Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

# SPRAWOZDANIE

Lab nr 4

Celem ćwiczenia było zapoznanie się z programowaniem grafiki przy użyciu shader'ów oraz użycie transformacji układu współrzędnych oraz obsługa kamery.

1. Dołączenie plików nagłówkowych.

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

2. Zmiana shader'ów w kodzie tak jak w instrukcji.

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

3. Utworzenie float vertices[] oraz ustawienie zmiennej punkty na 36 – tyle ile punktów zdeklarowano w kodzie.
4. Zmiana 2 na 3 – mamy 3 współrzędne.

```
// 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*)(3 * sizeof(GLfloat)));
```

5. Utworzenie macierzy modelu i wysłanie jej do shadera.
6. Utworzenie macierzy widoku i zdefiniowanie zmiennych ją opisujących, wysłanie macierzy do karty graficznej.

7. Utworzenie macierzy projekcji.

```
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(glm::vec3(0.0f, 0.0f, 1.0f),
    glm::vec3(1.0f, 0.0f, 0.0f),
    glm::vec3(0.0f, 1.0f, 0.0f));
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));

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));
```

8. Utworzenie zmiennych do sterowania kamerą.

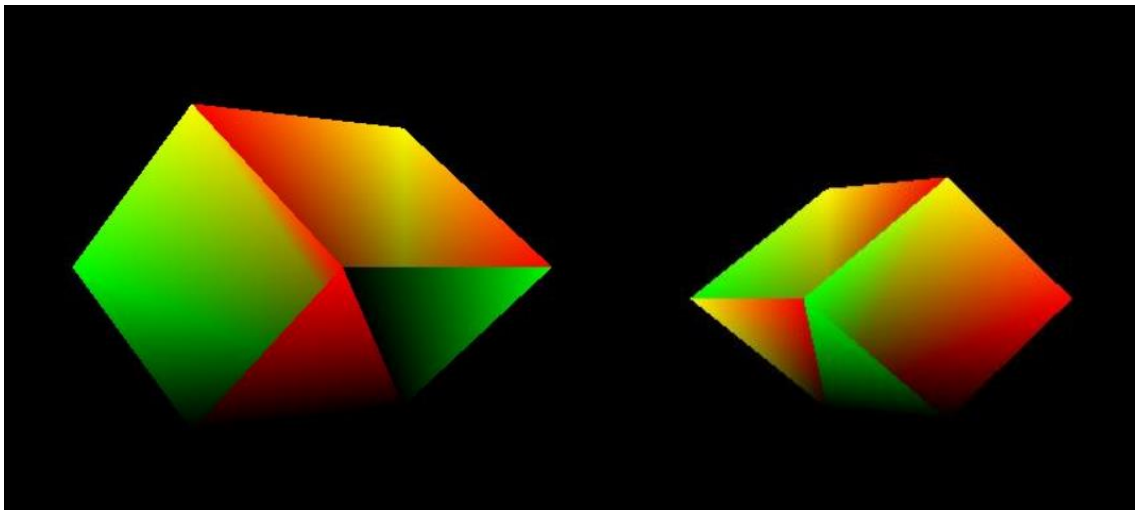
```
float cameraSpeed = 0.1;
float cameraSpeedSides = 0.05;
float obrot = 0;
```

9. Włączenie bufora przed pętlą zdarzeń umożliwiając tym wyświetlanie się przestrzenne figury. ***glEnable(GL_DEPTH_TEST);***
10. Przypisanie do danych klawiszy odpowiednich akcji.

```
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
{
    cameraPos -= cameraSpeed * cameraFront;
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
{
    cameraPos += cameraSpeed * cameraFront;
}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
{
    obrot -= cameraSpeedSides;
    cameraFront.x = sin(obrot);
    cameraFront.z = -cos(obrot);
}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
{

    obrot += cameraSpeedSides;
    cameraFront.x = sin(obrot);
    cameraFront.z = -cos(obrot);
}
```

Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

WYNIKI PROGRAMU:



KOD:

```
#include <iostream>
#include <GL/glew.h>
#include <SFML/Window.hpp>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <math.h>
#include <stdlib.h>
#include <time.h>
// Kody shaderów
const GLchar* vertexSource = R"glsl(
#version 150 core
in vec3 position;
in vec3 color;
out vec3 Color;
uniform mat4 model;
uniform mat4 view;
uniform mat4 proj;
void main(){
Color = color;
gl_Position = proj*view*model*vec4(position, 1.0); //okreslanie pozycji
}
)glsl";
const GLchar* fragmentSource = R"glsl(
#version 150 core
in vec3 Color;
out vec4 outColor;
```

Izabela Wenda
Inżynieria Obliczeniowa
Gr nr 3

```
void main()
{
outColor = vec4(Color, 1.0);
}
)glsl";
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);
 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;
 //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, 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*)(3 * sizeof(GLfloat)));
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(glm::vec3(0.0f, 0.0f, 1.0f),
glm::vec3(1.0f, 0.0f, 0.0f),
glm::vec3(0.0f, 1.0f, 0.0f));
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));
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;
int i = 0; //input
int prev_punkty = punkty;
int j = 0;
int mouse_y = 0; // mouse_y = windowEvent.mouseMove.y;
float cameraSpeed = 0.1;
float cameraSpeedSides = 0.05;
float obrot = 0;
glEnable(GL_DEPTH_TEST);
while (running)
{
sf::Event windowEvent;
while (window.pollEvent(windowEvent)) {
switch (windowEvent.type) {
case sf::Event::Closed:
running = false;
break;
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
{
cameraPos -= cameraSpeed * cameraFront;
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
{
cameraPos += cameraSpeed * cameraFront;
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
{
obrot -= cameraSpeedSides;
cameraFront.x = sin(obrot);
cameraFront.z = -cos(obrot);
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
{
obrot += cameraSpeedSides;
cameraFront.x = sin(obrot);
```

```
cameraFront.z = -cos(obrot);
}
}
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);
// 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;
}
```