Grupa: IO gr.1	Ćwiczenie: Lab 3	Imię Nazwisko: Malwina Cieśla
Wizualizacja Danych		

Cel ćwiczenia:

Zapoznanie z programowaniem grafiki przy użyciu shader'ów, SFML obsługa zdarzeń klawiatury i myszki.

Przebieg ćwiczenia:

Na początku należało stworzyć obsługę zdarzeń dla klawiatury, gdzie używanymi klawiszami byłby esc oraz liczby 0-9. Klawisz escape powodowałaby zamknięcie okna, a każda poszczególna liczba zmieniałaby prymityw potrzebny do rysowania figury:

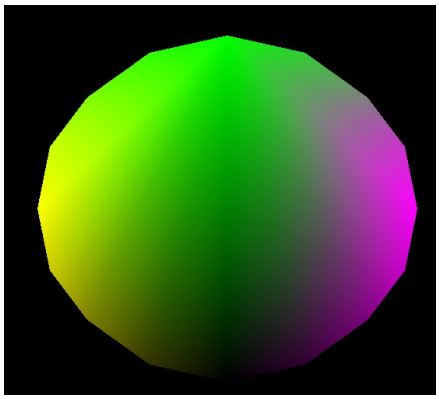
```
sf::Event::KeyPressed;
switch (windowEvent.key.code) {
case sf::Keyboard::Escape: //zamkn
    window.close();
    break;
case sf::Keyboard::Num1:
    prymityw=GL_POINTS;
    break;
case sf::Keyboard::Num2:
    prymityw=GL_LINES;
    break;
case sf::Keyboard::Num3:
    prymityw=GL_LINE_STRIP;
    break;
case sf::Keyboard::Num4:
    prymityw=GL_LINE_LOOP;
    break;
case sf::Keyboard::Num5:
    prymityw=GL_TRIANGLES;
    break;
case sf::Keyboard::Num6:
    prymityw=GL_TRIANGLE_STRIP;
    break;
```

Ilustracja 1: Obsługa zdarzeń klawiatury

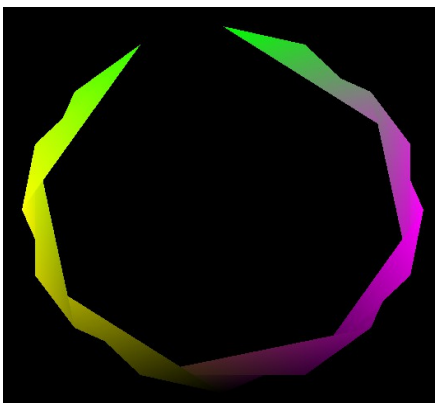
```
case sf::Keyboard::Num7:
    prymityw=GL_TRIANGLE_FAN;
    break;
case sf::Keyboard::Num8:
    prymityw=GL_QUADS;
    break;
case sf::Keyboard::Num9:
    prymityw=GL_QUAD_STRIP;
    break;
case sf::Keyboard::Num0:
    prymityw = GL_POLYGON;
    break;
```

Ilustracja 2: Obsługa zdarzeń klawiatury

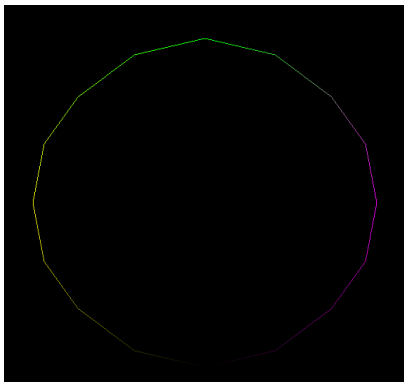
Następnie należało stworzyć tablicę dynamiczną, aby móc poprzez ruch myszki zmieniać liczbę wierzchołków:



Ilustracja 3: Przykład



Ilustracja 5: Przykład



Ilustracja 4: Przykład

```
GLfloat* vertices = new GLfloat[punkty * 6];
```

Ilustracja 6: Tablica dynamiczna

Następnie stworzyłam funkcję, dzięki której przy pomocy przechwyconej pozycji myszki będę mogła dodać punkt do tablicy vertices:

```
void addPoint(GLfloat* ver, double x, double y, int r, int g, int b, int punkty) {
    ver[punkty - 6] = x;
    ver[punkty - 5] = y;
    ver[punkty - 4] = 0;
    ver[punkty - 3] = r;
    ver[punkty - 2] = g;
    ver[punkty - 1] = b;
}
```

Ilustracja 7: Funkcja addPoint

Obsługę zdarzeń dla ruchu myszki przedstawiam poniżej:

```
case (sf::Event::MouseMoved):
        GLfloat mouseY = (GLfloat)sf::Mouse::getPosition(window).y;
        punkty++;
        GLfloat* newVer = new GLfloat[(punkty - 1) * 6];
        vertices = new GLfloat[punkty * 6];
        for (int i = 0; i < punkty - 1; i++)
            vertices[i] = newVer[i];
        sf::Vector2i position = sf::Mouse::getPosition(window);
        addPoint(vertices, position.x, position.y, static_cast<float>(rand())/static_cast<float>(RAND_MAX),
            static_cast<float>(rand()) / static_cast<float>(RAND_MAX),
            static_cast<float>(rand()) / static_cast<float>(RAND_MAX),punkty);
        break;
}
```

Ilustracja 8: Obsługa myszy

Dodatkowo stworzyłam również obsługę przycisku myszki, która po kliknięciu pokazuje nam ile zostało stworzonych punktów:

```
sf::Event::MouseMoved;
switch (windowEvent.mouseMove.x) {
case sf::Mouse::Left:
        cout << "Punkty: " << punkty;
        break;
}
```

Ilustracja 9: Obsługa myszy

Kod:

```
// Nagłówki
#include "stdafx.h"
#include <GL/glew.h>
#include <SFML/Window.hpp>
#include <iostream>
using namespace std;

// Kody shaderów
const GLchar* vertexSource = R"glsl(
#version 150 core
```

```
in vec3 position;
in vec3 color;
out vec3 Color;
void main(){
Color = color;
gl_Position = 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";

void addPoint(GLfloat* ver, double x, double y, int r, int g, int b, int punkty) {
	ver[punkty - 6] = x;
	ver[punkty - 5] = y;
	ver[punkty - 4] = 0;
	ver[punkty - 3] = r;
	ver[punkty - 2] = g;
	ver[punkty - 1] = b;
}

int main()
{
	int punkty = 6;
	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 = new GLfloat[punkty * 6];
```

```
	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)));
	GLenum prymityw=GL_POLYGON;
	// Rozpoczęcie pętli zdarzeń
	int i = 0;
```

```
	bool running = true;
	while (running) {
		sf::Event windowEvent;
		while (window.pollEvent(windowEvent)) {
			switch (windowEvent.type) {
			case sf::Event::Closed:
				running = false;
				break;
			case (sf::Event::MouseMoved):
					GLfloat mouseY =
(GLfloat)sf::Mouse::getPosition(window).y;
					punkty++;
					GLfloat* newVer = new GLfloat[(punkty - 1) * 6];
					vertices = new GLfloat[punkty * 6];
					for (int i = 0; i < punkty - 1; i++)
						vertices[i] = newVer[i];
					sf::Vector2i position = sf::Mouse::getPosition(window);
					addPoint(vertices, position.x, position.y,
static_cast<float>(rand())/static_cast<float>(RAND_MAX),
						static_cast<float>(rand()) /
static_cast<float>(RAND_MAX),
						static_cast<float>(rand()) /
static_cast<float>(RAND_MAX),punkty);
					break;
			}
		}
		sf::Event::KeyPressed;
		switch (windowEvent.key.code) {
		case sf::Keyboard::Escape: //zamknięcie okna na klawisz esc
			window.close();
			break;
		case sf::Keyboard::Num1:
			prymityw=GL_POINTS;
			break;
		case sf::Keyboard::Num2:
			prymityw=GL_LINES;
			break;
		case sf::Keyboard::Num3:
			prymityw=GL_LINE_STRIP;
			break;
		case sf::Keyboard::Num4:
			prymityw=GL_LINE_LOOP;
			break;
		case sf::Keyboard::Num5:
			prymityw=GL_TRIANGLES;
			break;
		case sf::Keyboard::Num6:
			prymityw=GL_TRIANGLE_STRIP;
			break;
		case sf::Keyboard::Num7:
			prymityw=GL_TRIANGLE_FAN;
```

```
                break;
            case sf::Keyboard::Num8:
                prymityw=GL_QUADS;
                break;
            case sf::Keyboard::Num9:
                prymityw=GL_QUAD_STRIP;
                break;
            case sf::Keyboard::Num0:
                prymityw = GL_POLYGON;
                break;
            }
            sf::Event::MouseMoved;
            switch (windowEvent.mouseMove.x) {
            case sf::Mouse::Left:
                        cout << "Punkty: " << punkty;
                        break;
            }
            //Mouse.onclick
            // Nadanie scenie koloru czarnego
            glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
            glClear(GL_COLOR_BUFFER_BIT);


            // Narysowanie kolorowego koła
            glDrawArrays(prymityw, 0, 16);

            // Wymiana buforów tylni/przedni
            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;
}
```