MINISTERUL EDUCAȚIEI NAȚIONALE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA: Automatica si Calculatoare
SPECIALIZAREA: Calculatoare si Tehnologia Informatiei
DISCIPLINA: Prelucrare grafică
PROIECT: Alien Invasion OpenGL Scene

Prof. coordonator:

prof.ing. Nandra Cosmin

Student:

Ricu Alexandru Razvan

Grupa: 30234

AN UNIVERSITAR

2024- 2025

Cuprins

1. Prezentarea temei alese

Proiectul meu constă în realizarea unei scene 3D care reprezintă o invazie extraterestră pe o autostradă, utilizând efecte speciale și interactivitate pentru a crea o experiență captivantă. Elementele cheie ale scenei includ o lumină direcțională ce reprezintă soarele, precum și o lumină de tip **point light**, care poate fi activată de utilizator prin apăsarea unei taste.

Pentru implementare, am utilizat bibliotecile **OpenGL**, **GLFW**, și **GLM**, iar modelele 3D au fost realizate cu ajutorul **Blender**.

Funcționalitățile Scenei

În cadrul acestui proiect, am integrat următoarele funcționalități:

1. Controlul Luminii

- **Lumină direcțională**:
  - Simulează soarele și luminează întreaga scenă, evidențiind realismul și detaliile obiectelor.
- **Lumină de tip point light**:
  - Poate fi activată sau dezactivată de utilizator prin apăsarea unei taste.
  - Simulează soarele și luminează întreaga scenă, comportandu-se similar cu cea directionala dar scade in intensitate cu marirea distantei fata de sursa

2. Controlul Camerei

- Camera permite explorarea scenei prin translatie si rotație
- Utilizatorul poate naviga folosind:
  - **Mouse-ul** pentru a roti camera.
  - **Tastele săgeți** sau **W/A/S/D** pentru a translata.

3. Efecte Speciale

- **Ceață**: Adăugată pentru a crea o atmosferă misterioasă.
- **Fulger**: Un efect vizual ce luminează întreaga scenă pentru scurte momente, simulând o furtună
- **Fragment discard**: Utilizat pentru creatia padurii.
- **Animații dinamice**: Navele extraterestre se deplasează deasupra autostrăzii, iar tancul isi misca turela.
- **Transparenta**: utilizata pentru sticlele de bautura si nava extraterestra cu cloak-ul activ.
- **Lumini Punctiforme**: reprezentate de luminile tancului si a celor doi stalpi de lumina.
- **Lumini spot**: reprezentate de functionalitatea de flashlight si de cele doua floodlights din turnurile de paza

4. Moduri de Vizualizare

- Modurile de vizualizare includ:
    - **Point mode** pentru a vedea punctele geometrice ale scenei.
    - **Wireframe mode** pentru o reprezentare schematică.

5. Umbre și Fotorealism

- Umbrele generate de lumina direcțională cat si de cea punctiforma adaugă realism scenei.
- Materialele texturate și transparența contribuie la un aspect vizual convingător.

Controlul Scenei

1. **Taste pentru control**:
    - X: Activează/Dezactivează lumina de tip **point light**.
    - W/A/S/D sau săgeți: Controlează translația camerei.
2. **Mouse**:
    - Mișcarea mouse-ului permite rotirea camerei pentru a explora scena.

Obiective și Scopuri

Acest proiect a avut ca scop:

- Familiarizarea cu programarea grafică utilizând **OpenGL**.
- Implementarea iluminării avansate, inclusiv **light switching** pentru dinamica scenei.
- Crearea unei experiențe vizuale captivante, utilizând efecte precum ceata, umbrele și fulgerele.
- Exersarea modelării 3D în **Blender** și integrarea acestor modele în aplicație.

Proiectul oferă o scenă interactivă și realistă, în care utilizatorul poate experimenta invazia extraterestră din diferite perspective, explorând lumina și efectele speciale.

2. Scenariul

2.1 Descrierea scenei si a obiectelor

Scena realizată reprezintă o **invazie extraterestră** într-un peisaj urban low-poly, cu elemente interactive și efecte vizuale care accentuează atmosfera dramatică. Aceasta a fost construită folosind modele 3D realizate în software-ul grafic **Blender** și completată cu modele externe în format **.obj** preluate de pe diverse platforme. Scena include multiple elemente dinamice și iluminate, toate integrate pentru a oferi o experiență vizuală captivantă.

Compoziția Scenei

1. Orașul Low-Poly

- În fundal, se află un **oraș low-poly**.
- **Intrările în oraș** sunt protejate de armată, cu:
    - **Turele de apărare**.
    - Un **tanc** amplasat în apropierea drumurilor principale, echipat cu un point light pentru iluminarea zonei.

2. Structuri de Supraveghere

- **Două turnuri de supraveghere** sunt amplasate în fața orașului, fiecare dotat cu **floodlights** pentru a ilumina împrejurimile și a crea o atmosferă tensionată.

3. Elemente de Iluminare

- Iluminarea scenei este gândită pentru a spori realismul:
    - **Tancul** dispune de un **point light**, simulând farurile acestuia.
    - **Stâlpii de electricitate**: Două stâlpi echipați cu **point lights** oferă iluminare ambientală pe drumurile care duc spre oraș.
    - **Lanterna camerei**: Este un **spotlight** mobil, care urmează direcția de mișcare a camerei, accentuând zonele de interes.
    - **Turnurile de supraveghere** sunt echipate cu floodlights pentru a oferi vizibilitate extinsă.

4. Drumuri și Poduri

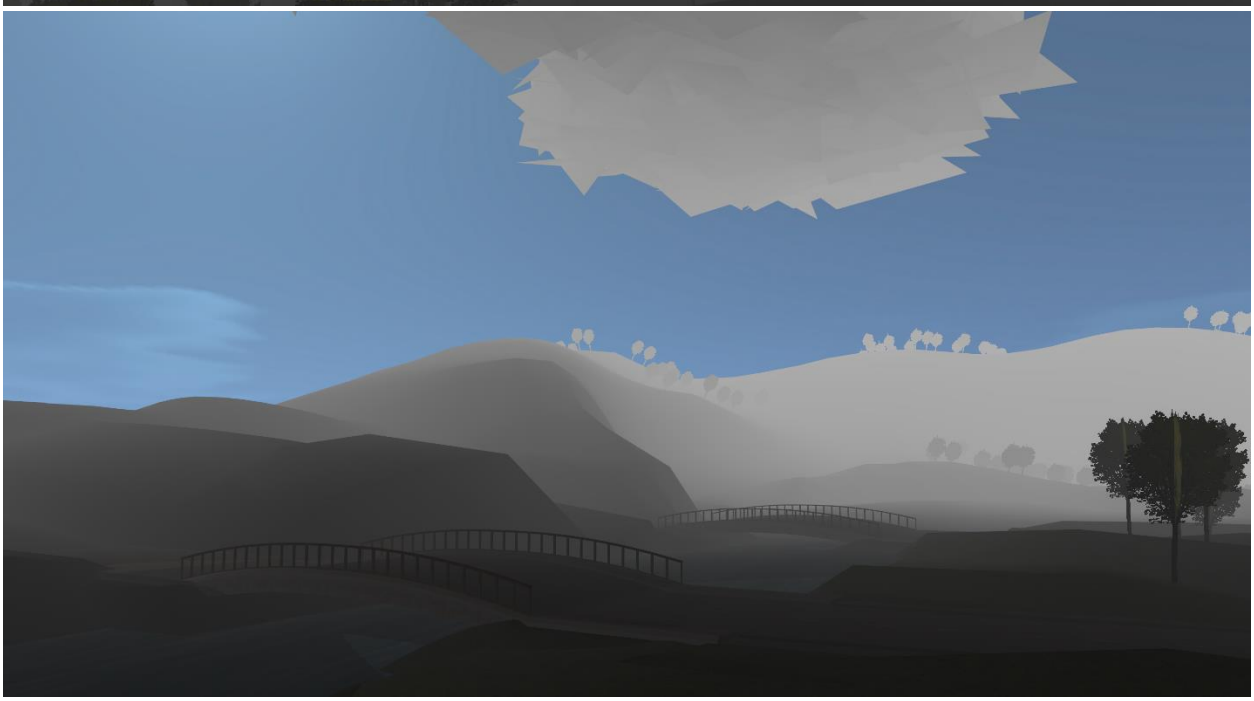
- Scena include **două drumuri paralele** ce duc către un **tunel**, care simbolizează intrarea principală în oraș.
- Două **poduri** traversează un **lac**, fiind elemente de legătură între zonele periferice și centrul scenei.

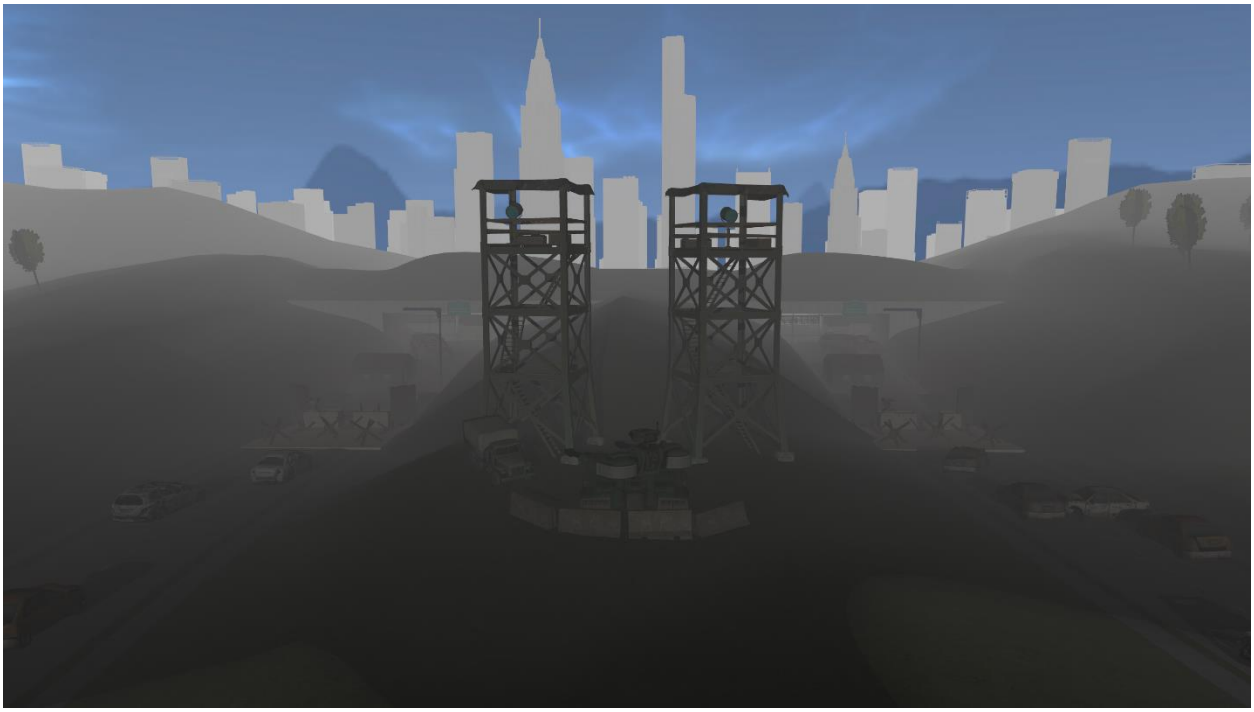
5. Pădure prin Fragment Discard

- O **pădure** înconjoară o parte a scenei, realizată utilizând tehnica **fragment discard** pentru a adăuga numerosi copaci.

6. Elemente transparente

- Nava și două sticle din scenă sunt realizate cu **transparență**.





2.2 Functionalitati

În această scenă 3D, am implementat mai multe funcționalități care adaugă dinamism, interactivitate și efecte vizuale spectaculoase. Fiecare element din scenă este dotat cu un comportament unic, care contribuie la realizarea unei atmosfere captivante și realiste. Mai jos sunt descrise principalele funcționalități implementate:

1. Rotirea Continua a Navei Extraterestre

- Folosind o funcție de animație, am aplicat o rotație incrementală continuă pe axa Y, pentru a obține efectul de rotație 360 de grade.

2. Tancul – Mișcarea Turelei și Rotirea Aleatoare

- Un timer este folosit pentru a măsura intervalele de 15 secunde, iar poziția turelei este schimbată într-un interval aleatoriu.
- Mișcarea se face în pași graduali, pentru a oferi un efect mai realist.

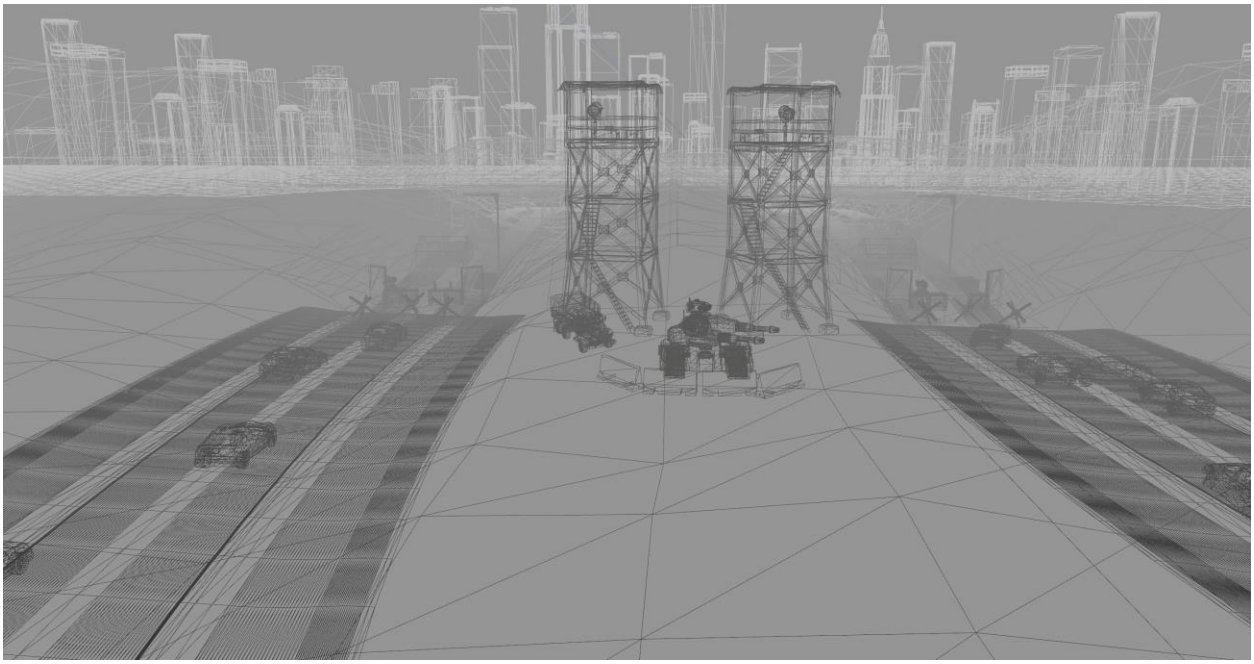
3. Furtuna cu Fulger – Efect Vizual de Lumină

- Fulgerul este activat la intervale aleatorii, iar atunci când fulgerul apare, toate sursele de lumină din scenă sunt intensificate temporar.

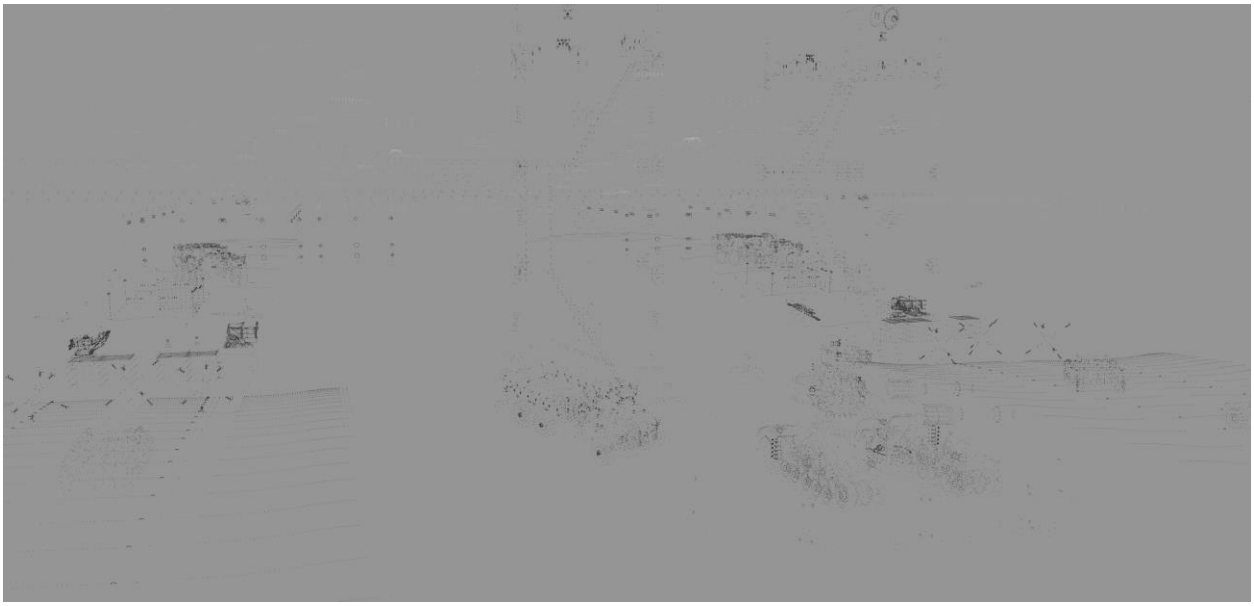
4. Lanterna – Spotlight Mobil

- Lanterna este o sursă de lumină **spotlight** care se mișcă odată cu direcția camerei, astfel încât întotdeauna să lumineze zona în care privirea utilizatorului se îndreaptă.
- Tasta **F** controlează activarea și dezactivarea lanternei, iar atunci când este activă, aceasta oferă o iluminare concentrată pe un punct specific din scenă.

- Tasta 1: vizualizarea scenei in modul de reprezentare **wireframe.**

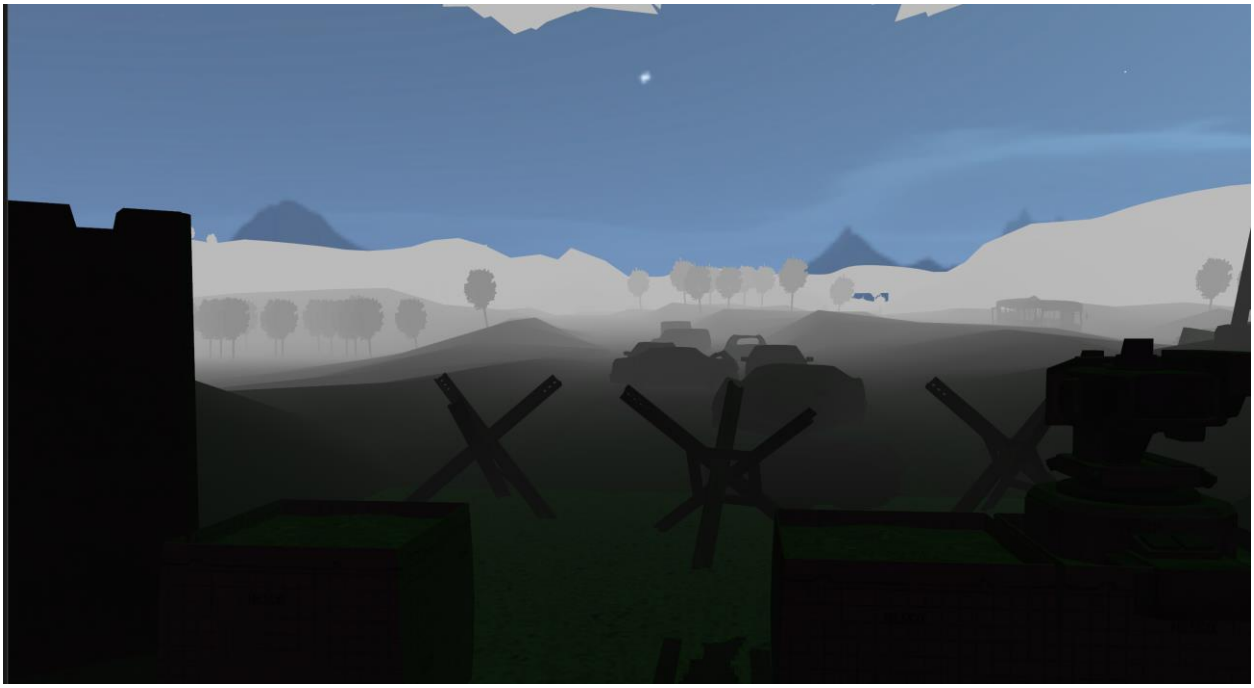


- Tasta 2: vizualizarea scenei in modul de reprezentare **punct.**



De asemenea, utilizatorul poate active/dezactiva lanterna folosind tasta F.

- Tasta X: activarea/dezactivarea luminii punctiforme ca sursa de lumina



3. Detalii de implementare

3.1 Functionalitati

- **Camera movement:**

  Navigarea camerei în OpenGL permite explorarea scenei 3D prin controlul tastaturii și mouse-ului. Utilizatorii pot mișca camera înainte, înapoi, stânga și dreapta cu tastele WASD, iar pe orizontală cu Z și C, în timp ce mișcarea mouse-ului controlează rotațiile camerei. Mișcarea fluidă este calculată cadru cu cadru, iar o matrice de proiecție pe perspectivă asigură un câmp vizual natural, oferind o experiență interactivă și intuitivă pentru explorarea completă a scenei.

```cpp
void Camera::move(MOVE_DIRECTION moveDir, float delta) {
    glm::vec3 movementOffset(0.0f);

    if (moveDir == MOVE_FORWARD) {
        movementOffset += delta * cameraFrontDirection;
    }
    else if (moveDir == MOVE_BACKWARD) {
        movementOffset -= delta * cameraFrontDirection;
    }
    else if (moveDir == MOVE_RIGHT) {
        movementOffset += delta * cameraRightDirection;
    }
    else if (moveDir == MOVE_LEFT) {
        movementOffset -= delta * cameraRightDirection;
    }

    cameraPosition += movementOffset;
    cameraTarget = cameraPosition + cameraFrontDirection;
}

void Camera::rotate(float deltaPitch, float deltaYaw) {
    pitch += deltaPitch;
    yaw += deltaYaw;

    pitch = glm::clamp(pitch, -89.0f, 89.0f);

    float cosPitch = cos(glm::radians(pitch));
    glm::vec3 newFront;
    newFront.x = cos(glm::radians(yaw)) * cosPitch;
    newFront.y = sin(glm::radians(pitch));
    newFront.z = sin(glm::radians(yaw)) * cosPitch;

    cameraFrontDirection = glm::normalize(newFront);
    cameraTarget = cameraPosition + cameraFrontDirection;
    cameraRightDirection = glm::normalize(glm::cross(cameraFrontDirection, cameraUpDirection));
}
```

- **Initializarea si desenarea obiectelor:**

  În OpenGL, desenarea unui obiect 3D implică activarea shaderului cu shader.useShaderProgram(), aplicarea transformărilor necesare cu funcții precum glm::rotate() și trimiterea matricei model către shader cu glUniformMatrix4fv(). În timpul unei treceri de adâncime (depthPass), matricea normală nu este transmisă, însă, în cazul iluminării, se calculează și se trimite normalMatrix, obținută din inversul și transpunerea matricei combinației vizualizare-transformare. Obiectul este apoi desenat prin apelul scene.Draw(shader), folosind toate datele și transformările procesate.

```
void initObjects() {
    scene.LoadModel("objects/scene/scena.obj");
    tankChasis.LoadModel("objects/tankChasis/tankChasis.obj");
    tankTurret.LoadModel("objects/turret/turret.obj");
    lightCube.LoadModel("objects/cube/cube.obj");
    screenQuad.LoadModel("objects/quad/quad.obj");
    glass.LoadModel("objects/bottles/bottle1.obj");
    glass2.LoadModel("objects/bottles/bottle2.obj");
    alien.LoadModel("objects/alienship/alien.obj");
    cloud.LoadModel("objects/cloud/untitled.obj");
    thunder.LoadModel("objects/thunder/thunder.obj");
    initSkybox();
}
```

```
shader.useShaderProgram();

model = glm::rotate(glm::mat4(1.0f), glm::radians(angleY), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE, glm::value_ptr(model));

// do not send the normal matrix if we are rendering in the depth map
if (!depthPass) {
    normalMatrix = glm::mat3(glm::inverseTranspose(view * model));
    glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));
}

scene.Draw(shader);

model = glm::translate(glm::mat4(1.0f), glm::vec3(-22.27f, 2.36f, -74.339f));
model = glm::scale(model, glm::vec3(0.800f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE, glm::value_ptr(model));

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

tankChasis.Draw(shader);
```

- **Desenarea obiectelor transparente si sortarea acestora dupa distanta de la camera:**

  În proiect, înainte de a desena obiectele transparente, calculez distanța lor față de cameră, le stochez într-un vector și le sortez în ordinea descrescătoare a adâncimii folosind std::sort. Ulterior, obiectele sunt desenate în această ordine. Transparența este activată cu glEnable(GL_BLEND) și configurată prin glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA) pentru a combina corect culorile obiectelor cu fundalul pe baza valorii alfa.

```
populateTransparentObjects();

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

```
void drawTransparentObjects(gps::Shader shader, bool depthPass) {
    shader.useShaderProgram();

    for (const TransparentObject& obj : transparentObjects) {
        model = glm::mat4(1.0f);
        model = glm::translate(model, obj.position);
        if (obj.scale == glm::vec3(1.000f)) {
            model = glm::rotate(model, glm::radians(shipRotationAngle), glm::vec3(0.0f, 1.0f, 0.0f));
        }
        model = glm::scale(model, obj.scale);

        glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE, glm::value_ptr(model));

        normalMatrix = glm::mat3(glm::inverseTranspose(view * model));
        glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));
        if (!depthPass) {
            normalMatrix = glm::mat3(glm::inverseTranspose(view * model));
            glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));
        }
        obj.object->Draw(shader);
    }
}
```

- **Animatia turelei tancului:**

Animația turelei tancului constă în două etape: rotația către un unghi țintă și o pauză temporizată înainte de a genera un nou unghi. În timpul rotației, diferența dintre unghiul curent (turretRotationAngle) și cel țintă (turretRotationTarget) este calculată. Dacă diferența este semnificativă (> 0.1 grade), turela continuă să se rotească la o viteză constantă, ajustată cu timpul delta. Când se atinge unghiul țintă, rotația se oprește, iar animația intră în faza de așteptare.

După pauza de 15 secunde, un nou unghi țintă este generat aleator între -90 și +90 de grade față de poziția curentă, iar rotația reîncepe.

```
if (isRotating) {
    float deltaRotation = turretRotationTarget - turretRotationAngle;
    if (fabs(deltaRotation) > 0.1f) {
        turretRotationAngle += (deltaRotation > 0.0f ? 1 : -1) * turretRotationSpeed * 0.016f;
    }
    else {
        turretRotationAngle = turretRotationTarget;
        isRotating = false;
        rotationWaitTime = 15.0f;
    }
}
if (rotationWaitTime > 0.0f) {
    rotationWaitTime -= 0.016f;
    if (rotationWaitTime <= 0.0f) {
        turretRotationStart = turretRotationAngle;
        turretRotationTarget = turretRotationStart + (rand() % 181 - 90);
        isRotating = true;
    }
}
```

- **Efectul de ceata:**

Efectul de ceață adaugă realism scenei, estompând obiectele pe măsură ce se îndepărtează de cameră. Funcția computeFog() calculează factorul de ceață pentru fiecare fragment, determinând gradul de amestecare a culorii cu ceața. **Densitatea ceții** este controlată de fogDensity, influențând cât de rapid devin obiectele invizibile. **Distanța fragmentului** față de cameră este calculată cu length(fPosEye) și utilizată într-o formulă exponențială pentru a atenua vizibilitatea treptat. Factorul final este limitat între 0.0 și 1.0 cu clamp().

```
float computeFog() {
    float fogDensity = 0.005f;
    float fragmentDistance = length(fPosEye);
    float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));
    return clamp(fogFactor, 0.0f, 1.0f);
}
```

- **Flashlight si floodlights:**

computeFlashlightEffect – Iluminarea dinamică a lanternei

computeFlashlightEffect calculează iluminarea dinamică a lanternei (spotlight), care urmează camera. Poziția și direcția luminii sunt transformate în spațiul camerei, iar unghiul (theta) dintre direcția luminii și vectorul spre fragment este calculat. Dacă fragmentul este în conul de lumină (definit de cutOff și outerCutOff), intensitatea este interpolată linear.

Iluminarea ambientă, difuză și specular este adăugată fragmentului, variind în funcție de poziția sa în conul de lumină.

```
void computeFlashlightEffect() {

    if(!flashlight.activated){
        return;
    }
    vec3 flashlightPosEye = (view * vec4(flashlight.position, 1.0f)).xyz;
    vec3 flashlightDirEye = normalize(mat3(view) * flashlight.direction);

    vec3 normalEye = normalize(fNormal);
    vec3 lightDirNflashlight = normalize(flashlightPosEye - fPosEye.xyz);
    float theta = dot(lightDirNflashlight, -flashlightDirEye);

    float epsilon = flashlight.cutOff - flashlight.outerCutOff;
    float intensity = clamp((theta - flashlight.outerCutOff) / epsilon, 0.0f, 1.0f);

    if (theta > flashlight.outerCutOff) {
        vec3 reflection = reflect(-lightDirNflashlight, normalEye);
        float specCoeff = pow(max(dot(viewDirN, reflection), 0.0f), shininess);

        ambient += flashlight.color * ambientStrength;
        diffuse += intensity * max(dot(normalEye, lightDirNflashlight), 0.0f) * flashlight.color;
        specular += intensity * specCoeff * flashlight.color * specularStrength;
    }
}
```

computeSpotLight – Reflectoare fixe (floodlights)

Funcția calculează iluminarea oferită de reflectoarele fixe din scenă. Poziția și direcția reflectoarelor sunt transformate în spațiul camerei pentru a determina relația cu fragmentul. Unghiul dintre direcția luminii și fragment este calculat, iar intensitatea variază în funcție de

poziția fragmentului în conul de lumină. Dacă fragmentul este în con, sunt aplicate componentele de iluminare ambientală, difuză și specular, creând un efect focalizat.

```
void computeSpotLight(vec3 lightPosWorld, vec3 lightDirection, vec3 lightColor, float cutoff, float outerCutoff) {
    vec3 lightPosEye = (view * vec4(lightPosWorld, 1.0f)).xyz;
    vec3 lightDirEye = normalize(mat3(view) * lightDirection);

    vec3 normalEye = normalize(fNormal);
    vec3 lightDirToFrag = normalize(lightPosEye - fPosEye.xyz);

    float theta = dot(lightDirToFrag, -lightDirEye);

    float epsilon = cutoff - outerCutoff;
    float intensity = clamp((theta - outerCutoff) / epsilon, 0.0f, 1.0f);

    if (theta > outerCutoff) {
        ambient += lightColor * ambientStrength;

        float diff = max(dot(normalEye, lightDirToFrag), 0.0f);
        diffuse += intensity * diff * lightColor;

        vec3 reflection = reflect(-lightDirToFrag, normalEye);
        float specCoeff = pow(max(dot(viewDirN, reflection), 0.0f), shininess);
        specular += intensity * specCoeff * lightColor * specularStrength;
    }
}
```

- **Efectul de lumina punctiforma:**

1. Poziția luminii în spațiul camerei:
    - Poziția luminii este transformată cu matricea de vizualizare (view), esențială pentru direcția luminii și distanța față de fragment.
2. Direcția și reflexia:
    - lightDirN este vectorul normalizat dintre poziția luminii și fragment.
3. Reflexia luminii este calculată cu funcția reflect pentru componenta specular.
4. Atenuarea luminii:
    - Intensitatea scade cu distanța, calculată printr-o formulă ce combină componente constantă, liniară și cuadratică pentru un efect realist.
5. Iluminarea ambientală, difuză și specular:
    - Ambient: Simulează lumina ambientală, afectată de atenuare.
    - Diffuse: Depinde de unghiul dintre direcția luminii și normală, iluminând zonele expuse.
    - Specular: Reflexiile sunt influențate de unghiul dintre direcția privirii și reflexie, cu intensitate controlată de un coeficient specular.

```
void computePointLightComponents(vec3 lightPosWorld, vec3 lightColor, float constant, float linear, float quadratic) {

    vec3 lightPosEye = (view * vec4(lightPosWorld, 1.0f)).xyz;

    vec3 normalEye = normalize(fNormal);
    vec3 lightDirN = normalize(lightPosEye - fPosEye.xyz);
    vec3 reflection = reflect(-lightDirN, normalEye);

    float specCoeff = pow(max(dot(viewDirN, reflection), 0.0f), shininess);
    float dist = length(lightPosEye - fPosEye.xyz);
    float attenuation = 1.0 / (constant + linear * dist + quadratic * (dist * dist));

    ambient += attenuation * ambientStrength * lightColor;
    diffuse += attenuation * max(dot(normalEye, lightDirN), 0.0f) * lightColor;
    specular += attenuation * specularStrength * specCoeff * lightColor;
}
```

- **Animatia camerei:**

  Funcția updateCameraAnimation gestionează animația camerei prin două etape: **deplasarea** și **rotirea**. În prima etapă, camera se mișcă lin între punctele din vectorul points folosind interpolarea bazată pe timpul scurs (t), actualizând în mod dinamic poziția și direcția pentru un efect realist. După parcurgerea tuturor punctelor, începe a doua etapă: o rotație treptată a camerei pe axa sa, până la 180 de grade, cu un unghi incrementat în funcție de durata rotației. La final, animația se oprește, iar direcția camerei este setată definitive.

```
void updateCameraAnimation() {
    if (isAnimating) {
        if (!isRotatingCamera) {
            float t = (glfwGetTime() - animationStartTime) / movementDuration;
            if (t > 1.0f) {
                t = 0.0f;
                currentPoint++;
                if (currentPoint >= sizeof(points) / sizeof(points[0]) - 1) {
                    isRotatingCamera = true;
                    rotationStartTime = glfwGetTime();
                    totalRotationYawSoFar = 0.0f;
                }
                animationStartTime = glfwGetTime();
            }
            if (currentPoint < sizeof(points) / sizeof(points[0]) - 1) {
                currentPos = points[currentPoint] + t * (points[currentPoint + 1] - points[currentPoint]);
                myCamera.setPosition(currentPos);
                direction = glm::normalize(points[currentPoint + 1] - currentPos);
                myCamera.setDirection(direction);
            }
        }
        if (isRotatingCamera) {
            float elapsedRotationTime = glfwGetTime() - rotationStartTime;
            if (totalRotationYawSoFar < 180.0f) {
                float rotationStepYaw = (180.0f * elapsedRotationTime / rotationDuration) - totalRotationYawSoFar;
                myCamera.rotate(0.0f, rotationStepYaw);
                direction = glm::normalize(glm::vec3(
                    cos(glm::radians(myCamera.getYaw())) * cos(glm::radians(45.0f)),
                    0.0f, sin(glm::radians(myCamera.getYaw())) * cos(glm::radians(45.0f))));
                myCamera.setDirection(direction);
                totalRotationYawSoFar += rotationStepYaw;
            }
            else {
                direction = glm::normalize(glm::vec3(
                    cos(glm::radians(myCamera.getYaw())) * cos(glm::radians(45.0f)),
                    sin(glm::radians(-10.0f)), sin(glm::radians(myCamera.getYaw())) * cos(glm::radians(45.0f))));
                myCamera.setDirection(direction);
                isRotatingCamera = false;
                isAnimating = false;
            }
        }
    }
}
```

- **Obiecte rezultate prin fragment discard:**

Fragment discard este o tehnică pentru eliminarea fragmentelor care nu trebuie afișate, optimizând performanța și contribuind la efecte vizuale realiste. Culoarea fragmentului este preluată din textură utilizând coordonatele de textură, iar valoarea de transparență (alpha) este verificată. Dacă alpha este sub un prag (de exemplu, 0.1), instrucțiunea discard este folosită pentru a elimina fragmentul.

```
    vec4 colorFromTexture = texture(diffuseTexture, fTexCoords);

    if (colorFromTexture.a < 0.1)
        discard;
```

- **Calcularea umbrei:**

Codul determină umbra unui fragment folosind Shadow Mapping. Coordonatele fragmentului în spațiul luminii (fragPosLightSpace) sunt normalizate prin împărțirea la componenta w, apoi scalate în intervalul [0, 1]. Adâncimea din shadowMap este comparată cu adâncimea curentă a fragmentului. Dacă adâncimea curentă depășește cea din shadowMap (cu o mică marjă de toleranță), fragmentul este considerat umbrit, iar funcția returnează 1.0. În caz contrar, returnează 0.0, indicând iluminarea directă.

```
float computeShadow() {
    vec3 normalizedCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    normalizedCoords = normalizedCoords * 0.5 + 0.5;

    float closestDepth = texture(shadowMap, normalizedCoords.xy).r;
    float currentDepth = normalizedCoords.z;

    if (normalizedCoords.z > 1.0f)
        return 0.0f;

    float shadow = currentDepth > closestDepth + 0.005 ? 1.0 : 0.0;
    return shadow;
}
```

```
float shadow = computeShadow();

vec3 color = min((ambient + (1.0f - shadow) * diffuse) + (1.0f - shadow) * specular, 1.0f);
```

3.2 Structuri de date custom folosite

Structura PointLight:

Structura PointLight este utilizată pentru a defini proprietățile unui light point în scenă. Această structură conține:

- **position**: Un glm::vec3 care reprezintă poziția luminii în coordonate mondiale.
- **color**: Un glm::vec3 care definește culoarea luminii.
- **constant, linear, quadratic**: Aceste valori de tip float sunt utilizate pentru calculele de atenuare ale luminii. Ele definesc cum scade intensitatea luminii pe măsură ce distanța față de sursa de lumină crește, iar termenul quadratic modelează decăderea naturală a luminii pe distanță.

Structura SpotLight:

Structura SpotLight este utilizată pentru a defini proprietățile unui spot light. Spot light-ul are următoarele proprietăți:

- **position**: Un glm::vec3 care reprezintă poziția spot light-ului în spațiul mondial.
- **direction**: Un glm::vec3 care definește direcția în care este orientat spot light-ul.
- **color**: Un glm::vec3 pentru culoarea spot light-ului.
- **cutOff și outerCutOff**: Aceste valori, exprimate în termeni de cosinus, definesc intervalul unghiular al spot light-ului. CutOff reprezintă regiunea cea mai luminoasă a spot light-ului, iar outerCutOff reprezintă limita exterioară a conului de lumină, dincolo de care intensitatea luminii scade treptat.

Structura Flashlight:

Structura Flashlight este o variație a spot light-ului utilizată pentru a simula o lanternă atașată la cameră. Ea conține:

- **position**: Un glm::vec3 care reprezintă poziția lanternei, sincronizată cu poziția camerei.
- **direction**: Un glm::vec3 care indică direcția în care este orientată lanterna, sincronizată cu direcția camerei.
- **color**: Un glm::vec3 pentru culoarea lanternei.

- **cutOff și outerCutOff**: Aceste valori definesc limitele unghiulare ale conului de lumină al lanternei, la fel ca în cazul spot light-ului.
- **activated**: O variabilă booleană pentru a controla dacă lanterna este activată sau nu.

```
struct PointLight {
    vec3 position;
    float constant;
    float linear;
    float quadratic;
    vec3 color;
};

struct Flashlight {
    vec3 position;
    vec3 direction;
    float cutOff;
    float outerCutOff;
    vec3 color;
    bool activated;
};

struct SpotLight {
    vec3 position;
    vec3 direction;
    float cutOff;
    float outerCutOff;
    vec3 color;
};
```

4. Prezentarea interfeței grafice utilizator / manual de utilizare

Acest manual descrie controalele disponibile pentru mișcarea camerei, manipularea luminii și diverse alte funcționalități ale scenei 3D. Folosind tastatura, utilizatorul poate interacționa cu scena în mod dinamic și poate modifica diverse aspecte ale iluminării și animației.

Mișcarea Camerei

W - Mergi înainte (spre direcția camerei).

S - Mergi înapoi (în direcția opusă camerei).

A - Mergi la stânga.

D - Mergi la dreapta.

Z - Rotește camera la stânga (modifică unghiul pe axa Y).

C - Rotește camera la dreapta (modifică unghiul pe axa Y).

Rotația Camerei și a Turretului

V - Rotește turela cu 15 grade. Dacă unghiul depășește 360 de grade, se resetează la 0.

T - Afișează direcția curentă a camerei în consolă (împreună cu valorile pe axele X, Y și Z).

Controlul Iluminării

X – Face switch intre lumina directionala si punctiforma.

J - Rotește lumina spre stânga cu un unghi de 1 grad.

L - Rotește lumina spre dreapta cu un unghi de 1 grad.

Moduri de Redare

1 - Setează modul de redare pe Punct (doar puncte).

2 - Setează modul de redare pe WIREFRAME (doar contururi).

3 - Setează modul de redare pe SOLID (model solid complet).

Animație și Mișcare Automatizată

Space - Pornește animația camerei și a obiectelor. Mișcarea va începe de la primul punct definit și va urma un traseu presetat.

Funcționalități Avansate

F - Activează sau dezactivează lanterna (flashlight) atașată camerei. Lanterna urmează mișcările camerei și luminează direcția în care camera se uită.

5. Concluzii si dezvoltari ulterioare

În urma realizării acestui proiect, am învățat să creez o scenă interactivă, să modelez obiecte și să aplic principii avansate de grafică 3D. Am dobândit o înțelegere mai aprofundată a conceptelor explicate la curs și am aplicat aceste cunoștințe pentru a crea o aplicație funcțională.

Proiectul a fost un mod creativ și eficient de a mă familiariza cu OpenGL, un API grafic puternic pentru dezvoltarea de aplicații 2D și 3D. De asemenea, am folosit software-ul grafic 3D Blender, în care am modelat scena inițială, incluzând obiectele și luminile ce compun mediul interactiv. Această experiență m-a ajutat să înțeleg și să aplic principiile grafice, cum ar fi manipularea obiectelor 3D, iluminarea, texturarea și aplicarea de efecte vizuale.

Dezvoltări ulterioare

Proiectul poate fi extins și îmbunătățit cu mai multe funcționalități interesante, inclusiv:

- Lasarea nopții: Implementarea unui ciclu de zi/noapte, unde condițiile de iluminare și texturile pot varia în funcție de ora din zi.
- Nava extraterestră: Crearea unei animații în care o navă extraterestră apare din spațiu pe ecran, adăugând un element surpriză și un efect vizual deosebit.
- Collision Detection: Implementarea unui sistem de detecție a coliziunilor pentru a permite interacțiuni corecte între obiectele din scenă.
- Ploaie acidă: Crearea unui efect vizual de ploaie acidă care interacționează cu mediul și obiectele, sporind realismul scenei.
- Îmbunătățirea calității umbrei: Optimizarea tehnicilor de generare a umbrelor pentru a le face mai realiste, de exemplu prin utilizarea de tehnici precum shadow mapping cu rezoluție mai mare și algoritmi mai avansați.

6. Referințe

- https://sketchfab.com/feed
- https://learnopengl.com/Lighting/Light-casters
- Laboratoarele de pe moodle
- https://www.youtube.com/watch?v=crOfyWiWxmc