POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Dokumentacja projektu z przedmiotu Grafika Komputerowa

Microcar Flex Furgon

Cezary Bober
Michał Kruczek

Rzeszów, 2019

Spis treści

1 Opis projektu

Celem projektu było stworzenie interaktywnego modelu prostego samochodu korzystając z biblioteki OpenGL. Dzięki wykonaniu projektu poszerzyliśmy naszą wiedzę dotyczącą zarówno tworzenia modeli 3D jak i z pozostałych zagadnień składających się na dziedzinę jaką jest grafika komputerowa.

Stworzony pojazd z pozwala na jazdę do przodu oraz do tyłu, zaimplementowane zostały także pewne elementy interaktywne takie jak otwieranie i zamykanie klapy bagażnika oraz maski. Samochód posiada także kręcący się wał napędowy napędzający potężny silnik.

1.1 Wykorzystane technologie

Projekt został napisany przy wykorzystaniu języka C++ w wersji 11. Przy tworzeniu projektu wykorzystywaliśmy bibliotekę graficzną *OpenGL*. Oprócz niej wykorzystana została biblioteka *OpenGL Mathematics*, która pozwala na proste wykorzystanie bardziej skomplikowanych narzędzi aparatu matematycznego. Kolejną biblioteką była *GLFW*, która zapewnia proste API do tworzenia okien, kontekstów i powierzchni, odbierania danych wejściowych i zdarzeń. Załadowanie tych bibliotek było możliwe dzięki bibliotece *OpenGL Extension Wrangler Library*, która zapewnia wydajne mechanizmy uruchamiania w celu określenia, które rozszerzenia OpenGL są obsługiwane na platformie docelowej.

Modele poszczególnych części samochodu zostały stworrzone w narzędziu *Blender*, który jest otwartym oprogramowaniem do modelowania i renderowania obrazów oraz animacji trójwymiarowych. Następnie dzięki bibliotece *Open Asset Import Library* zostały one umieszczone w programie.

2 Inspiracja

Inspiracją do stworzenia omawianego modelu samochodu był posiadany przez jednego z autorów projektu samochód *Microcar MC1* przedstawiony na zdjęciu poniżej.

Umieszczona w projekcie wersja jest modelem zmodyfikowanym na potrzeby handlu towarowego. Niewielkie rozmiary i mała waga powodują, że jest to bardzo dobry pojazd do ciasnych i zatłoczonych miast.

Rysunek 1: Microcar MC1 *"Szerszeń"*

Rysunek 2: Microcar Flex Furgon

3 Powstawanie projektu

3.1 Hello World

```
1  #include "Window.h"
2
3  Window::Window()
4  {
5          width = 800;
```

```
6           height = 600;
7           xChange = 0;
8           yChange = 0;
9           mouseFirstMoved = true;
10          for (size_t i = 0; i < 1024; i++) keys[i] = false;
11  }
12
13  Window::Window(GLint windowWidth, GLint windowHeight)
14  {
15          width = windowWidth;
16          height = windowHeight;
17          xChange = 0;
18          yChange = 0;
19          mouseFirstMoved = true;
20          for (size_t i = 0; i < 1024; i++) keys[i] = false;
21  }
22
23  int Window::Initialize()
24  {
25          if (!glfwInit())
26          {
27                  printf("Error Initialising GLFW");
28                  glfwTerminate();
29                  return 1;
30          }
31
32          glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
33          glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
34          glfwWindowHint(GLFW_OPENGL_PROFILE,
            ↪  GLFW_OPENGL_CORE_PROFILE);
35          glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
36          glfwWindowHint(GLFW_REFRESH_RATE, 60);
37
38          const GLFWvidmode* mode =
            ↪  glfwGetVideoMode(glfwGetPrimaryMonitor());
39          mainWindow = glfwCreateWindow(width, height, "Microcar
            ↪  Flex Furgon", NULL, NULL);
40          width = mode->width;
41          height = mode->height;
42          if (!mainWindow)
43          {
44                  printf("Error creating GLFW window!");
45                  glfwTerminate();
46                  return 1;
47          }
48
49          glfwGetFramebufferSize(mainWindow, &bufferWidth,
            ↪  &bufferHeight);
```

```
50          glfwMakeContextCurrent(mainWindow);
51          createCallbacks();
52          glfwSetInputMode(mainWindow, GLFW_CURSOR,
            ↪  GLFW_CURSOR_DISABLED);
53
54          glewExperimental = GL_TRUE;
55
56          GLenum error = glewInit();
57          if (error != GLEW_OK)
58          {
59                  printf("Error: %s", glewGetErrorString(error));
60                  glfwDestroyWindow(mainWindow);
61                  glfwTerminate();
62                  return 1;
63          }
64
65          glEnable(GL_DEPTH_TEST);
66          glViewport(0, 0, bufferWidth, bufferHeight);
67          glfwSetWindowUserPointer(mainWindow, this);
68  }
69
70  void Window::createCallbacks()
71  {
72          glfwSetKeyCallback(mainWindow, handleKeys);
73          glfwSetCursorPosCallback(mainWindow, handleMouse);
74  }
75
76  GLfloat Window::getXChange()
77  {
78          GLfloat theChange = xChange;
79          xChange = 0.0f;
80          return theChange;
81  }
82
83  GLfloat Window::getYChange()
84  {
85          GLfloat theChange = yChange;
86          yChange = 0.0f;
87          return theChange;
88  }
89
90  void Window::handleKeys(GLFWwindow* window, int key, int code, int
    ↪  action, int mode)
91  {
92          Window* theWindow =
            ↪  static_cast<Window*>(glfwGetWindowUserPointer(window));
93
94          if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
```

```
95          {
96                  glfwSetWindowShouldClose(window, GL_TRUE);
97          }
98          else if (key >= 0 && key < 1024)
99          {
100                 if (key == GLFW_KEY_B && action == GLFW_PRESS)
101                 {
102                         theWindow->keys[key] =
                            ↪  !theWindow->keys[key];
103                 }
104                 else if (key == GLFW_KEY_B && action ==
                    ↪  GLFW_RELEASE)
105                         return;
106                 else if (action == GLFW_PRESS)
107                 {
108                         theWindow->keys[key] = true;
109                 }
110                 else if (action == GLFW_RELEASE)
111                 {
112                         theWindow->keys[key] = false;
113                 }
114         }
115 }
116
117 void Window::handleMouse(GLFWwindow* window, double xPos, double
    ↪  yPos)
118 {
119         Window* theWindow =
            ↪  static_cast<Window*>(glfwGetWindowUserPointer(window));
120
121         if (theWindow->mouseFirstMoved)
122         {
123                 theWindow->lastX = xPos;
124                 theWindow->lastY = yPos;
125                 theWindow->mouseFirstMoved = false;
126         }
127
128         theWindow->xChange = xPos - theWindow->lastX;
129         theWindow->yChange = theWindow->lastY - yPos;
130
131         theWindow->lastX = xPos;
132         theWindow->lastY = yPos;
133 }
134
135 Window::~Window()
136 {
137         glfwDestroyWindow(mainWindow);
138         glfwTerminate();
```

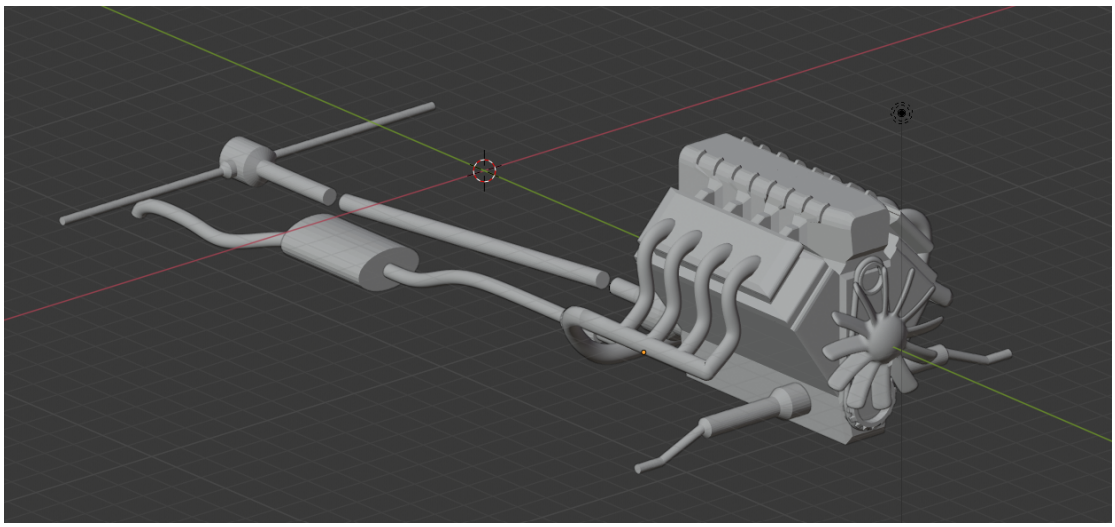
```
}
```

```
#pragma once

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <vector>
#include <sstream>
#include <GL\glew.h>
#include "Window.h"

GLfloat deltaTime = 0.0f;
GLfloat lastTime = 0.0f;
GLfloat timeCounter = 0.0f;
int frameCount = 0;

static const char* vShader = "Shaders/shader.vert";
static const char* fShader = "Shaders/shader.frag";

void computeFPS(GLFWwindow* pWindow)
{
	GLfloat now = glfwGetTime();
	deltaTime = now - lastTime;
	lastTime = now;
	timeCounter += deltaTime;
	frameCount++;

	if (timeCounter >= 1.0) {
		double fps = double(frameCount) / timeCounter;

		std::stringstream ss;
		ss << " [" << fps << " FPS]";

		glfwSetWindowTitle(pWindow, ss.str().c_str());

		frameCount = 0;
		timeCounter = 0;
	}
}

int main()
{
	Window mainWindow(1200, 800);
	mainWindow.Initialize();

	while (!mainWindow.getShouldClose())
	{
```

```
47                  computeFPS(mainWindow.mainWindow);
48
49                  glfwPollEvents();
50
51                  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
52                  glClear(GL_COLOR_BUFFER_BIT |
                    ↪  GL_DEPTH_BUFFER_BIT);
53
54                  glUseProgram(0);
55                  mainWindow.swapBuffers();
56          }
57
58          return 0;
59  }
```

3.2 Utworzenie podłoża

3.3 Stworzenie modeli 3D



Rysunek 3: Podwozie samochodu

3.4 Implementacja modeli w programie

3.5 Ożywienie samochodu

4 Podsumowanie

Rysunek 4: Porównanie oryginału z utworzonym modelem

Literatura

[1] T. Gałaj, *Learn OpenGL*, https://shot511.github.io/pages/learnopengl/