Grupa: IO gr.1	Ćwiczenie: Lab 4	Imię Nazwisko: Malwina Cieśla
Wizualizacja Danych		

Cel ćwiczenia:

Zapoznanie z programowaniem grafiki przy użyciu shader'ów, użycie transformacji układu współrzędnych, obsługa kamery.

Przebieg ćwiczenia:

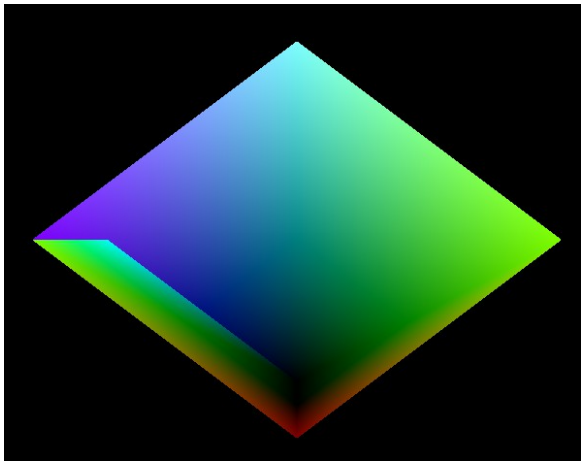
Na początku należało stworzyć sześcian oraz włączyć z-bufor:

```
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
```

Ilustracja 1: Fragment tablicy vertices



Ilustracja 2: Otrzymana figura

W celu uzyskania ostatecznego położenia punktów konieczne jest przekształcenie ich współrzędnych z uwzględnieniem macierzy modelu, widoku i projekcji, co w shaderze wygląda następująco:

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

Ilustracja 3: vertexSource

```
#include<glm/glm.hpp>
#include<glm/gtc/matrix_transform.hpp>
#include<glm/gtc/type_ptr.hpp>
#include <GL/glew.h>
```

Ilustracja 4: Potrzebne biblioteki

Aby rozpocząć rysowanie w 3D należy stworzyć macierz modelu. Macierz modelu może składać się z przekształcenia translacji, skalowania i / lub rotacji, które mają zostać wykonane w celu przekształcenia wszystkich wierzchołków obiektu w globalną przestrzeń świata. W moim programie wygląda to następująco:

```
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));
```

Ilustracja 5: Macierz modelu wraz z wysłaniem do shader'a

Następnie należy stworzyć macierz widoku, która przekształca wszystkie współrzędne świata na współrzędne widoku, które są zależne od położenia kamery i kierunku. Aby zdefiniować kamerę, trzeba znać jej pozycję w przestrzeni świata, kierunek w którym patrzy i wektor skierowany w górę. Stworzona macierz widoku wraz z funkcją tworzącą macierz widoku przedstawia się następująco:

```
glm::mat4 view;
view = glm::lookAt(glm::vec3(0.0f, 0.0f, 3.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
```

Ilustracja 6: Macierz widoku przy pomocy lookAt

Aby kamera mogła się poruszać trzeba zdefiniować zmienne ją opisujące:

```
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);
```

Ilustracja 7: Zmienne kamery

Następnie należało stworzyć kontrolę klawiszów (strzałek) klawiatury oraz klawisz dla obrotu obrazu. Fragment kodu z kontrolą eventów przedstawiam poniżej:

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
    cameraPos += cameraSpeed * cameraFront;
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
    cameraPos -= cameraSpeed * cameraFront;
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
    cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
    cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::L)) {
    obrot -= cameraSpeed;
    cameraFront.x = sin(obrot);
    cameraFront.z = -cos(obrot);
}
```

Ilustracja 8: Kontrola Eventów

Dodatkowo należy stworzyć macierz projekcji, którą można zdefiniować następująco:

```
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));
```

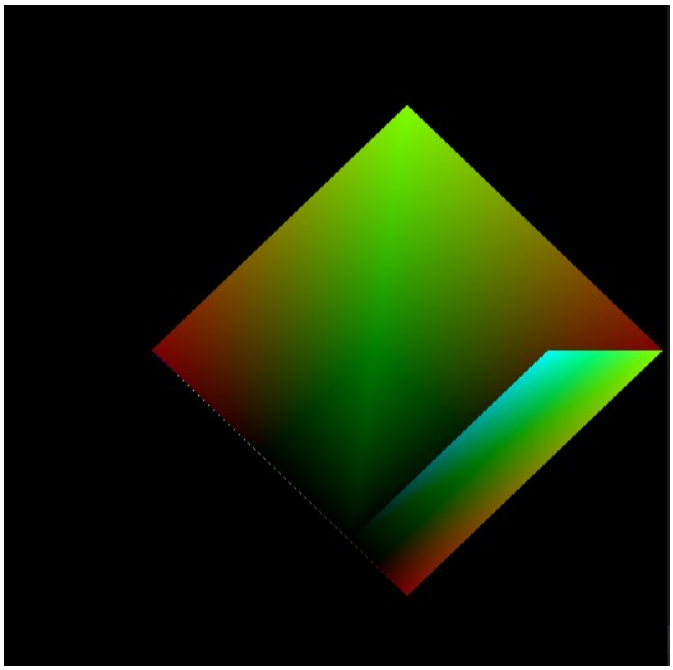
Ilustracja 9: Macierz projekcji

Ostatecznie należy przesłać macierz do karty graficznej:

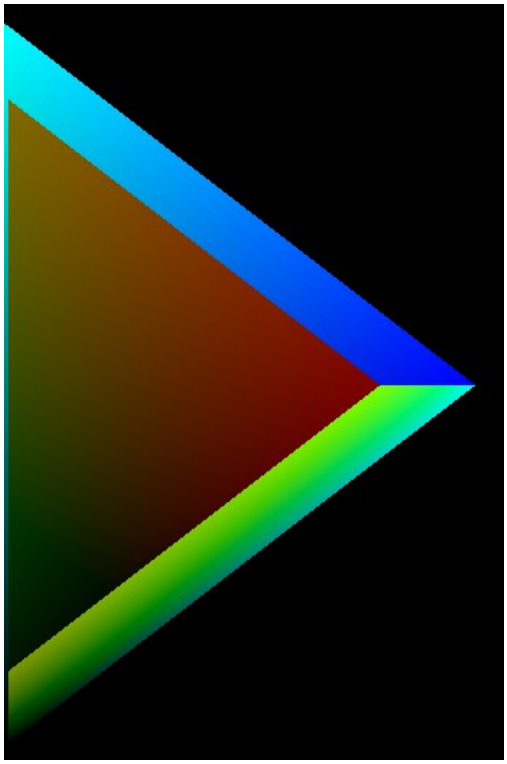
```
GLint uniView = glGetUniformLocation(shaderProgram, "view");
glUniformMatrix4fv(uniView, 1, GL_FALSE, glm::value_ptr(view));
```

Ilustracja 10: Przesłanie macierzy

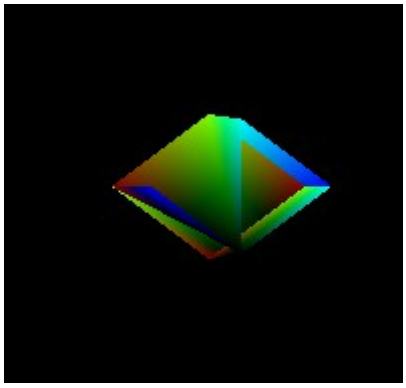
Dzięki temu udało mi się stworzyć obraz sześcianu, który po przekształceniach kamery wygląda następująco:



Ilustracja 11: Przykładowy obraz



Ilustracja 12: Przykładowe przybliżenie i obrócenie



Ilustracja 13: Przykładowy obraz

KOD:

```
#include "stdafx.h"
#include<glm/glm.hpp>
#include<glm/gtc/matrix_transform.hpp>
#include<glm/gtc/type_ptr.hpp>
#include <GL/glew.h>
#include <SFML/Window.hpp>
#include <SFML/System/Time.hpp>
#include <iostream>
using namespace std;

//ruch 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);

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

const GLchar* fragmentSource = R"glsl(
#version 150 core
in vec3 Color;
out vec4 outColor;
void main(){
outColor = vec4(Color, 1.0);
}
)glsl";

bool firstMouse = true;
int lastX, lastY;
double yaw = -90;
double pitch = 0;
double obrot = 5;
void setCamera(GLint view) {
	float cameraSpeed = 0.001f;
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
		cameraPos += cameraSpeed * cameraFront;
	}
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
		cameraPos -= cameraSpeed * cameraFront;
```

```
	}
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
		cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;
	}
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
		cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) * cameraSpeed;
	}
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::L)) {
		obrot -= cameraSpeed;
		cameraFront.x = sin(obrot);
		cameraFront.z = -cos(obrot);
	}

	glm::mat4 thisView;
	thisView = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
	glUniformMatrix4fv(view, 1, GL_FALSE, glm::value_ptr(thisView));
}

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);

	// 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 wierzchołkowych
	GLuint vbo;
	glGenBuffers(1, &vbo);

	GLfloat 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 };

glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

// Utworzenie i skompilowanie shadera wierzchołków
cout << "Compilation vertexShader: ";
GLuint vertexShader =
        glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexSource, NULL);
glCompileShader(vertexShader);
GLint status;
glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &status);
if (status == GL_TRUE)
        cout << "OK" << endl;
else {
        char buffer[512];
        glGetShaderInfoLog(vertexShader, 512, NULL, buffer);
}

// Utworzenie i skompilowanie shadera fragmentów
```

```
	cout << "Compilation fragmentShader: ";
	GLuint fragmentShader =
		glCreateShader(GL_FRAGMENT_SHADER);
	glShaderSource(fragmentShader, 1, &fragmentSource, NULL);
	glCompileShader(fragmentShader);
	GLint status2;
	glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &status2);
	if (status2 == GL_TRUE)
		cout << "OK" << endl;
	else {
		char buffer[512];
		glGetShaderInfoLog(fragmentShader, 512, NULL, buffer);
	}

	// 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);
	GLint colAttrib = glGetAttribLocation(shaderProgram, "color");
	glEnableVertexAttribArray(colAttrib);
	glVertexAttribPointer(colAttrib, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (void*)(2
* sizeof(GLfloat)));

	//macierz modelu wraz z wysłaniem do shader'a
	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::vec3 direction;
	direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
	direction.y = sin(glm::radians(pitch));
	direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
	float lastX = 400, lastY = 300;
	const float cameraSpeed = 0.05f;
	glm::vec3 cameraTarget = glm::vec3(0.0f, 0.0f, 0.0f);
	glm::vec3 cameraDirection = glm::normalize(cameraPos - cameraTarget);

	glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f);
	glm::vec3 cameraRight = glm::normalize(glm::cross(up, cameraDirection));
	glm::vec3 cameraUp = glm::cross(cameraDirection, cameraRight);

	//macierz widoku przy pomocy lookAt
	glm::mat4 view;
```

```
	view = glm::lookAt(glm::vec3(0.0f, 0.0f, 3.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

// wysłanie macierzy do karty graficznej
		GLint uniView = glGetUniformLocation(shaderProgram, "view");
		glUniformMatrix4fv(uniView, 1, GL_FALSE, glm::value_ptr(view));

//macierz 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));

	// Rozpoczęcie pętli zdarzeń
	bool running = true;
	sf::Time time;
	while (running) {
		sf::Event windowEvent;
		while (window.pollEvent(windowEvent)) {
			switch (windowEvent.type) {
			case sf::Event::Closed:
				running = false;
				break;
			}
		}
		sf::Event::KeyPressed;
		switch (windowEvent.key.code) {
		case sf::Keyboard::Escape: //zamknięcie okna na klawisz esc
			window.close();
			break;
		}

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

		glDrawArrays(GL_POLYGON, 0, 36);
		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;
}
```