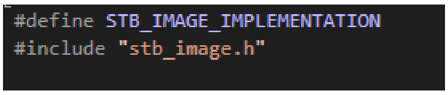
Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

SPRAWOZDANIE

Lab nr 6

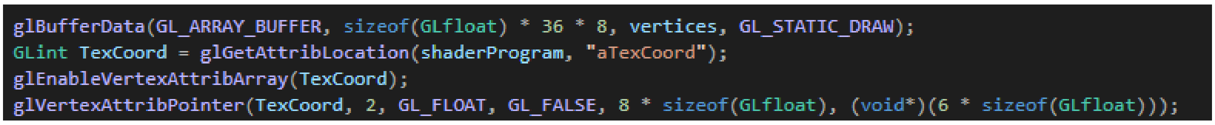
Celem ćwiczenia było zapoznanie się z funkcjami realizującymi wczytywanie tekstur oraz mechanizmami nakładania tekstur na wielokąty.

1. Dodanie pliku nagłówkowego ”stb_image.h”.



```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
```

2. Zmiana liczby na 8, na każdy wierzchołek przypada 8 wartości.

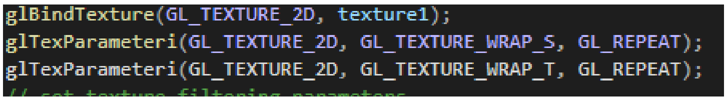


```
glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * 36 * 8, vertices, GL_STATIC_DRAW);
GLint TexCoord = glGetAttribLocation(shaderProgram, "aTexCoord");
glEnableVertexAttribArray(TexCoord);
glVertexAttribPointer(TexCoord, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 * sizeof(GLfloat)));
```

3. Generowanie tekstury.

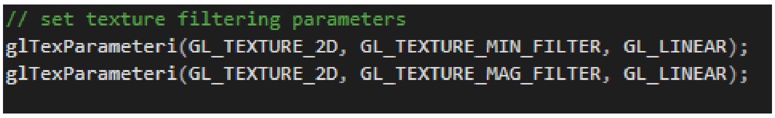
glGenTextures(1, &texture1);

4. Ustawienie textury jako bieżącej.



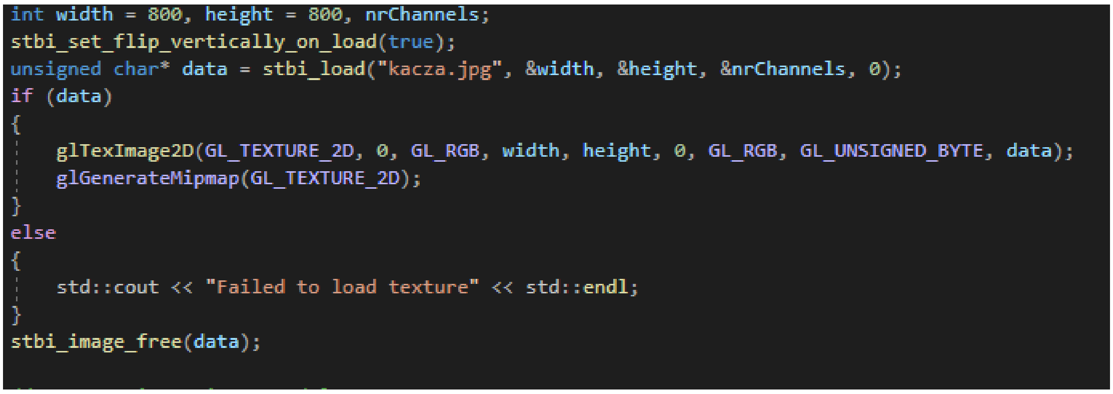
```
glBindTexture(GL_TEXTURE_2D, texture1);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

5. Ustawienie parametrów filtrowania textury.



```
// set texture filtering parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

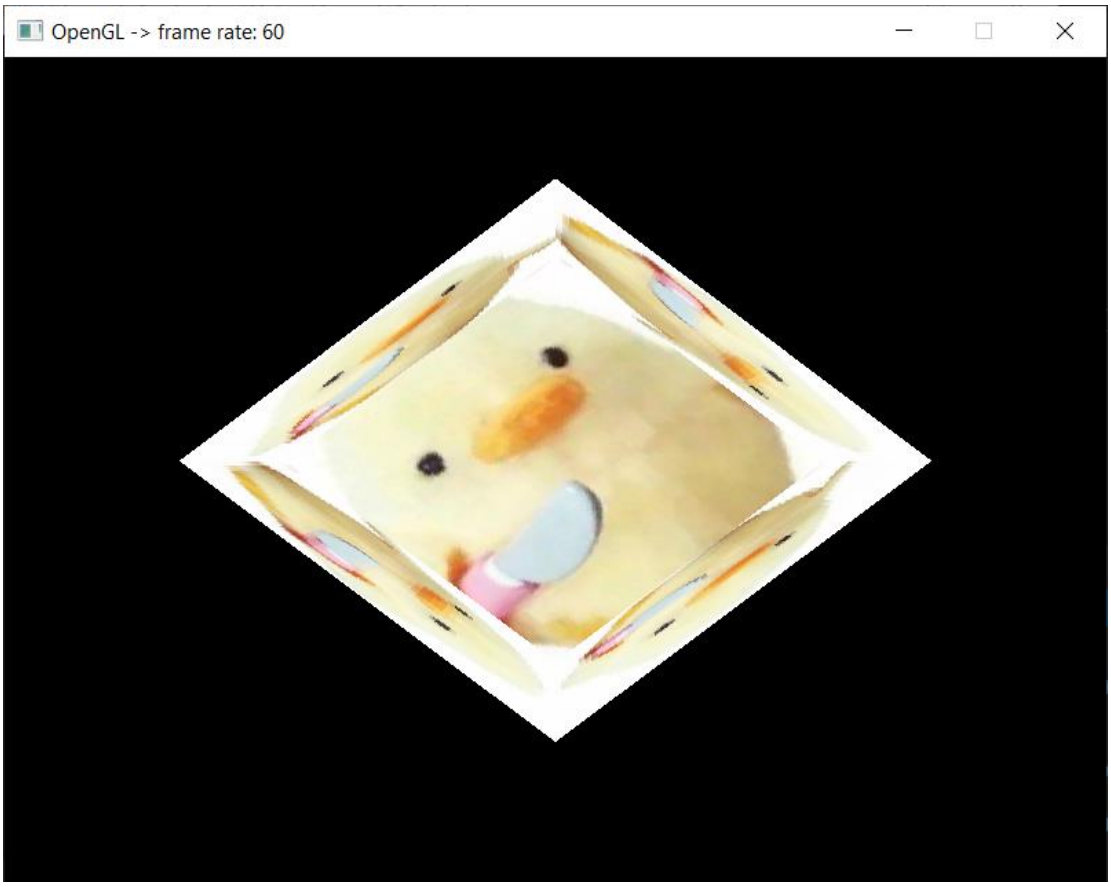
6. Załadowanie obrazu, stworzenie textury i wygenerowanie mipmap.

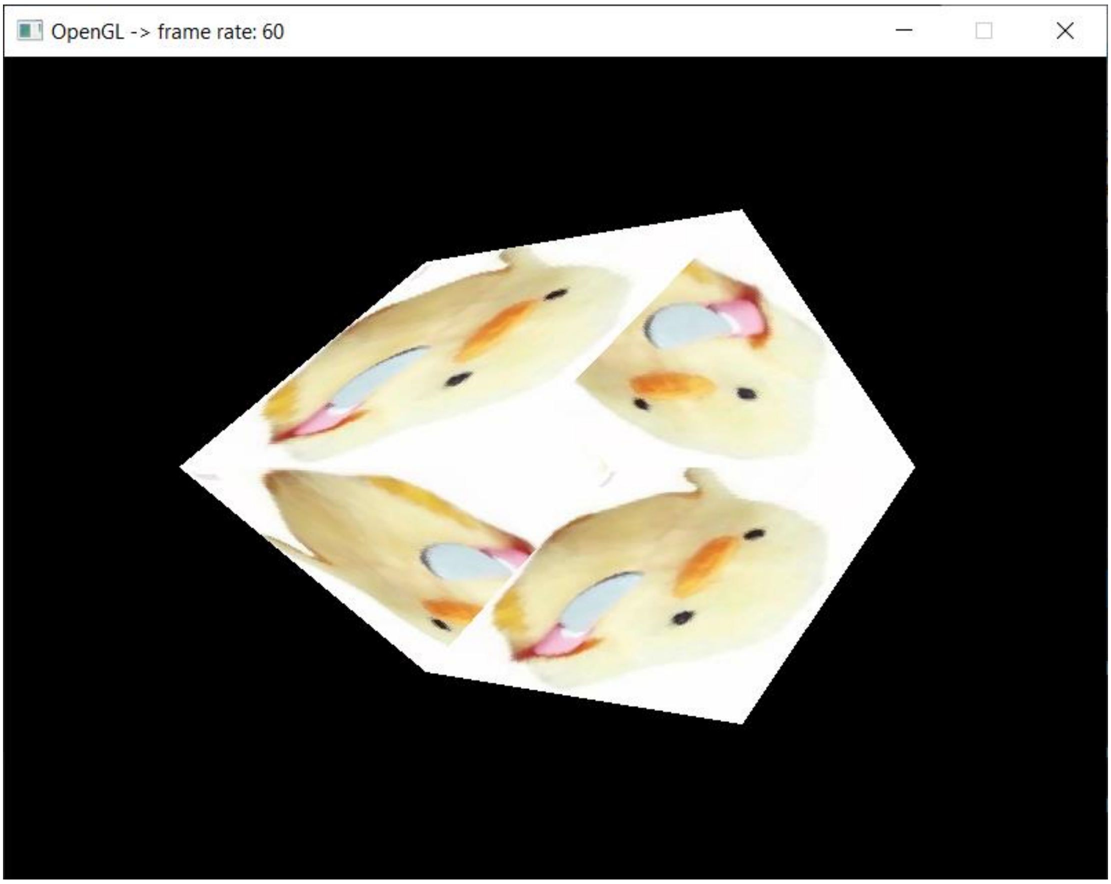


```
int width = 800, height = 800, nrChannels;
stbi_set_flip_vertically_on_load(true);
unsigned char* data = stbi_load("kacza.jpg", &width, &height, &nrChannels, 0);
if (data)
{
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
    glGenerateMipmap(GL_TEXTURE_2D);
}
else
{
    std::cout << "Failed to load texture" << std::endl;
}
stbi_image_free(data);
```

Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

WYNIKI PROGRAMU:





Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

KOD:

```
// Naglowki
#define _USE_MATH_DEFINES
#include "stdafx.h"
#include <GL/glew.h>
#include <SFML/Window.hpp>4
#include <iostream>
#include <math.h>
#include <random>
#include <Windows.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <SFML/System/Time.hpp>
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

const GLchar* vertexSource = R"glsl(
#version 150 core
in vec2 aTexCoord;
in vec3 position;
in vec3 color;
out vec2 TexCoord;
out vec3 Color;
uniform mat4 model;
uniform mat4 view;
uniform mat4 proj;
void main()
{
Color = color;
TexCoord = aTexCoord;
gl_Position = proj * view * model * vec4(position, 1.0);
}
)glsl";

const GLchar* fragmentSource = R"glsl(
#version 150 core
in vec2 TexCoord;
uniform sampler2D texture1;
in vec3 Color;
out vec4 outColor;
void main()
{
outColor = texture(texture1, TexCoord);
}
)glsl";

// Kamera
glm::vec3 cameraPos = glm::vec3(0.0f, 0.0f, 3.0f);
glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);
glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

void Camera3D(GLint uniView) {
        glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
        glUniformMatrix4fv(uniView, 1, GL_FALSE, glm::value_ptr(view));
}

double yaw = -90;
double pitch = 0;
double lastX = 0, lastY = 0;
double sensitivity = 0.1;
glm::vec3 front;
sf::Vector2i localPosition;

void ustawKamereMysz(GLint uniView, float time, sf::Window& window) {
        localPosition = sf::Mouse::getPosition(window);

        double xoffset = localPosition.x - lastX;
        double yoffset = localPosition.y - lastY;
        lastX = localPosition.x;
        lastY = localPosition.y;

        xoffset *= sensitivity;
        yoffset *= sensitivity;
        yaw += xoffset;
        pitch -= yoffset;

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

        front.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
        front.y = sin(glm::radians(pitch));
        front.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
        cameraFront = glm::normalize(front);
}
```

```
GLfloat* toCartesian(float p, float z, int num) {
    GLfloat* arr = new GLfloat[6 * num];
    float change = 360.0f / static_cast <GLfloat>(num);
    float degree = 0;
    srand(time(NULL));
    for (int i = 0; i < num; i++) {
        arr[0 + i * 6] = p * static_cast <GLfloat> (cos(degree *
            M_PI / 180));
        arr[1 + i * 6] = p * static_cast <GLfloat> (sin(degree *
            M_PI / 180));
        arr[2 + i * 6] = z;
        arr[3 + i * 6] = rand() % 11 / 10.0;
        arr[4 + i * 6] = rand() % 11 / 10.0;
        arr[5 + i * 6] = rand() % 11 / 10.0;
        degree += change;
    }
    return arr;
}

using namespace std;

int main() {

    sf::ContextSettings settings;
    settings.depthBits = 24;
    settings.stencilBits = 8;

    // Okno renderingu
    sf::Window window(sf::VideoMode(800, 600, 32), "OpenGL",
        sf::Style::Titlebar | sf::Style::Close, settings);

    int frames = 60;
    //window.setFramerateLimit(frames);

    window.setTitle("OpenGL -> frame rate: " + to_string(frames));
    // Inicjalizacja GLEW
    glewExperimental = GL_TRUE;
    glewInit();

    // Utworzenie VAO (Vertex Array Object)
    GLuint vao;
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);
```

```
// Utworzenie VBO (Vertex Buffer Object)
// i skopiowanie do niego danych wierzcholkowych
GLuint vbo;
glGenBuffers(1, &vbo);
GLfloat vertices[] = {
-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f, 1.0f,
0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f, 1.0f,
-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
0.5f, 0.5f, 0.5f, 1.0f, 1.0f, 0.0f, 1.0f, 1.0f,
0.5f, 0.5f, 0.5f, 1.0f, 1.0f, 0.0f, 1.0f, 1.0f,
-0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
-0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
-0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f, 0.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
-0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f, 0.0f,
0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,
0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
0.5f, -0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f, 0.0f,
0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 1.0f, 0.0f,
0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
-0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f
};

glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

```
// Utworzenie i skompilowanie shadera wierzcholkow
GLuint vertexShader =
    glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexSource, NULL);
glCompileShader(vertexShader);

// Utworzenie i skompilowanie shadera fragmentow
GLuint fragmentShader =
    glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentSource, NULL);
glCompileShader(fragmentShader);

// Sprawdzanie czy shadery sie dobrze zaladowaly
GLint check1;
GLint check2;
glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &check1);
glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &check2);

cout << "Compilation vertexShader: ";
if (check1 == GL_TRUE)
    cout << "works" << endl;
else {
    cout << "error" << endl;
    GLint infoLength;
    glGetShaderiv(vertexShader, GL_INFO_LOG_LENGTH,
        &infoLength);
    GLchar* buffer = new GLchar[infoLength];
    GLsizei bufferSize;
    glGetShaderInfoLog(vertexShader, infoLength, &bufferSize,
        buffer);
    cout << buffer << endl;
    delete[] buffer;
}

cout << "Compilation fragmentShader: ";
if (check2 == GL_TRUE)
    cout << "works" << endl;
else {
    cout << "error" << endl;
    GLint infoLength2;
    glGetShaderiv(fragmentShader, GL_INFO_LOG_LENGTH,
        &infoLength2);
    GLchar* buffer2 = new GLchar[infoLength2];
    GLsizei bufferSize2;
```

```
        glGetShaderInfoLog(fragmentShader, infoLength2,
            &bufferSize2, buffer2);
        cout << buffer2 << endl;
        delete[] buffer2;
    }

    // Zlinkowanie obu shaderow w jeden wspolny program
    GLuint shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glBindFragDataLocation(shaderProgram, 0, "outColor");
    glLinkProgram(shaderProgram);
    glUseProgram(shaderProgram);

    // Specifikacja formatu danych wierzcholkowych
    GLint posAttrib = glGetAttribLocation(shaderProgram, "position");
    glEnableVertexAttribArray(posAttrib);
    glVertexAttribPointer(posAttrib, 3, GL_FLOAT, GL_FALSE, 8 *
        sizeof(GLfloat), 0);
    GLint colAttrib = glGetAttribLocation(shaderProgram, "color");
    glEnableVertexAttribArray(colAttrib);
    glVertexAttribPointer(colAttrib, 3, GL_FLOAT, GL_FALSE, 8 *
        sizeof(GLfloat), (void*)(3 * sizeof(GLfloat)));
    GLenum primitive = GL_TRIANGLES;

    glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * 36 * 8, vertices, GL_STATIC_DRAW);
    GLint TexCoord = glGetAttribLocation(shaderProgram, "aTexCoord");
    glEnableVertexAttribArray(TexCoord);
    glVertexAttribPointer(TexCoord, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(6 *
sizeof(GLfloat)));
    unsigned int texture1;
    glGenTextures(1, &texture1);
    glBindTexture(GL_TEXTURE_2D, texture1);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    int width = 1024, height = 576, nrChannels;
    stbi_set_flip_vertically_on_load(true);
    unsigned char* data = stbi_load("kacza.jpg", &width, &height, &nrChannels, 0);
    if (data)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
```

```
{
    std::cout << "Failed to load texture" << std::endl;
}
stbi_image_free(data);

// Utworzenie macierzy modelu
glm::mat4 model = glm::mat4(1.0f);
model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f,
    0.0f, 1.0f));
GLint uniTrans = glGetUniformLocation(shaderProgram, "model");
glUniformMatrix4fv(uniTrans, 1, GL_FALSE, glm::value_ptr(model));

glm::mat4 view;
view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
GLint uniView = glGetUniformLocation(shaderProgram, "view");
glUniformMatrix4fv(uniView, 1, GL_FALSE, glm::value_ptr(view));
glm::mat4 proj = glm::perspective(glm::radians(45.0f), 800.0f /
    800.0f, 0.06f, 100.0f);
GLint uniProj = glGetUniformLocation(shaderProgram, "proj");
glUniformMatrix4fv(uniProj, 1, GL_FALSE, glm::value_ptr(proj));
float cameraSpeed = 0.05;
int pos;
window.setMouseCursorGrabbed(true);
window.setMouseCursorVisible(false);
sf::Clock clock;
sf::Time time;

bool running = true;
while (running) {
    time = clock.getElapsedTime();
    clock.restart();

    sf::Event windowEvent;
    while (window.pollEvent(windowEvent)) {
        switch (windowEvent.type) {
        case sf::Event::Closed:
            running = false;
            break;
        case sf::Event::KeyPressed:
            switch (windowEvent.key.code)
            {
            case sf::Keyboard::Up:
                cameraPos += cameraSpeed * cameraFront;
                break;
                case sf::Keyboard::Down:
                    cameraPos -= cameraSpeed * cameraFront;
                    break;
                case sf::Keyboard::Left:
                    cameraPos -= glm::normalize(glm::cross(cameraFront,
cameraUp)) * cameraSpeed;
                    break;
                case sf::Keyboard::Right:
                    cameraPos += glm::normalize(glm::cross(cameraFront,
cameraUp)) * cameraSpeed;
                    break;
                case sf::Keyboard::Escape: // close application
                    window.close();
                    break;
                case sf::Keyboard::Num1:
                    primitive = GL_POINTS;
                    break;
                case sf::Keyboard::Num2:
                    primitive = GL_LINES;
                    break;
                case sf::Keyboard::Num3:
                    primitive = GL_LINE_STRIP;
                    break;
                case sf::Keyboard::Num4:
                    primitive = GL_LINE_LOOP;
                    break;
                case sf::Keyboard::Num5:
                    primitive = GL_TRIANGLES;
                    break;
                case sf::Keyboard::Num6:
                    primitive = GL_TRIANGLE_STRIP;
                    break;
                case sf::Keyboard::Num7:
                    primitive = GL_TRIANGLE_FAN;
                    break;
                case sf::Keyboard::Num8:
                    primitive = GL_QUADS;
                    break;
                case sf::Keyboard::Num9:
                    primitive = GL_QUAD_STRIP;
                    break;
                case sf::Keyboard::Num0:
                    primitive = GL_POLYGON;
                    break;
                }
            case sf::Event::MouseMoved:
                ustawKamereMysz(uniView, time.asMicroseconds(), window);
                break;
            }
        }
        Camera3D(uniView);

        // Nadanie scenie koloru czarnego
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        //glDrawArrays(primitive, 0, num);
        glDrawArrays(GL_TRIANGLES, 0, 36);

        // Wymiana buforow tylni/przedni
        window.display();
        //Sleep(100);
        cameraSpeed = 0.000099f * time.asMicroseconds();
    }
    // Kasowanie programu i czyszczenie buforoww
    glDeleteProgram(shaderProgram);
    glDeleteShader(fragmentShader);
    glDeleteShader(vertexShader);
    glDeleteBuffers(1, &vbo);
    glDeleteVertexArrays(1, &vao);
    // Zamkniecie okna renderingu
    window.close();
    return 0;
}
```