Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

SPRAWOZDANIE

Lab nr 7

Celem ćwiczenia było zapoznanie się z funkcjami realizującymi oświetlenie sceny. Ustawienie właściwości materiału oraz właściwości źródeł światła.

1. Zaimportowanie obiektu VBo wraz z informacjami o wwktorach normalnych.
2. Przekazanie informacji o wektorach normalnych określając specyfikacje formatu danych wierzchołkowych.

```
GLint NorAttrib = glGetAttribLocation(shaderProgram, "aNormal");
glEnableVertexAttribArray(NorAttrib);
glVertexAttribPointer(NorAttrib, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (void*)(3 *sizeof(GLfloat)));
glm::vec3 lightPos(1.2f, 1.0f, 2.0f);
```

3. Ustalenie położenia światła.

```
glm::vec3 lightPos(1.2f, 1.0f, 2.0f);
GLint uniLightPos = glGetUniformLocation(shaderProgram, "lightPos");
glUniform3fv(uniLightPos, 1, &lightPos[0]);
```

4. Odebranie informacji o wektorach normalnych i przekazanie do fragmentu shadera.
5. Zadeklarowanie zmiennej do przekazania pozycji fragmentów.

```
//dodano nowe zmienne poniżej
in vec3 aNormal;
out vec3 Normal;
out vec3 FragPos;
```

6. Odebranie informacji o wektorach normalnych z vertex shadera.

```
in vec3 Normal;
in vec3 FragPos;
```

7. Dodanie mocy i koloru światła otoczenia poprzez przemnożenie aktualnego koloru fragmentu przez składową światła.
8. Dodanie składowej koloru światła rozproszonego.

```
vec4 ambient = ambientStrength * vec4(1.0,1.0,1.0,1.0);
vec3 difflightColor = vec3(1.0,1.0,1.0);
```

9. Wyznaczenie wektora kierunku między źródłem światła, a pozycją fragmentu i znormalizowanie zarówno normalnego, jak i wynikowego wektora kierunkowego.

```
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
```

10. Ustalenie wpływu składowej rozproszonej światła na bieżacy fragment, biorąc iloczyn skalarny normy i wektora lightDir.

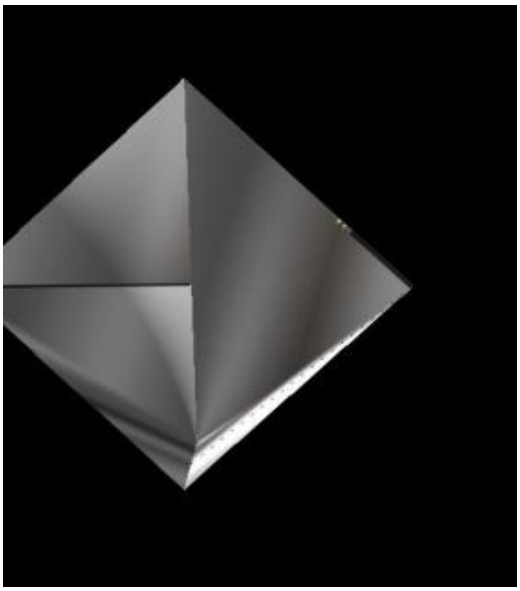
```
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * difflightColor;
```

11. Dodanie dwóch składowych otoczenia i światła rozproszonego i pomnożenie przez bieżacy kolor fragmentu.

```
//zmieniono ustawienie wyjściowe koloru
outColor = (ambient+vec4(diffuse,1.0)) * texture(texture1, TexCoord);
```

Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

WYNIK PROGRAMU:



KOD(problem z dodaniem koloru):

```
// Naglowki
#include <iostream>
#include <GL/glew.h>
#include <SFML/Window.hpp>
#include <ctime>
#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"
#define SCREEN_WIDTH 800
#define SCREEN_HEIGHT 800
// Kody shaderów
const GLchar* vertexSource = R"glsl(
#version 150 core
in vec3 position;
in vec3 color;
in vec2 aTexCoord; // współrzędne tekstur jako atrybuty wierzchołka
out vec2 TexCoord;
out vec3 Color;
uniform mat4 model; //dodanie zmiennej odpowiadającej modelowi
uniform mat4 view; //dodanie zmiennej odpowiadającej widokowi
uniform mat4 proj; //dodanie zmiennej odpowiadającej projekcji
```

Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

```
//dodano nowe zmienne poniżej
in vec3 aNormal;
out vec3 Normal;
out vec3 FragPos;
void main(){
//Color = color;
TexCoord = aTexCoord;
Normal = aNormal;
gl_Position = proj*view*model*vec4(position, 1.0); //okreslanie pozycji
FragPos = vec3(model * vec4(position, 1.0));
}
)glsl";
const GLchar* fragmentSource = R"glsl(
#version 150 core
//in vec3 Color;
in vec2 TexCoord;
uniform sampler2D texture1; //przekazanie tekstury
out vec4 outColor;
// dodano poniżej zmienne
in vec3 Normal;
in vec3 FragPos;
uniform vec3 lightPos;
uniform float ambientStrength ; // ambient
//vec3 ambientlightColor = vec3(1.0,1.0,1.0);
//vec4 ambient = ambientStrength * vec4(ambientlightColor,1.0);
vec4 ambient = ambientStrength * vec4(1.0,1.0,1.0,1.0);
vec3 difflightColor = vec3(1.0,1.0,1.0);
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * difflightColor;
void main()
{
//outColor = vec4(Color, 1.0);
//outColor=texture(texture1, TexCoord); // ustawienie tekstury
//zmieniono ustawienie wyjściowe koloru
outColor = (ambient+vec4(diffuse,1.0)) * texture(texture1, TexCoord);
}
)glsl";
GLfloat* CreateVert(GLfloat* vert, int punkty)
{
        delete[] vert; // usunięcie starej tablicy
        GLfloat doublePi = 2 * 3.1415; //360 stopni
        GLfloat z = 0; // współrzędna z = 0, ponieważ rysujemy w 2d
```

```
	GLfloat r = 0.7; // długość promienia [0-1] 1 to cały okno
	//std::unique_ptr<GLfloat[]> vertices(new GLfloat[punkty * 6]);
	GLfloat* vertices = new GLfloat[punkty * 6];
	//GLfloat vertices[punkty * 6];// tablica przechowująca wierzchołki wraz z rgb
	// * 6 dlatego że potrzebujemy współrzędnych x, y, z oraz kolorów rgb
	// x = rcos(theta)
	// y = rsin(theta)
	// z = z
	srand((unsigned)time(0)); //żeby za każdym razem nie losowo tych samych kolorów
po 1 włączeniu
	for (size_t i = 0; i < punkty; i++)
	{
		vertices[(i * 6)] = r * cos(i * doublePi / (punkty - 1)); //współrzędna x
		// dziele 2pi przez ilość wierzchołków oraz razy i aby osiągnać pełen zakres 0-
360
		vertices[(i * 6) + 1] = r * sin(i * doublePi / (punkty - 1));//współrzędna y
		// dziele 2pi przez ilość wierzchołków oraz razy i aby osiągnać pełen zakres 0-
360
		vertices[(i * 6) + 2] = z;////współrzędna z
		vertices[(i * 6) + 3] = static_cast <float> (rand()) / (static_cast <float>
(RAND_MAX));
		// randomowy float z zakresu 0-1 ponieważ taki jest zakres rgb. (tutaj Red)
		vertices[(i * 6) + 4] = static_cast <float> (rand()) / (static_cast <float>
(RAND_MAX));
		// randomowy float z zakresu 0-1 ponieważ taki jest zakres rgb. (tutaj Blue)
		vertices[(i * 6) + 5] = static_cast <float> (rand()) / (static_cast <float>
(RAND_MAX));
		// randomowy float z zakresu 0-1 ponieważ taki jest zakres rgb. (tutaj Green)
	}
	return vertices;
};
void drawarray(int punkty, int figure) // funkcją odpowiedzialna za rysowanie figury
{
	switch (figure)
	{// w zależności od wartości zmiennej figure, przekazywany jest do funkcji
glDrawArrays odpowiedni prymityw
	case 0:
		glDrawArrays(GL_TRIANGLE_FAN, 0, punkty);
		break;
	case 1:
		glDrawArrays(GL_POLYGON, 0, punkty);
		break;
	case 2:
		glDrawArrays(GL_POINTS, 0, punkty);
```

```
			break;
		case 3:
			glDrawArrays(GL_LINES, 0, punkty);
			break;
		case 4:
			glDrawArrays(GL_LINE_STRIP, 0, punkty);
			break;
		case 5:
			glDrawArrays(GL_LINE_LOOP, 0, punkty);
			break;
		case 6:
			glDrawArrays(GL_TRIANGLES, 0, punkty);
			break;
		case 7:
			glDrawArrays(GL_TRIANGLE_STRIP, 0, punkty);
			break;
		case 8:
			glDrawArrays(GL_QUADS, 0, punkty);
			break;
		case 9:
			glDrawArrays(GL_QUAD_STRIP, 0, punkty);
			break;
		default:
			glDrawArrays(GL_TRIANGLE_FAN, 0, punkty);
			break;
		}
}
int main()
{
		sf::ContextSettings settings;
		settings.depthBits = 24;
		settings.stencilBits = 8;
		// Okno renderingu
		sf::Window window(sf::VideoMode(SCREEN_WIDTH, SCREEN_HEIGHT, 32),
"OpenGL",
		sf::Style::Titlebar | sf::Style::Close, settings);
		// Inicjalizacja GLEW
		glewExperimental = GL_TRUE;
		glewInit();
		// Utworzenie VAO (Vertex Array Object)
		GLuint vao;
		glGenVertexArrays(1, &vao);
		glBindVertexArray(vao);
		GLuint vbo;
```

```
glGenBuffers(1, &vbo);

float vertices[] = {
-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f,
0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f,
0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f,
0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f,
-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f,
-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,
0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f,
0.5f, 0.5f, 0.5f, 1.0f, 1.0f, 0.0f,
0.5f, 0.5f, 0.5f, 1.0f, 1.0f, 0.0f,
-0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f,
-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,
-0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,
-0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,
-0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,
0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,
0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f,
0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,
0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
0.5f, -0.5f, -0.5f, 1.0f, 1.0f, 0.0f,
0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f,
0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f,
-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 0.0f,
-0.5f, -0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f,
0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,
0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,
-0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 0.0f,
-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f
};
int punkty = 36; // ilość wierzchołków figury + 1
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * punkty * 6, vertices,
GL_STATIC_DRAW);// zmieniony sposób alokacji pamięci
```

```
	// Utworzenie i skompilowanie shadera wierzchołków
	GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
	glShaderSource(vertexShader, 1, &vertexSource, NULL);
	glCompileShader(vertexShader);
	GLint status;
	glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &status); // sprawdzenie statusu kompilacjii
	if (status == GL_FALSE)// jeżeli kompilacja się nie powiodła wykona się poniższy kod
	{
		GLint maxLength = 0; // zmienna służąca do przechowywaniu długości informacji o błedzie
		glGetShaderiv(vertexShader, GL_INFO_LOG_LENGTH, &maxLength); // przekazanie informacji o długości do zmiennej
		char buffer[512]; //tablica przechowująca informacje o błędzie
		glGetShaderInfoLog(vertexShader, maxLength, &maxLength, buffer);// przekazanie informacji do tablicy
		std::cout << "Compilation vertexShader ERROR\n";
		std::cout << buffer << std::endl; //wypisanie odpowiedniego komunikatu
	}
	else
		std::cout << "Compilation vertexShader OK\n";
	GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
	glShaderSource(fragmentShader, 1, &fragmentSource, NULL);
	glCompileShader(fragmentShader);
	GLint status_second;
	glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &status_second);// sprawdzenie statusu kompilacjii
	if (status_second == GL_FALSE)// jeżeli kompilacja się nie powiodła wykona się poniższy kod
	{
		GLint maxLength_second = 0;// zmienna służąca do przechowywaniu długości informacji o błedzie
		glGetShaderiv(fragmentShader, GL_INFO_LOG_LENGTH, &maxLength_second);// przekazanie informacji o długości do zmiennej
		char buffer_second[512];//tablica przechowująca informacje o błędzie
		glGetShaderInfoLog(fragmentShader, maxLength_second, &maxLength_second, buffer_second);// przekazanie informacji do tablicy
		std::cout << "Compilation fragmentShader ERROR\n";
		std::cout << buffer_second << std::endl; //wypisanie odpowiedniego komunikatu
	}
	else
		std::cout << "Compilation fragmentShader OK\n";
	// Zlinkowanie obu shaderów w jeden wspólny program
```

```
	GLuint shaderProgram = glCreateProgram();
	glAttachShader(shaderProgram, vertexShader);
	glAttachShader(shaderProgram, fragmentShader);
	glBindFragDataLocation(shaderProgram, 0, "outColor");
	glLinkProgram(shaderProgram);
	glUseProgram(shaderProgram);
	// Specifikacja formatu danych wierzchołkowych
	GLint posAttrib = glGetAttribLocation(shaderProgram, "position");
	glEnableVertexAttribArray(posAttrib);
	glVertexAttribPointer(posAttrib, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), 0);
	// zmieniamy względem starego kodu 5 na 6 przy sizeof(GLfloat),
	//bo teraz każdy wierzchołek składa się z 6 wartości
	GLint colAttrib = glGetAttribLocation(shaderProgram, "color");
	glEnableVertexAttribArray(colAttrib);
	glVertexAttribPointer(colAttrib, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (void*)(3
*sizeof(GLfloat)));
	glm::mat4 model = glm::mat4(1.0f);// tworzenie macierzy modelu
	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));
	// tworzenie macierzy widoku
	glm::mat4 view = glm::lookAt(glm::vec3(0.0f, 0.0f, 1.0f), glm::vec3(1.0f, 0.0f, 0.0f),
glm::vec3(0.0f, 1.0f, 0.0f));
	//obsługa kamery
	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);
	view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
	GLint uniView = glGetUniformLocation(shaderProgram, "view");
	glUniformMatrix4fv(uniView, 1, GL_FALSE, glm::value_ptr(view));
	//tworzenie macierzy projekcji
	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 obrot = 0;
	bool running = true;
	float cameraSpeed = 0.0; // zmienne do sterowania kamerą
	double cameraSpeedSides = 0.05;
	sf::Clock clock;
	sf::Time time;
	window.setMouseCursorGrabbed(true); //przechwycenie kursora myszy w oknie
	window.setMouseCursorVisible(false); //ukrycie kursora myszy
	window.setFramerateLimit(20); // ustawienie maksymalnych fpsów
	double yaw = -90; //obrót względem osi Y
```

```
	double pitch = 0; //obrót względem osi X
	sf::Vector2i localPosition = sf::Mouse::getPosition(window);// współrzedne myszy
	double lastX = 0;// ostatnia pozycja x myszy
	double lastY = 0;// ostatnia pozycja y myszy
	double xoffset = localPosition.x - lastX; // różnica w pozycji x
	double yoffset = localPosition.y - lastY; // różnica w pozycji y
	lastX = localPosition.x; // aktualizacja współrzędnej x
	lastY = localPosition.y;// aktualizacja współrzędnej y
	const float sensitivity = 0.1; // czułość
	int fps_before = 0; // poprzednie fpsy
	glEnable(GL_DEPTH_TEST);
	unsigned int texture1; //Jak do wszystkich obiektów w OpenGL do tekstury można
odwołać się poprzez
	glGenTextures(1, &texture1); // Generuje tekstury
	glBindTexture(GL_TEXTURE_2D, texture1); //Ustawienie tekstury jako bieżąca
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	int width, height, nrChannels;
	stbi_set_flip_vertically_on_load(true); // obrócenie struktury na oś y
	unsigned char* data = stbi_load("metal.jpg", &width, &height, &nrChannels, 0); //
załadowanie zdjęcia
	if (data) // jeżeli załadowanie się powiodło nestępuje dodanie zdjęcia jako struktury
	{
		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); // uwolnienie pamięci
	glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * punkty * 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)));
	glBindTexture(GL_TEXTURE_2D, texture1); //zbindowanie tekstury
	GLint NorAttrib = glGetAttribLocation(shaderProgram, "aNormal");
	glEnableVertexAttribArray(NorAttrib);
```

```
	glVertexAttribPointer(NorAttrib, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat),
(void*)(3 *sizeof(GLfloat)));
	glm::vec3 lightPos(1.2f, 1.0f, 2.0f);
	GLint uniLightPos = glGetUniformLocation(shaderProgram, "lightPos");
	glUniform3fv(uniLightPos, 1, &lightPos[0]);
	float ambientStr = 0.5; // moc światła
	GLint loc = glGetUniformLocation(shaderProgram, "ambientStrength"); //bindowanie
do zmiennej z shadera
	glUniform1f(loc, ambientStr); // zapisanie zmiennej ambientStrength z shadera
wartości ambientStr
	int wlacznik = 1; // zmienna służąca do włączania/wyłączania światła
	while (running) {
		time = clock.getElapsedTime(); // pobranie czasu
		int fps = 1.0f / time.asSeconds();// obliczenie fpsów
		if (fps != fps_before)// jeżeli się zmieniły to wypisanie ich w tytule
		{
			std::string title = "FPS = " + std::to_string(fps);
			window.setTitle(title);
		}
		fps_before = fps;// zapisanie poprzednich fps
		clock.restart();// restart zegara
		sf::Event windowEvent;
		while (window.pollEvent(windowEvent)) {
			switch (windowEvent.type) {
			case sf::Event::Closed:
				running = false;
				break;
			case sf::Event::MouseMoved:
				// pobranie pozycji kursora myszy
				localPosition = sf::Mouse::getPosition(window);
				break;
			}
			if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
			{// oddalenie kamery
				cameraPos -= cameraSpeed * cameraFront;
			}
			if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
			{//przybliżenie kamery
				cameraPos += cameraSpeed * cameraFront;
			}
			if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
			{// obrócenie kamery w lewo
				cameraPos -= glm::normalize(glm::cross(cameraFront,
cameraUp)) *
```

```
                        cameraSpeed;
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
                {//obrócenie kamery w prawo
                        cameraPos += glm::normalize(glm::cross(cameraFront,
cameraUp)) *
                                cameraSpeed;
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Num1))
                {// rozjaśnienie
                        ambientStr += 0.01;
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Num2))
                {// przyciemnienie
                        ambientStr -= 0.01;
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Num3))
                {// włączenie/ wyłaczenie
                // jeżeli -1 to wyłaczony i moc oświetlenia ustawiona jest na -0.25
                // jeżeli 1 to światło jest właczone i oświetlenia ustawiona jest na
                // wcześniej ustawioną wartość
                   wlacznik = wlacznik * -1;
                        if (wlacznik == 1)
                                std::cout << "wlaczono swiatlo" << std::endl;
                        else
                        {
                                std::cout << "wylaczono swiatlo" << std::endl;
                        }
                        // żeby uniknać podwójnego pomnożenia przez za długo
przytrzymany przycisk
                                sf::sleep(sf::milliseconds(100));
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::R))
                {// reset mocy oświetlenia
                        ambientStr = 0.5;
                }
        }
        // obsługa kamery
        auto timeasMicro = time.asMicroseconds(); // pobranie czasu jako
mikrosekundy
        cameraSpeed = 0.000002f * timeasMicro; // zaktualizowanie szybkości
kamery
        xoffset = localPosition.x - lastX; // aktualizacja offsetu x
        yoffset = localPosition.y - lastY;// aktualizacja offsetu y
```

```
		lastX = localPosition.x; // aktualizacja ostaniej pozycji x myszy
		lastY = localPosition.y; // aktualizacja ostatniej pozycji y myszy
		xoffset *= sensitivity; // zaktualizawanie offsetu o czestotliwość
		yoffset *= sensitivity; // zaktualizawanie offsetu o czestotliwość
		yaw += xoffset; //zaktualizowanie pożenia x do ustawienia kamery
		pitch -= yoffset; // zaktualizowanie pożenia y do ustawienia kamery
		if (pitch > 89.0f)// jeżeli położenie y wieksze niż 89 to aktualizujemy je do 89
			pitch = 89.0f;
		if (pitch < -89.0f)// jeżeli położenie y mniejsze niż -89 to aktualizujemy je do -
89
			pitch = -89.0f;
		glm::vec3 front;
		front.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch)); // obliczenie
pozycji x kamery
			front.y = sin(glm::radians(pitch));// obliczenie pozycji y kamery
		front.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch)); // obliczenie
pozycji z kamery
			cameraFront = glm::normalize(front); // ustawienie frontu kamery
		view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
		glUniformMatrix4fv(uniView, 1, GL_FALSE, glm::value_ptr(view));
		// Nadanie scenie koloru czarnego
		glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
		if (wlacznik == -1)
		{
			glUniform1f(loc, -0.25);
		}
		else
		{
			glUniform1f(loc, ambientStr);
		}
		// funkcja rysująca figurę
		glDrawArrays(GL_TRIANGLE_FAN, 0, punkty);
		window.display();
	}
	// Kasowanie programu i czyszczenie buforów
	glDeleteProgram(shaderProgram);
	glDeleteShader(fragmentShader);
	glDeleteShader(vertexShader);
	glDeleteBuffers(1, &vbo);
	glDeleteVertexArrays(1, &vao);
	// Zamknięcie okna renderingu
	window.close();
	return 0;
```

```
}
```