KAROL DZIARKOWSKI, WOJCIECH FERDA | INFORMATYKA TECHNICZNA GR.2

Sprawozdanie z projektu "Symulator Układu Słonecznego"

1. Wprowadzenie

Celem projektu było stworzenie symulatora Układu Słonecznego z wykorzystaniem bibliotek OpenGL oraz GLM. Projekt umożliwia użytkownikowi nawigację po wirtualnym układzie planetarnym oraz podziwianie modeli planet i ich ruchu.

2. Opis implementacji

2.1. Kamera

Plik camera.cpp zawiera implementację klasy Camera, która odpowiada za obsługę kamery w przestrzeni 3D. Kamera jest inicjalizowana z określoną pozycją, kierunkiem patrzenia oraz parametrami ruchu i czułości myszy. Metody klasy Camera pozwalają na przetwarzanie ruchu klawiatury i myszy oraz aktualizację wektorów kamery, co umożliwia płynne poruszanie się po scenie.

```
Camera::Camera(glm::vec3 position, glm::vec3 up, GLfloat yaw, GLfloat pitch)
    : Front(glm::vec3(0.0f, 0.0f, -1.0f)),
    MovementSpeed(SPEED),
    MouseSensitivity(SENSITIVTY),
    Zoom(ZOOM)
{
    this->Position = position;
    this->WorldUp = up;
    this->Yaw = yaw;
    this->Pitch = pitch;
    this->updateCameraVectors();
}
```

2.2. Główna funkcja

Plik main.cpp zawiera główną funkcję programu, która inicjalizuje okno aplikacji za pomocą klasy OpenGLWindow i uruchamia główną pętlę renderowania.

```
int main(int argc, char* argv[])
{
    OpenGLWindow Window;

    Window.createWindow(1080, 720, "Solar System Simulator ", false);
    Window.runApp();

    return 0;
}
```

2.3. Klasa OpenGLWindow

Plik openGLwindow.cpp implementuje klasę OpenGLWindow, która zarządza tworzeniem okna, inicjalizacją sceny, obsługą zdarzeń i renderowaniem.

Tworzenie okna: Funkcja createWindow tworzy okno aplikacji i ustawia niezbędne parametry dla GLFW i GLEW.

```
bool OpenGLWindow::createWindow(int width, int height, std::string windowTitle, bool showfullScreen)
{
    // Init GLFW
    if (!glfwInit())
    {
        std::cerr << "ERROR::GLFW initialisation failed!" << std::endl;
        glfwTerminate();
        return false;
    }

    // Set all the required options for GLFW
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

    this->width = width;
    this->height = height;

    this->windowTitle = windowTitle;

    fullScreen = showfullScreen;
    if (showfullScreen)
    {
        GLFWmonitor* monitor = glfwGetPrimaryMonitor();
        const GLFWvidmode* vidMode = glfwGetVideoMode(monitor);
        if (vidMode != nullptr)
        {
            window = glfwCreateWindow(vidMode->width, vidMode->height, windowTitle.c_str(), monitor, nullptr);
        }
    }
    else
    {
        this->window = glfwCreateWindow(width, height, windowTitle.c_str(), nullptr, nullptr);
    }



    if (window == nullptr)
    {
        std::cerr << "ERROR::Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return false;
    }
```

```
glfwMakeContextCurrent(this->window);

//// Set the required callback functions
glfwSetKeyCallback(this->window, this->key_callback);
glfwSetScrollCallback(window, this->scroll_callback);
glfwSetCursorPosCallback(window, this->mouse_callback);

glewExperimental = GL_TRUE;
// Initialize GLEW to setup the OpenGL Function pointers
if (glewInit() != GLEW_OK)
{
    std::cerr << "ERROR::Failed to initialize GLEW" << std::endl;
    return false;
}

glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

// Define the viewport dimensions
glfwGetFramebufferSize(window, &this->width, &this->height);
glViewport(0, 0, this->width, this->height);
return true;
```

Inicjalizacja sceny: Funkcja initializeScene wczytuje shadery, tekstury i tworzy bufory dla sfer reprezentujących planety oraz inne elementy sceny.

```
void OpenGLWindow::initializeScene()
{
    // Setup OpenGL options
    glEnable(GL_DEPTH_TEST);

    Shader vShader, fShader;
    vShader.loadShaderFromFile("..\\Shaders\\planet.vert", GL_VERTEX_SHADER);
    fShader.loadShaderFromFile("..\\Shaders\\planet.frag", GL_FRAGMENT_SHADER);

    if (!vShader.isShaderLoaded() || !fShader.isShaderLoaded())
    {
        return;
    }

    planetsSheProg.createProgram();
    planetsSheProg.addShaderToProgram(vShader);
    planetsSheProg.addShaderToProgram(fShader);
    vShader.deleteShader();
    fShader.deleteShader();

    if (!planetsSheProg.linkProgram())
    {
        return;
    }

    Shader vSunShader, fSunShader;
    vSunShader.loadShaderFromFile("..\\Shaders\\sun.vert", GL_VERTEX_SHADER);
    fSunShader.loadShaderFromFile("..\\Shaders\\sun.frag", GL_FRAGMENT_SHADER);

    if (!vSunShader.isShaderLoaded() || !fSunShader.isShaderLoaded())
    {
        std::cout << "==========" << std::endl;

        return;
    }
```

```
    sunSheProg.createProgram();
    sunSheProg.addShaderToProgram(vSunShader);
    sunSheProg.addShaderToProgram(fSunShader);
    vSunShader.deleteShader();
    fSunShader.deleteShader();

    if (!sunSheProg.linkProgram())
    {
        return;
    }


    // Create Buffers
    sun.generateBuffers();
    mercury.generateBuffers();
    venus.generateBuffers();
    earth.generateBuffers();
    moon.generateBuffers();
    mars.generateBuffers();
    jupiter.generateBuffers();
    saturn.generateBuffers();
    uranus.generateBuffers();
    neptune.generateBuffers();
    skyBox.generateBuffers();

    // Load textures
    sunTexture.fastLoad();
    mercuryTexture.fastLoad();
    venusTexture.fastLoad();
    earthTexture.fastLoad();
    moonTexture.fastLoad();
    marsTexture.fastLoad();
    jupiterTexture.fastLoad();
    saturnTexture.fastLoad();
    uranusTexture.fastLoad();
    neptuneTexture.fastLoad();
    skyBoxTexture.fastLoad();
}
```

Renderowanie sceny: Funkcja renderScene odpowiada za rysowanie sceny. Kamera jest ustawiana w zależności od wybranej planety, a następnie rysowane są wszystkie elementy sceny.

```cpp
void OpenGLWindow::renderScene()
{
    glfwPollEvents();
    doMovement();
    showFPS();
    onWindowSizeChanged();

    // Clear the buffer
    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glm::vec3 offset = glm::vec3(1.0f, 1.0f, 1.0f);
    switch (PlanetViewe)
    {
    case(Planets::Mercury):
        camera.setPosition(mercury.getOrigin() + offset);
        break;
    case(Planets::Venus):
        camera.setPosition(venus.getOrigin() + offset);
        break;
    case(Planets::Earth):
        camera.setPosition(earth.getOrigin() + offset);
        break;
    case(Planets::Moon):
        camera.setPosition(moon.getOrigin() + offset);
        break;
    case(Planets::Mars):
        camera.setPosition(mars.getOrigin() + offset);
        break;
    case(Planets::Jupiter):
        camera.setPosition(jupiter.getOrigin() + offset);
        break;
    case(Planets::Saturn):
        camera.setPosition(saturn.getOrigin() + offset);
        break;
    case(Planets::Uranus):
        camera.setPosition(uranus.getOrigin() + offset);
        break;
    case(Planets::Neptune):
        camera.setPosition(neptune.getOrigin() + offset);
        break;
    case(Planets::Space):
    default:
        break;
    }

    sunSheProg.useProgram();
    GLint PVM_p = glGetUniformLocation(sunSheProg.getShaderProgramID(), "PVM");
```

```cpp
    // SKY BOX
    glBindVertexArray(skyBox.getVAO());
    glm::mat4 PVM = getProjectionMatrix();
    glUniformMatrix4fv(PVM_p, 1, GL_FALSE, glm::value_ptr(PVM));
    skyBoxTexture.useTexture();
    skyBox.draw();
    glBindVertexArray(0);

    // SUN
    glBindVertexArray(sun.getVAO());
    sun.addRotation();
    PVM = getProjectionMatrix() * camera.getViewMatrix() * sun.getModelMatrix();
    glUniformMatrix4fv(PVM_p, 1, GL_FALSE, glm::value_ptr(PVM));
    sunTexture.useTexture();
    sun.draw();
    glBindVertexArray(0);

    planetsSheProg.useProgram();
    GLint PV_p = glGetUniformLocation(planetsSheProg.getShaderProgramID(), "PV");
    GLint model_p = glGetUniformLocation(planetsSheProg.getShaderProgramID(), "model");

    // MERCURY
    glBindVertexArray(mercury.getVAO());
    mercury.addRotation(2.5f, 0.8, 25);
    glm::mat4 PV = getProjectionMatrix() * camera.getViewMatrix();
    glUniformMatrix4fv(PV_p, 1, GL_FALSE, glm::value_ptr(PV));
    glUniformMatrix4fv(model_p, 1, GL_FALSE, glm::value_ptr(mercury.getModelMatrix()));
    mercuryTexture.useTexture();
    mercury.draw();
    glBindVertexArray(0);
```

```
// VENUS
glBindVertexArray(venus.getVAO());
venus.addRotation(3.5f, 0.9, 28);
PV = getProjectionMatrix() * camera.getViewMatrix();
glUniformMatrix4fv(PV_p, 1, GL_FALSE, glm::value_ptr(PV));
glUniformMatrix4fv(model_p, 1, GL_FALSE, glm::value_ptr(venus.getModelMatrix()));
venusTexture.useTexture();
venus.draw();
glBindVertexArray(0);

// EARTH
glBindVertexArray(earth.getVAO());
earth.addRotation(4.5f, 0.8, 30);
PV = getProjectionMatrix() * camera.getViewMatrix();
glUniformMatrix4fv(PV_p, 1, GL_FALSE, glm::value_ptr(PV));
glUniformMatrix4fv(model_p, 1, GL_FALSE, glm::value_ptr(earth.getModelMatrix()));
earthTexture.useTexture();
earth.draw();
glBindVertexArray(0);

// MOON
glBindVertexArray(moon.getVAO());
moon.addRotation(0.5f, 4, 30, earth.getOrigin());
PV = getProjectionMatrix() * camera.getViewMatrix();
glUniformMatrix4fv(PV_p, 1, GL_FALSE, glm::value_ptr(PV));
glUniformMatrix4fv(model_p, 1, GL_FALSE, glm::value_ptr(moon.getModelMatrix()));
moonTexture.useTexture();
moon.draw();
glBindVertexArray(0);
```

```
// MARS
glBindVertexArray(mars.getVAO());
mars.addRotation(5.5f, 0.7, 34);
PV = getProjectionMatrix() * camera.getViewMatrix();
glUniformMatrix4fv(PV_p, 1, GL_FALSE, glm::value_ptr(PV));
glUniformMatrix4fv(model_p, 1, GL_FALSE, glm::value_ptr(mars.getModelMatrix()));
marsTexture.useTexture();
mars.draw();
glBindVertexArray(0);

// JUPITER
glBindVertexArray(jupiter.getVAO());
jupiter.addRotation(7.0f, 0.6, 40);
PV = getProjectionMatrix() * camera.getViewMatrix();
glUniformMatrix4fv(PV_p, 1, GL_FALSE, glm::value_ptr(PV));
glUniformMatrix4fv(model_p, 1, GL_FALSE, glm::value_ptr(jupiter.getModelMatrix()));
jupiterTexture.useTexture();
jupiter.draw();
glBindVertexArray(0);

// SATURN
glBindVertexArray(saturn.getVAO());
saturn.addRotation(8.5f, 0.5, 35);
PV = getProjectionMatrix() * camera.getViewMatrix();
glUniformMatrix4fv(PV_p, 1, GL_FALSE, glm::value_ptr(PV));
glUniformMatrix4fv(model_p, 1, GL_FALSE, glm::value_ptr(saturn.getModelMatrix()));
saturnTexture.useTexture();
saturn.draw();
glBindVertexArray(0);
```

```
    // URANUS
    glBindVertexArray(uranus.getVAO());
    uranus.addRotation(9.5f, 0.7, 40);
    PV = getProjectionMatrix() * camera.getViewMatrix();
    glUniformMatrix4fv(PV_p, 1, GL_FALSE, glm::value_ptr(PV));
    glUniformMatrix4fv(model_p, 1, GL_FALSE, glm::value_ptr(uranus.getModelMatrix()));
    uranusTexture.useTexture();
    uranus.draw();
    glBindVertexArray(0);

    // NEPTUNE
    glBindVertexArray(neptune.getVAO());
    neptune.addRotation(10.0f, 0.85f, 40);
    PV = getProjectionMatrix() * camera.getViewMatrix();
    glUniformMatrix4fv(PV_p, 1, GL_FALSE, glm::value_ptr(PV));
    glUniformMatrix4fv(model_p, 1, GL_FALSE, glm::value_ptr(neptune.getModelMatrix()));
    neptuneTexture.useTexture();
    neptune.draw();
    glBindVertexArray(0);


    // Swap the screen buffers
    glfwSwapBuffers(this->window);
}
```

2.4. Obsługa zdarzeń

Klawiatura: Funkcja keyCallback odpowiada za obsługę zdarzeń klawiatury, takich jak zamykanie okna czy zmiana widoku kamery.

```cpp
void OpenGLWindow::key_callback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
    std::cout << key << std::endl;
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, GL_TRUE);

    if (key >= 0 && key < 1024)
    {
        if (action == GLFW_PRESS)
            keys[key] = true;
        else if (action == GLFW_RELEASE)
            keys[key] = false;
    }
```

```cpp
void OpenGLWindow::doMovement()
{
    // Camera controls
    if (keys[GLFW_KEY_W])
        camera.processKeyboard(FORWARD, deltaTime);
    if (keys[GLFW_KEY_S])
        camera.processKeyboard(BACKWARD, deltaTime);
    if (keys[GLFW_KEY_A])
        camera.processKeyboard(LEFT, deltaTime);
    if (keys[GLFW_KEY_D])
        camera.processKeyboard(RIGHT, deltaTime);

    if (keys[GLFW_KEY_1])
        PlanetViewe = Planets::Mercury;
    if (keys[GLFW_KEY_2])
        PlanetViewe = Planets::Venus;
    if (keys[GLFW_KEY_3])
        PlanetViewe = Planets::Earth;
    if (keys[GLFW_KEY_4])
        PlanetViewe = Planets::Moon;
    if (keys[GLFW_KEY_5])
        PlanetViewe = Planets::Mars;
    if (keys[GLFW_KEY_6])
        PlanetViewe = Planets::Jupiter;
    if (keys[GLFW_KEY_7])
        PlanetViewe = Planets::Saturn;
    if (keys[GLFW_KEY_8])
        PlanetViewe = Planets::Uranus;
    if (keys[GLFW_KEY_9])
        PlanetViewe = Planets::Neptune;
    if (keys[GLFW_KEY_0])
        PlanetViewe = Planets::Space;
}
```

Mysz: Funkcja mouseCallback obsługuje ruch myszy, co pozwala na obracanie kamery wokół punktu obserwacji.

```cpp
void OpenGLWindow::mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    GLfloat xoffset = xpos - lastX;
    GLfloat yoffset = lastY - ypos;

    lastX = xpos;
    lastY = ypos;

    camera.processMouseMovement(xoffset, yoffset);
}
```

3. Wnioski

Nasz projekt "Układu Słonecznego" to program umożliwiający wizualizację planet i ich ruchu w przestrzeni 3D. Wykorzystuje technologie takie jak OpenGL, GLFW, GLEW oraz GLSL do renderowania grafiki i zarządzania wejściami użytkownika. Użytkownik może poruszać się po scenie, przybliżać i oddalać widok, oraz oglądać realistycznie odwzorowane planety i słońce dzięki zastosowaniu tekstur.

4. Bibliografia

Dokumentacja OpenGL

Dokumentacja GLM

Dokumentacja GLFW

Dokumentacja GLEW