Drukuj

Wprowadzenie do programowania w OpenGL z użyciem shader'ów

1. **Cel ćwiczenia:**

   Zapoznanie z programowaniem grafiki przy użyciu shader'ów.

2. **Wstęp:**
   Shadery – to krótkie programy napisane w specjalnym języku (shader language, podobnym do C), który w opisuje właściwości pikseli (fragmentów) oraz wierzchołków (vertex'ów). Technologia ta zastąpiła stosowaną wcześniej jednostkę T&L.

3. **Ćwiczenie:**
   Należy przygotować projekt umożliwiający rysowanie sceny z użyciem potoku renedowania zgodnego z OpenGL 3.3+. Do utworzenia okna oraz do zapewnienia interakcji z użytkownikiem należy wykorzystać bibliotekę SFML - Simple and Fast Multimedia Library oraz biblioteki GLEW The OpenGL Extension Wrangler Library.

4. **Budowa prjektu:**

   Projekt będzie budowany w Visual Studio 2015
   1. **Należy utworzyć projekt:**

      File -> New ->Project ...

      Visual C++ -> Wln 32 Console Application - wpisać nazwę projektu,

      Next -> Finish.

   2. **Dodanie bibliotek:**

      Aby otrzymać dostęp do rozszerzeń OpenGL i najnowszej wersji konieczne jest dodoanie biblioteki GLEW

      - Należy pobrać najnowszą wersję biblioteki GLEW dla win 32 i 64 ze strony http://glew.sourceforge.net/

      - Naspępnie przekopiować rozpakowany katalog z biblioteką do katalogu projektu (w przypadku przenoszenia projektu nie będzie problemów z bibliotekami),

      - Następnie w menu Project -> Properties -> C/C++ -> General -> Aditional Include Directories -> <Edit...> w pustym obszarze dopisać ..\glew-2.1.0\include (dołączenie katalogu z plikami nagłówkowymi)

      - Następnie w menu Project -> Properties -> Linker -> General -> Aditional Library Directories -> <Edit...> w pustym obszarze dopisać ..\glew-2.1.0\lib\Release\Win32 (dołączenie katalogu z plikami bibliotek)

      - Następnie w menu Project -> Properties -> Linker -> Input -> Additional Dependencies <Edit...> w pustym obszarze dopisać glew32.lib i zaakceptować OK

      - Ponieważ przedstawiony sposób dołącza biblioteki dynamicznie konieczne jest skopiowanie pliku dll do katalogu Release i/lub Debug (odpowiednie katalogi zostaną utworzone po uruchomieniu programu)

      - Należy skopiować z katalogu \bin\Release\Win32\ plik glew32.dll do katalogów Release i Debug.

      Od tej pory można używać biblioteki po dodaniu pliku nagłówkowego #include <GL\glew.h>

      Aby otrzymać możliwość utworzenia okna w naszym przypadku użyjemy biblioteki SFML

      Należy pobrać najnowszą wersję biblioteki SFML dla win 32 ze strony https://www.sfml-dev.org/

      - Naspępnie przekopiować rozpakowany katalog z biblioteką do katalogu projektu (w przypadku przenoszenia projektu nie będzie problemów z bibliotekami),

      - Następnie w menu Project -> Properties -> C/C++ -> General -> Aditional Include Directories -> <Edit...> w pustym obszarze dopisać ..\SFML-2.4.2\include (dołączenie katalogu z plikami nagłówkowymi)

      - Następnie w menu Project -> Properties -> Linker -> General -> Aditional Library Directories -> <Edit...> w pustym obszarze dopisać ..\SFML-2.4.2\lib (dołączenie katalogu z plikami bibliotek)

- Następnie w menu Project -> Properties -> Linker -> Input -> Additional Dependencies <Edit...> w pustym obszarze dopisać

sfml-graphics-d.lib
sfml-window-d.lib
sfml-audio-d.lib
sfml-network-d.lib
sfml-system-d.lib
opengl32.lib
i zaakceptować OK

- Ponieważ przedstawiony sposób dołącza biblioteki dynamicznie konieczne jest skopiowanie pliku dll do katalogu Release i/lub Debug (odpowiednie katalogi zostaną utworzone po uruchomieniu programu)

- Należy skopiować z katalogu \SFML-2.4.2\bin\ plik sfml-window-d-2.dll i sfml-system-d-2.dll do katalogu Debug. W przypadku Release należy skopiować pliki bez litery d w nazwie.

Od tej pory można używać biblioteki po dodoaniu pliku nagłówkowego #include #include <SFML\Window.hpp>

5. **Zadanie projektowe:**

Przykładowy kod:

```
// Nagłówki
#include "stdafx.h"
#include <GL/glew.h>
#include <SFML/Window.hpp>

// Kody shaderów
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";
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[] = {
0.0f, 0.5f, 1.0f, 0.0f, 0.0f,
0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
```

```
-0.5f, -0.5f, 0.0f, 0.0f, 1.0f
};
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

// Utworzenie i skompilowanie shadera wierzchołków
GLuint vertexShader =
glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexSource, NULL);
glCompileShader(vertexShader);

// Utworzenie i skompilowanie shadera fragmentów
GLuint fragmentShader =
glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentSource,NULL);
glCompileShader(fragmentShader);

// 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, 5 * sizeof(GLfloat), 0);
GLint colAttrib = glGetAttribLocation(shaderProgram, "color");
glEnableVertexAttribArray(colAttrib);
glVertexAttribPointer(colAttrib, 3, GL_FLOAT,GL_FALSE, 5 * sizeof(GLfloat), (void*)(2 * sizeof(GLfloat)));

// Rozpoczęcie pętli zdarzeń
bool running = true;
while (running) {
sf::Event windowEvent;
while (window.pollEvent(windowEvent)) {
switch (windowEvent.type) {
case sf::Event::Closed:
running = false;
break;
}
}
// Nadanie scenie koloru czarnego
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);

// Narysowanie trójkąta na podstawie 3 wierzchołków
glDrawArrays(GL_TRIANGLES, 0, 3);
// 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;
}
```

Na podstawie przykładowego projektu wykonać zadania:

- Dodać kod sprawdzający poprawność kompilacji shader'ów oraz wyświetlający odpowiedni komunikat w konsoli łącznie z kodami ewentualnych błędów.
- Wyświtlaną figurę zmienić na dowolny wielokąt foremny (współrzędne cylindryczne),
- Dodać do tablicy współrzędną Z (teraz jest tylko X i Y),
- W każdym wierzchołku nowej figury ustawic inny kolor RGB.

Przykładowy wygląd aplikacji:

```
 Compilation vertexShader ERROR
ERROR: 0:7: 'color' : undeclared identifier
ERROR: 0:7: 'assign' :  cannot convert from 'float' to 'varying 3-component vector of float'


 Compilation fragmentShader OK
```