaie: Funcția initRainDrop() ne inițiază o structură care ne reprezintă o picătură de ploaie. În structura găsim pozitia picăturii de ploaie(**drops.position**), viteza picăturii (**drops.velocity**) și mărimea acestia(**drops.size**).

```
void initRainDrop(int numRaindrops, float groundHeight) {
        for (int i = 0; i < numRaindrops; i++) {
                RainDrops drops;
                drops.position = glm::vec3(rand() % 25 - 13,
                                           rand() % 20 + 5,
                                           rand() % 25 - 13);

                drops.velocity = glm::vec3(0.0f, -9.8f, 0.0f);
                drops.size = 0.7;

                raindrops.push_back(drops);
        }
}
```

Simularea căderii picăturilor de ploaie: Funcți updateRainDrops() ne ajută simulăm efectul de cădere a picăturilor de ploaie la un interval de timp(**deltaTime**) iar dacă picăturile au ajus la baza scenei, acestea sunt reinițializate cu o poziție aleatorare.

```
void updateRainDrops(float deltaTime, float groundHeight) {
        for (RainDrops& drops : raindrops) {
                drops.velocity.y -= 9.8f * deltaTime;
                drops.position += drops.velocity * deltaTime;

                if (drops.position.y < groundHeight) {
                        drops.position.y = rand() % 20 + 5;
                        drops.position.x = rand() % 25 - 13;
                        drops.position.z = rand() % 25 - 13;

                        // Resetez viteza de cadere
                        drops.velocity.y = -9.8f;
                }
        }
}
```

Animația: consta în translația obiectului în origine, rotatia acestuia și translația lui inapoi la poziția inițială.

```
modelElice = glm::mat4(1.0f);
modelElice = glm::translate(modelElice, glm::vec3(0.0f, -1.0f, 0.0f));
modelElice = glm::scale(modelElice, glm::vec3(0.5f));
modelElice = glm::translate(modelElice, glm::vec3(-1.62485f, 1.35772f, 6.14168f));
modelElice = glm::rotate(modelElice, glm::radians(angleElice), glm::vec3(1.0f, 0.0f, 0.0f));
modelElice = glm::translate(modelElice, glm::vec3(1.62485f, -1.35772f, -6.14168f));
glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE,
glm::value_ptr(modelElice));

if (!depthPass) {
        normalMatrix = glm::mat3(glm::inverseTranspose(view * modelElice));
        glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));
}

eliceMoara1.Draw(shader);
```

Ceața: pentru implementarea efectului de ceata putem pune aplica folosirea unei funcții de interpolare liniară care va amesteca culoarea cetii cu aceea a fragmentului, pe baza distanței fragmentului.

```
float applyFog() {
        float fogDensity = 0.1f;
        float fragmentDistance = length(fPosEye);
        float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));
        return clamp(fogFactor, 0.0f, 1.0f);
}
```

Noaptea: se aplică o schimbarea luminozitate în scenă.

```
if (enableNight) {
                fColor = vec4(0.0f, 0.0f, 0.0f, 1.0f);
        }
```

Transparență: se aplică o combinare a colorii obiectului cu cea a culorii actuale din framebuffer iar cantitatea de transparența a obiectului este definită canalul alfa din reprezentarea culorii. Dacă a patra componentă a culorii este 1.0f, atunci obiectul este opac, dacă valoarea este 0.0f atunci obiectul este transparent.

Deasementea obiectele transparente trebuie desenate după obiectele opace.

```
drawObjects(myCustomShader, false);

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
drawTransparentObjects(myCustomShader, false);
glDisable(GL_BLEND);
```

Eliminarea fragmentelor: se aplica o verificare pe valoarea alfa din texture care poate fi utilizată pentru a elimina un fragment.

```
vec4 colorFromTexture = texture(diffuseTexture, fTexCoords);
if (colorFromTexture.a < 0.1) discard;
```

3.2.Modelul grafic

Modelul grafic al proiectului se bazează pe utilizarea fișierelor de tip **obj** pentru modelele 3D, care sunt după încărcate în scenă. Fiecare model este asociat cu texturi pentru un aspect realist. Texturile sunt mapate pe obiecte utilizând coordonate UV definite în fișierele modelului.

Obiecte folosite:

- Scena principală: Model 3D complex cu mai multe texturi aplicate pentru detalierea suprafeței (iarbă, apă, lemn, copaci, plante, etc.)
- Elicea morii de vânt: Animată prin rotații continue, oferind astfel o scenă dinamică.
- Sticlele: Obiecte decorative care sunt duplicate și aplicat efectul de transparență acestea.
- Picăturile de ploaie: Create utilizând un sistem de particule care folosește un model obj simplu pentru fiecare particulă.

3.3. Structuri de date

- Vectori: utilizați pentru gestionarea picăturilor de ploaie. Fiecare particulă este reprezentată printr-o structură ce conține poziția, viteza și dimensiunea acesteia.
- Matrice 4x4: folosite pentru transformări precum translații, rotații și scalări ale obiectelor.

3.4. Ierarhia de clase

- **gps::Camera**: gestionează poziția, direcția, rotația camerei și permite deplasarea liberă prin scenă și schimbarea perspectivei utilizatorului.
- **gps::Model3D:** încarcă și redă modelele 3D utilizând fișiere OBJ și texturi asociate.
- **gps::Shader**: gestionează încărcarea și utilizarea shaderelor în OpenGL (vertex și fragment).

4. Prezentarea interfeței grafice utilizator/manual de utilizare

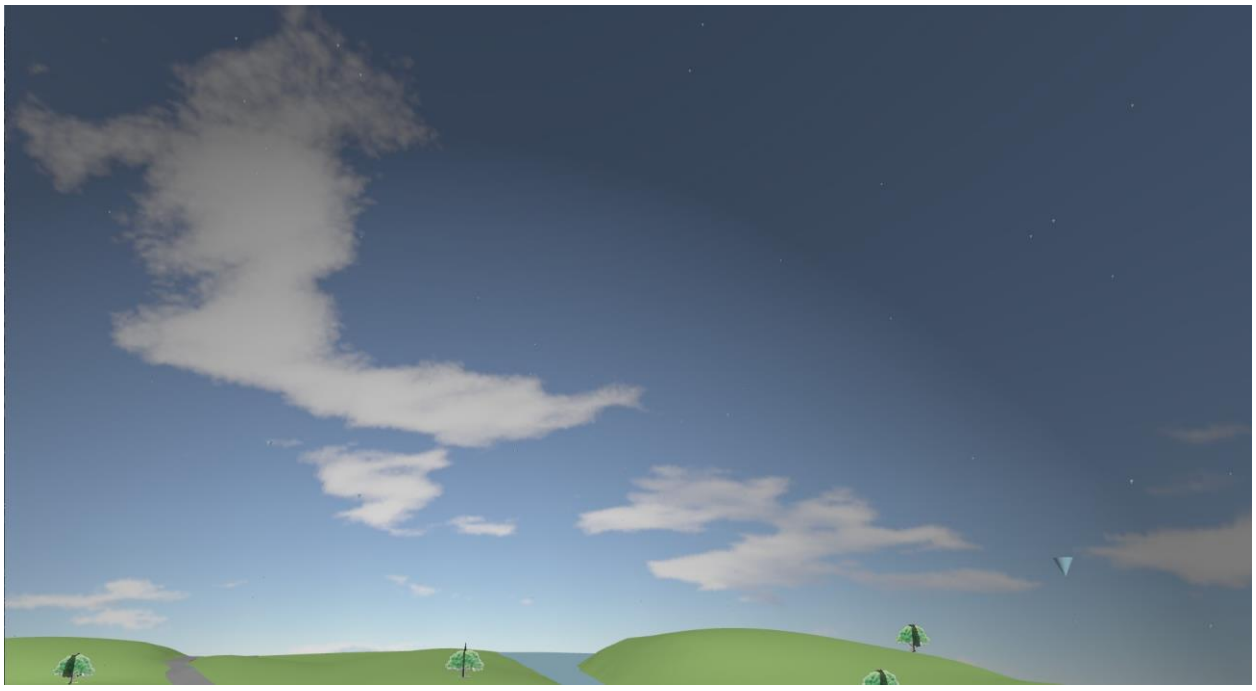
Interfața a fost prezentată la punctul 3. În ceea ce privește manualul de utilizare avem la dispoziție urmatoarele comenzi:

- Tasta “W”: deplasare înainte
- Tasta “S”: deplasare inapoi
- Tasta “A”: deplasare la stânga
- Tasta “D”: deplasare la dreapta
- Tasta “Z”: rotire spre stânga a cameră de vizualizare
- Tasta “C”: rotire spre dreapta a cameră de vizualizare
- Tasta “1”: modul solid
- Tasta “2”: modul wireframe
- Tasta “3”: modul poligon
- Tasta “P”: modul prezentare automată scenei
- Tasta “U”: vizualizarea umbrelor în scenă

- Tasta “J”: rotire sursă de lumină principală la stânga
- Tasta “L”: rotire sursă de lumină principală la dreapta

- Tasta “F”: vizualizarea cetei în scenă

- Tasta “R”: vizualizarea simulării de ploaie



- Tasta “N”: vizualizarea simulării pentru modul noapte

- Tasta “T”: prezentare animație
- Tasta “Y”: vizualizare surse de lumina ambientale (punctiforme)

- Mouse: deplasare în altitudinea și latitudinea scenei

5. Concluzii și dezvoltări ulterioare

Prin intermediul acestui proiect, ne-am dezvolatat sfera cunoștiințelor în elemente de grafică, familiarizându-ne cu anumite noțiuni, dar și cu prelucrarea obiectelor prin intermediul OpenGL și aplicației Blender.

Ca si dezvoltari ulterioare, proiectul ar putea avea mai multe obiecte sau o scenă mai comlexă și mult mai detaliată, dar și efecte de fotorealism precum ninsoare, tunet, lumină de tip spot (spot light) sau efecte sonore.

6. Referințe

- https://learnopengl.com/
- http://www.opengl-tutorial.org/
- https://stackoverflow.com/
- Laboratorare și Curs Prelucrare Grafică