Universitatea Tehnică din Cluj-Napoca

Facultatea de Automatică și Calculatoare

Ruinele dintr-un deșert

Implementat în OpenGL

Nume: Hojda Matei-Andrei

Grupa: 30238

Cuprins

1. Specificarea subiectului

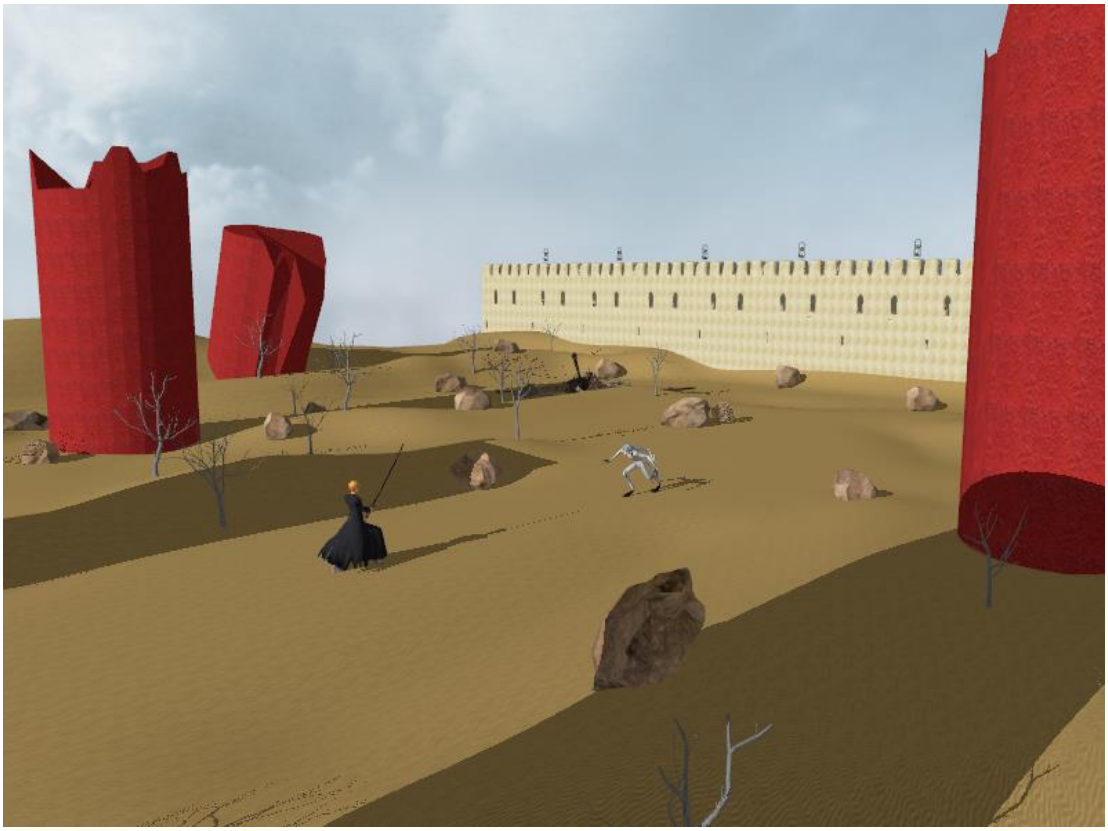
Scopul acestui proiect este să proiecteze și să implementeze o scenă în C++ folosind OpenGL. Blender este folosit, de asemenea, pentru poziționarea, modelarea și texturizarea obiectelor care vor fi ulterior importate în proiectul principal. Utilizatorul are, de asemenea, posibilitatea să interacționeze cu scena, să se deplaseze în jurul ei și să declanșeze animații.

2. Scenariu

2.1 Scena și descrierea obiectelor

Scena reprezintă ruinele aflate intr-un deșert, din care au rămas doar stâlpii de susținere ai intrării principale și un zid enorm în spate. Pe zid se află 11 lanterne vechi folosite atunci pentru iluminarea pereților pe timpul nopții. În centrul scenei se află două personaje care par să aibă niște conturi de reglat. În apropierea zidului se află un tufiș de deșert rotund care servește la animația in scenă. În total se află în scenă 8 stâlpi de susținere, 51 de copaci și 31 de pietre, toate de forme și mărimi diferite.

O vizualizare generală a scenei este prezentă mai jos:



2.2 Funcţionalităţi

Menționat ca în prima secțiune, ultilizatorul poate să și interacționeze cu scena, nu doar să o observe. Funcționalitățile includ deplasarea prin scenă cu ajutorul tastaturii și a mouse-ului , activarea animației tufei, mișcarea soarelui(schimbarea luminii și a modului în care umbrele sunt afișate), observarea scenei în modurile solid, wireframe și punct, trecerea de la zi la noapte, afișarea hărții de adâncimi și prezentarea scenei cu o cameră cinematică.

3. Detaliile implementării

3.1 Funcţii şi algoritmi speciali

3.1.1 Soluţii posibile

3.1.1.1 Mișcarea camerei

Mișcarea camerei a fost implementată folosind tutorialul de pe site-ul “Learn OpenGL”. Cu ajutorului funcțiilor de mișcare și rotație, ne este posibilă plimbarea și uitarea prin scenă prin intermediul mouse-ului și a tastaturii, iar la apăsarea unei taste se va realiza o animație de prezentare a scenei.

Camera.cpp

```cpp
//update the camera internal parameters following a camera move event
void Camera::move(MOVE_DIRECTION direction, float speed) {
    //TODO
    if (direction == gps::MOVE_FORWARD) {
        this->cameraPosition += speed * this->cameraFrontDirection;
    }

    if (direction == gps::MOVE_BACKWARD) {
        this->cameraPosition -= speed * this->cameraFrontDirection;
    }

    if (direction == gps::MOVE_LEFT) {
        this->cameraPosition -= speed * this->cameraRightDirection;
    }

    if (direction == gps::MOVE_RIGHT) {
        this->cameraPosition += speed * this->cameraRightDirection;
    }

    if (direction == gps::MOVE_UP) {
        this->cameraPosition += speed * this->cameraUpDirection;
    }

    if (direction == gps::MOVE_DOWN) {
        this->cameraPosition -= speed * this->cameraUpDirection;
    }

    this->cameraTarget = this->cameraPosition + this->cameraFrontDirection;
}

//update the camera internal parameters following a camera rotate event
//yaw - camera rotation around the y axis
//pitch - camera rotation around the x axis
void Camera::rotate(float pitch, float yaw) {
    //TODO
    glm::vec3 directie;
    directie.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
    directie.y = sin(glm::radians(pitch));
    directie.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));

    this->cameraFrontDirection = glm::normalize(directie);
    this->cameraTarget = this->cameraPosition + this->cameraFrontDirection;
    this->cameraRightDirection = glm::normalize(glm::cross(this->cameraFrontDirection, this->cameraUpDirection));
}
```

Main.cpp (pentru mișcarea camerei cu mouse-ul)

```cpp
void mouseCallback(GLFWwindow* window, double xpos, double ypos) {
    if (fistMouseMove) {
        fistMouseMove = false;
        xLast = xpos;
        yLast = ypos;
    }

    float xOffset = xpos - xLast;
    float yOffset = yLast - ypos;

    xLast = xpos;
    yLast = ypos;

    xOffset *= cameraSensitivity;
    yOffset *= cameraSensitivity;

    yaw += xOffset;
    pitch += yOffset;

    if (pitch > 89.0f) {
        pitch = 89.0f;
    }
    if (pitch < -89.0f) {
        pitch = -89.0f;
    }

    myCamera.rotate(pitch, yaw);
}
```

Mai.cpp (pentru mișcarea camerei cu tastatura)

```
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_SPACE]) {
    myCamera.move(gps::MOVE_UP, cameraSpeed);
}

if (pressedKeys[GLFW_KEY_LEFT_CONTROL]) {
    myCamera.move(gps::MOVE_DOWN, cameraSpeed);
}
```

Prezentarea scenei repezintă o rotație mai lungă în jurul axei Z și una mai scurtă în jurul axei X, care rămâne fixată în centrul scenei pentru a oferi o vizualizare 360 a scenei.

Main.cpp (keyboardCallback)

```
if (key == GLFW_KEY_Q && action == GLFW_PRESS) {
    animatiePrezentareScena = !animatiePrezentareScena;
    if (!animatiePrezentareScena) {
        myCamera = gps::Camera(glm::vec3(xAnim, 46.55f, zAnim), glm::vec3(0.0f, 20.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    }
}
```

Main.cpp (prezentareScena)

```
void prezentareScena() {
    if (animatiePrezentareScena) {
        if (startAnimatie) {
            startAnimatie = false;
            xAnim = 0.0f;
            zAnim = -200.50f;
            myCamera = gps::Camera(glm::vec3(xAnim, 46.55f, zAnim), glm::vec3(0.0f, 20.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
            timeStamp = 0;
        }

        xAnim = sin(timeStamp / 100.0f) * 100;
        zAnim = cos(timeStamp / 100.0f) * 50;
        timeStamp++;
        myCamera = gps::Camera(glm::vec3(xAnim, 46.55f, zAnim), glm::vec3(0.0f, 20.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
    }
}
```

3.1.1.2 Sursele de lumină

În scena noastră sunt prezente în total 12 surse de lumină de două tipuri diferite: o lumină direcțională și 11 lumini punctiforme.

Lumina direcțională este folosită pentru a simula lumina provenind de la soare. Este utilizată iluminarea Phong, care combină componente de lumină ambientală, difuză și specular. Aceasta a fost implementată urmărind tutorialul de pe site-ul “Learn OpenGL”.

Lumina punctiformă este implementată cu resursele de pe site-ul “Learn OpenGL”. Luminiile punctiforme sunt localizate la flăcarile lanternelor ce apar pe timpul nopții.

3.1.1.3 Umbrele

Toate obiectele prezente în scenă au umbre generate față de lumina direcțională. După cum s-a menționat în capitolele anterioare, sursa de lumină direcțională (soarele) poate fi rotită în jurul scenei de către utilizator folosind tastatura, schimbând astfel și umbrele. O prezentare generală a umbrei (împreună cu lumina direcțională) poate fi observată în imaginea de mai jos:

Umbrele au fost implementate folosind sursele din laboratorului.

3.1.1.4 Texturile

Obiectele au fost descărcate de pe site-uri cu modele 3D gratis (site-urile sunt specificate în referință). Acestea au fost poziționate în scenă și texturate folosind programul Blender. Un exemplu interesant de texturare este texturarea hainelor unuia dintre personajul nostru principal, care este prezentată în imaginea de mai jos:

3.1.1.5 Ceaţă

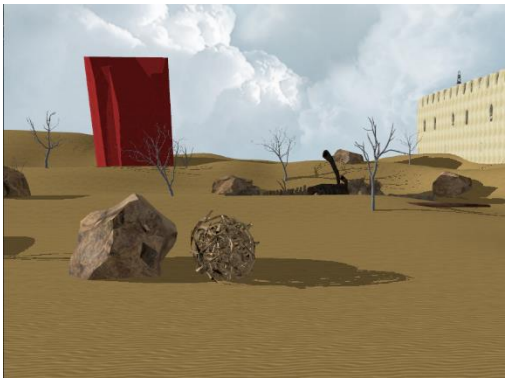
Ceața a fost implementată folosind resursele din laborator, cu o intensitate redusă pentru a permite observarea efectului pe care o oferă. Scena noastră prezintă un efect de ceață, adaptat la scena deșertului, cu o nuntă de galben, prezentată în imaginea de mai jos:

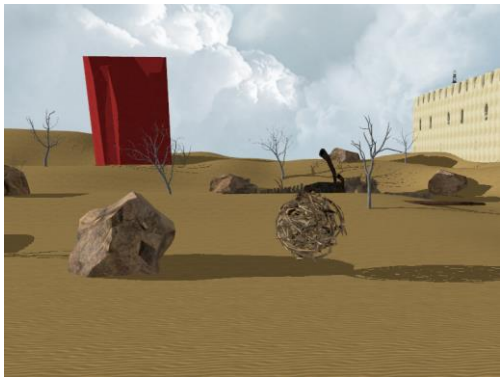
3.1.1.6 Animaţii

În scena noastră există o animație.

Prima animație este mișcarea tufei de la poziția ei inițială până la zid și înapoi, folosind translațiiile și rotațiile necesare. Pentru a realiza această animație, el este transalatat în origine și

rotit pentru fiecare frame. Când tasta de animție este apăsată, tufișul va fi adus în origine, rotit, iar apoi translatat înapoi cu un offset care se modifică dinamic.





Main.cpp (keyboardCallback)

```
if (key == GLFW_KEY_1 && action == GLFW_PRESS && !animateTumbleweed) {
    animateTumbleweed = true;
}
```

Main.cpp (funcția ce realizează animația tufei)

```
void tumbleWeedAnimation(gps::Shader shader) {
    if (animateTumbleweed) {
        model = glm::translate(glm::mat4(1.0f), 1.0f * (tumbleWeedPos + tumbleWeedOffset));
        model = glm::rotate(model, glm::radians(tumbleWeedRotAngle), glm::vec3(0.0f, 0.0f, 1.0f));
        model = glm::translate(model, 1.0f * (-tumbleWeedPos));
        glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE, glm::value_ptr(model));

        if (!animateBackTumbleWeed) {
            tumbleWeedRotAngle += 0.7f;
            tumbleWeedOffset.x -= 0.04f;
        }
        else {
            tumbleWeedRotAngle -= 0.7f;
            tumbleWeedOffset.x += 0.04f;
        }

        if (!animateBackTumbleWeed) {
            tumbleWeedOffset.y += tumbleWeedYdir;
        }
        else {
            if (tumbleWeedOffset.y > 0.0f) {
                tumbleWeedOffset.y += tumbleWeedYdir;
            }
            else {
                tumbleWeedOffset.y = 0.0f;
            }
        }

        if (tumbleWeedOffset.y >= 5.0f) {
            tumbleWeedYdir = -0.05f;
        }
        if (tumbleWeedPos.x + tumbleWeedOffset.x >= -170.0f) {
            if (tumbleWeedOffset.y <= 0.0f) {
                tumbleWeedYdir = 0.05f;
            }
        }
        else {
            if (tumbleWeedOffset.y <= 2.0f) {
                tumbleWeedYdir = -0.05f;
                animateBackTumbleWeed = true;
            }
        }

        if (animateBackTumbleWeed && tumbleWeedOffset.x >= 0.0f) {
            animateTumbleweed = false;
            animateBackTumbleWeed = false;
            tumbleWeedOffset.x = 0.0f;
        }
    }
}
```

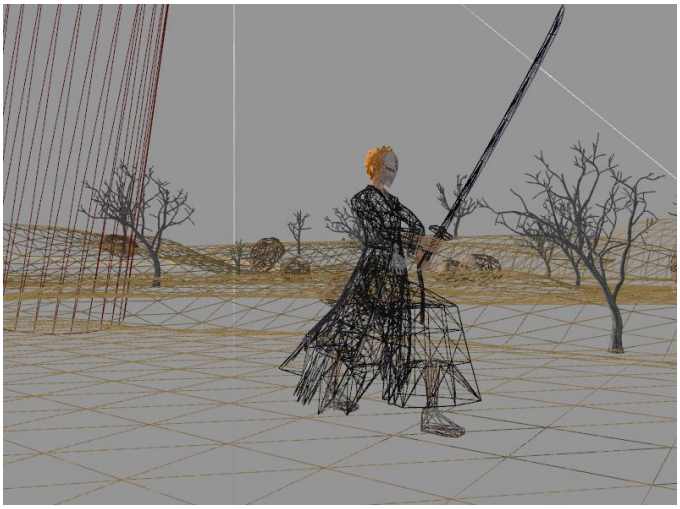
Main.cpp (drawObjects)

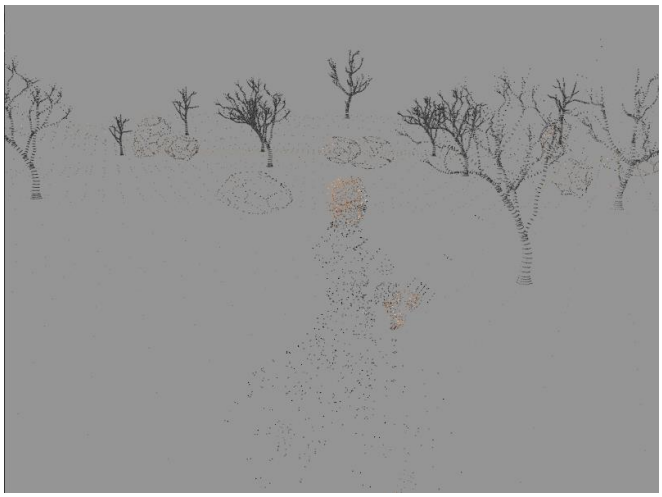
```
tumbleWeedAnimation(shader);

tumbleWeed.Draw(shader);
```

A doua animație reprezintă prezentarea scenei automat, descrisă în secțiunea 3.1.1.1.

3.1.1.7 Vizualizarea scenei în modul solid și wireframe

Obiectele din scenă pot fi randate fie în modul solid, fie în modul wireframe, această alegere aparținând utilizatorului și fiind realizată prin intermediul tastaturii. Vizualizarea scenei în aceste moduri este prezentată mai jos:





3.1.2 Motivația abordării alese

Abordarea prezentată în secțiunea anterioară a fost motivată de dorința de a crea o scenă cât mai realistă care să permită totuși adăugarea unor personaje fantastice, fără să elimine acest efect de realism. Animația tufei are rolul de a adăuga o dinamică scenei.

3.2 Modelul grafic

Obiectele prezente în scenă folosesc reprezentarea poligonală. După cum afirmă cursul al patrulea al cursului de Procesare Grafică, această reprezentare are următoarele avantaje și probleme:

Avantaje:

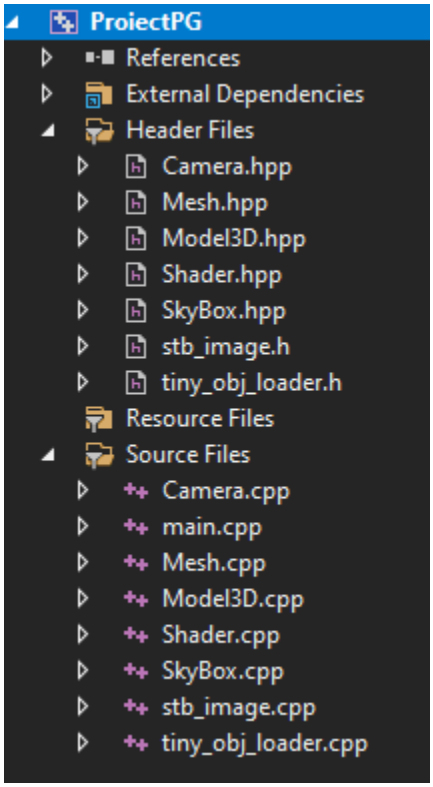
- Omniprezență în grafică pe calculator
- Simplu și direct
- Folosit ca formă intermediară pentru multe alte structuri de date

Probleme:

- Aproximarea suprafețelor curbate
- Precizia depinde extrem de numărul de poligoane; de exemplu, în cazul zoom-ului sau scalării
- Aplicarea texturilor
- Suprafețe netede
- Algoritmi de umbrire

Reprezentarea poligonală poate fi observată utilizând vizualizarea wireframe a scenei. Să luăm, de exemplu, obiectul unui rață. Dacă ne apropiem suficient de mult, putem vedea poligoanele care formează obiectul.

3.3 Ierarhia claselor



4. Manualul de utilizare

După cum am menționat anterior, utilizatorul poate să și interacționeze cu scena folosind mouse-ului și tastatura. Fiecare funcționalitate și comenzile respective sunt prezentate mai jos.

Mouse:

- rotirea camerei

Tastatură:

- W/A/S/D: mișcarea camerei prin scenă
- L_CTRL: coborârea camerei în scenă
- SPACE: ridicarea camerei în scenă
- J/L: mișcarea luminii direcționale (soarelui) pe axa Z
- 1: redarea animației tufei
- M: afișarea/ascunderea hărții de adâncime
- N: trecerea din zi în noapte și vice-versa
- T/Y/U: vizualizarea scenei în modurile solid, wireframe și punct

- F: mărirea intensității cetii
- G: scăderea intensității cetii
- Q: animația de prezentare a scenei

4. Concluzii și dezvoltări viitoare

Scopul acestui proiect de a crea unei scene a unor ruine în deșert cu care utilizatorul să poată interacționa a fost atins, iar aplicația este gata să fie folosită. Dezvoltările viitoare care ar putea fi realizate la acest moment ar fi coliziunea cu obiectele, intrarea în perspectiva personajelor în modul 3rd person și, cel mai important, animația celor două personaje pentru a face scena mult mai realistă.

6. Referințe

- https://learnopengl.com/Lighting/Multiple-lights
- https://ogldev.org/www/tutorial20/tutorial20.html
- https://learnopengl.com/Getting-started/Camera
- https://www.youtube.com/watch?v=kCCsko29pv0
- https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping
- https://moodle.cs.utcluj.ro/pluginfile.php/186200/mod_resource/content/9/Laborator%206.pdf
- https://moodle.cs.utcluj.ro/pluginfile.php/186209/mod_resource/content/3/Laboratory_work_7_RO.pdf
- https://moodle.cs.utcluj.ro/pluginfile.php/186215/mod_resource/content/2/Laboratory_work_8_RO.pdf
- https://moodle.cs.utcluj.ro/pluginfile.php/186230/mod_resource/content/4/Laboratory%20work%209%20RO.pdf
- https://moodle.cs.utcluj.ro/pluginfile.php/186247/mod_resource/content/2/Laboratory_work_11.pdf