Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

SPRAWOZDANIE

Lab nr 3

Celem ćwiczeń było zapoznanie się z programowaniem grafiki przy użyciu shader'ów oraz obsługa zdarzeń klawiatury i myszki.

1. Dodanie funkcji tworzącej nowy wektor z współrzędnymi zależnymi od ilości punktów oraz losowymi kolorami.

```
GLfloat* CreateVert(int punkty, GLfloat*& v)
{
    delete[] v;
    GLfloat* vertices = new GLfloat[punkty * 6];
    srand((unsigned)time(0));
    GLfloat temp = 0.5;////(float)(rand() % 51) + 25) / 100; ///
    GLfloat k = 2 * 3.14 / punkty; //// *2

    for (int i = 0; i < punkty; i++)
    {
        vertices[i * 6] = temp * cos((i + 1) * k);
        vertices[i * 6 + 1] = temp * sin((i + 1) * k);
        // vertices[(i * 6) + 2] = 0;
        vertices[(i * 6) + 2] = ((float)(rand() % 101) + 0) / 100;
        vertices[(i * 6) + 3] = ((float)(rand() % 101) + 0) / 100;
        vertices[(i * 6) + 4] = ((float)(rand() % 101) + 0) / 100;
    }
    return vertices;
}
```

2. Utworzenie funkcji odpowiedzialnej za użycie prymitywu.

```
void draw(int punkty, int input)// funkcją odpowiedzialna za rysowanie figury
{
    switch (input)
    {
    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;
    }
}
```

Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

3. Sprawdzenie pozycji kursora co drugą pętlę.

```
while (running) {
    sf::Event windowEvent;
    while (window.pollEvent(windowEvent)) {
        j++;
        if (j % 2)
            mouse_y = windowEvent.mouseMove.y;
        switch (windowEvent.type) {
        case sf::Event::Closed:
```

4. Obsługa klawiszy 0-9.

```
switch (windowEvent.type) {
case sf::Event::Closed:
    running = false;
    break;
case sf::Event::KeyPressed: /// case dla klawiatury
    switch (windowEvent.key.code)
    {
    case sf::Keyboard::Num0:
        i = 0;
        break;
    case sf::Keyboard::Num1:
        i = 1;
        break;
    case sf::Keyboard::Num2:
        i = 2;
        break;
    case sf::Keyboard::Num3:
        i = 3;
        break;
    case sf::Keyboard::Num4:
        i = 4;
        break;
    case sf::Keyboard::Num5:
        i = 5;
        break;
    case sf::Keyboard::Num6:
        i = 6;
        break;
    case sf::Keyboard::Num7:
        i = 7;
        break;
    case sf::Keyboard::Num8:
        i = 8;
        break;
    case sf::Keyboard::Num9:
        i = 9;
        break;
    }
    break;
```

Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

5. Obsługa kursora.

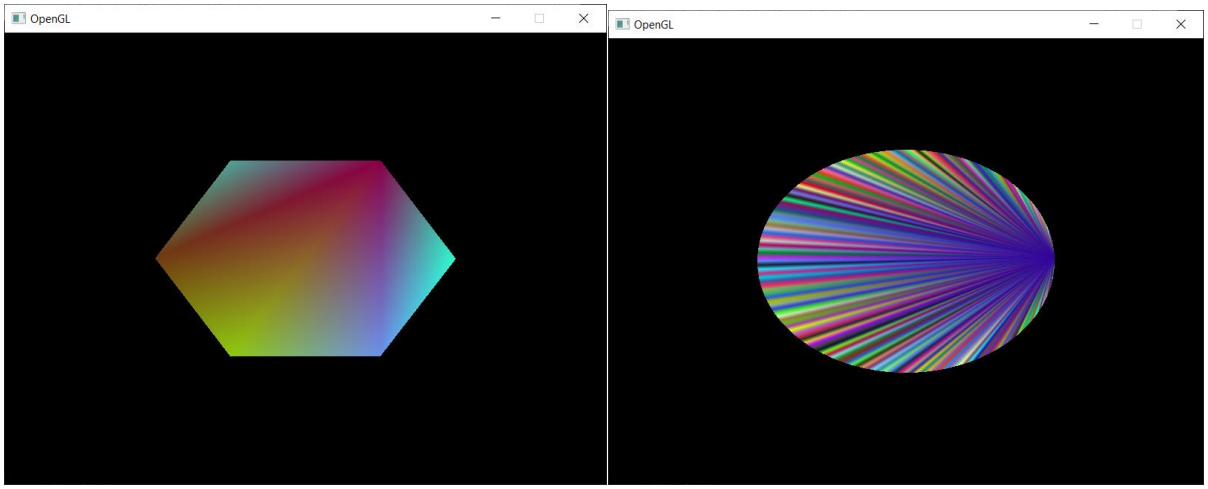
```
case sf::Event::MouseMoved:

    std::cout << "mousey: " << mouse_y << "\n";
    std::cout << windowEvent.mouseMove.y << "\n";
    if (windowEvent.mouseMove.y > mouse_y)
    {
        punkty++;
    }
    if (windowEvent.mouseMove.y < mouse_y && punkty != 3)
    {
        punkty--;
    }
    break;
}
```

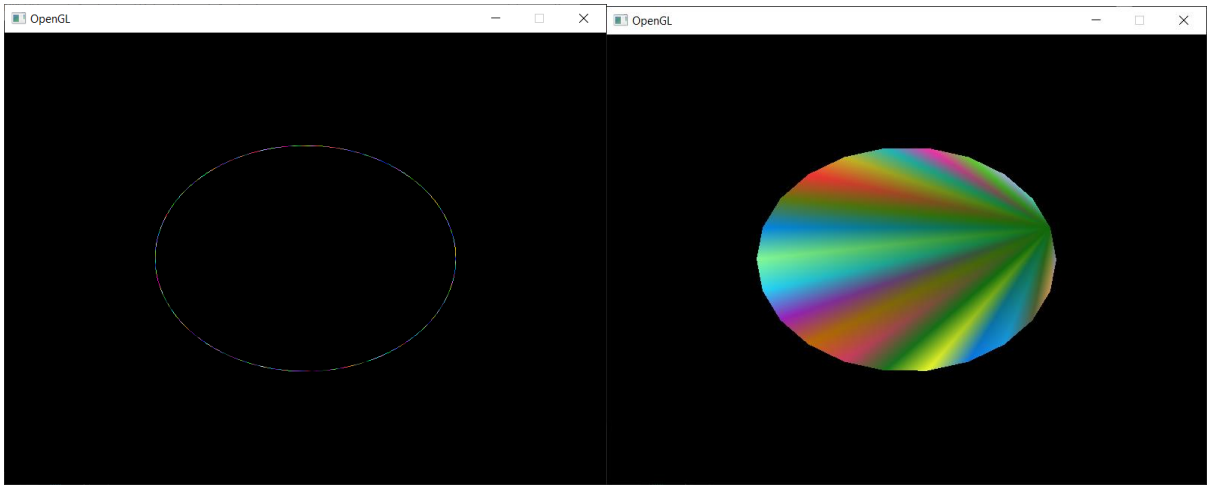
6. Zmiana argumentów funkcji przekazujących rozmiar wektora z 4 na.

```
GLint posAttrib = glGetAttribLocation(shaderProgram, "position");
glEnableVertexAttribArray(posAttrib);
glVertexAttribPointer(posAttrib, 2, 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),
```

PRZYKŁADY DZIAŁANIA PROGRAMU:



Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3



KOD:

```
//Naglowki
#include <iostream>
#include <GL/glew.h>
#include <SFML/Window.hpp>
#include <math.h>
#include <stdlib.h>
#include <time.h>

// Kody shaderow
const GLchar* vertexSource = R"glsl(
#version 150 core
in vec2 position;
in vec3 color;
out vec3 Color;
void main(){
Color = color;
gl_Position = vec4(position, 0.0, 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";

/* GLfloat vertices[] = {
 0.7f * cos(3.14159/4),0.7f * sin(3.14159 / 4), 1.0f, 0.0f, 0.0f,
 0.7f * cos(3.14159 *3 / 4),0.7f * sin(3.14159 *3/ 4), 0.5f, 0.0f, 0.8f,
 0.7f * cos(3.14159 * 5/ 4),0.7f * sin(3.14159 *5/ 4), 0.0f, 1.0f, 0.0f,
 0.7f * cos(3.14159 *7/ 4),0.7f * sin(3.14159 *7/ 4), 0.0f, 0.5f, 1.0f,
 };*/
```

Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

```
GLfloat* CreateVert(int punkty, GLfloat*& v)
{
	delete[] v;
	GLfloat* vertices = new GLfloat[punkty * 6];
	srand((unsigned)time(0));
	GLfloat temp = 0.5;//((float)(rand() % 51) + 25) / 100; ///
	GLfloat k = 2 * 3.14 / punkty; //// *2

	for (int i = 0; i < punkty; i++)
	{
		vertices[i * 6] = temp * cos((i + 1) * k);
		vertices[i * 6 + 1] = temp * sin((i + 1) * k);
		// vertices[(i * 6) + 2] = 0;
		vertices[(i * 6) + 2] = ((float)(rand() % 101) + 0) / 100;
		vertices[(i * 6) + 3] = ((float)(rand() % 101) + 0) / 100;
		vertices[(i * 6) + 4] = ((float)(rand() % 101) + 0) / 100;
	}
	return vertices;
}
void draw(int punkty, int input)// funkcją odpowiedzialna za rysowanie figury
{
	switch (input)
	{// 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(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);
int punkty = 3;
GLfloat* vertices = new GLfloat[punkty * 6];
vertices = CreateVert(punkty, vertices);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * punkty * 6, vertices,
		GL_STATIC_DRAW);
// Utworzenie i skompilowanie shadera wierzchołków
GLuint vertexShader =
		glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexSource, NULL);
glCompileShader(vertexShader);
// dopunkt 1
GLint status;
glGetShaderiv(vertexShader, GL_COMPILE_STATUS,
		&status);
if (status)
{
		std::cout << "compilation vertexShader OK \n";
}
else
{
		char buffer[512];
		glGetShaderInfoLog(vertexShader, 512, NULL, buffer);
		std::cout << buffer;
}
// Utworzenie i skompilowanie shadera fragmentów
GLuint fragmentShader =
		glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentSource, NULL);
glCompileShader(fragmentShader);
// dopunkt 1
GLint status2;
glGetShaderiv(fragmentShader, GL_COMPILE_STATUS,
		&status2);
if (status)
{
		std::cout << "compilation vertexShader OK \n";
}
else
{
		char buffer2[512];
		glGetShaderInfoLog(fragmentShader, 512, NULL, buffer2);
		std::cout << buffer2;
}
```

```
// 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, 2, 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)));
// Rozpoczęcie pętli zdarzeń
bool running = true;
int i = 0; //input
int prev_punkty = punkty;
int j = 0;
int mouse_y = 0; // mouse_y = windowEvent.mouseMove.y;
while (running) {
	sf::Event windowEvent;
	while (window.pollEvent(windowEvent)) {
		j++;
		if (j % 2)
			mouse_y = windowEvent.mouseMove.y;
		switch (windowEvent.type) {
		case sf::Event::Closed:
			running = false;
			break;
		case sf::Event::KeyPressed: /// case dla klawiatury
			switch (windowEvent.key.code)
			{
			case sf::Keyboard::Num0:
				i = 0;
				break;
			case sf::Keyboard::Num1:
				i = 1;
				break;
			case sf::Keyboard::Num2:
				i = 2;
				break;
			case sf::Keyboard::Num3:
				i = 3;
				break;
			case sf::Keyboard::Num4:
				i = 4;
				break;
			case sf::Keyboard::Num5:
				i = 5;
				break;
			case sf::Keyboard::Num6:
				i = 6;
				break;
			case sf::Keyboard::Num7:
				i = 7;
				break;
			case sf::Keyboard::Num8:
				i = 8;
				break;
```

```
                                        case sf::Keyboard::Num9:
                                                i = 9;
                                                break;
                                        }
                                        break;

                                case sf::Event::MouseMoved:

                                        std::cout << "mousey: " << mouse_y << "\n";
                                        std::cout << windowEvent.mouseMove.y << "\n";
                                        if (windowEvent.mouseMove.y > mouse_y)
                                        {
                                                punkty++;
                                        }
                                        if (windowEvent.mouseMove.y < mouse_y && punkty != 3)
                                        {
                                                punkty--;
                                        }
                                        break;
                                }
                        }
                        if (prev_punkty != punkty)
                        {
                                vertices = CreateVert(punkty, vertices);
                                // ponowna alokacja

                        }
                        glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * punkty * 6,
                                vertices, GL_DYNAMIC_DRAW);
                        prev_punkty = punkty;

                        // Nadanie scenie koloru czarnego
                        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
                        glClear(GL_COLOR_BUFFER_BIT);
                        // Narysowanie trójkąta na podstawie 3 wierzchołków
                        draw(punkty, i);
                        // glDrawArrays(GL_POLYGON,0, 4);
                        // 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;
}
```