

3. Übungsblatt - C++

Henrik Gerdes, Manuel Eversmeyer

11. November 2018

```
1  /*
2  *   main.c
3  *
4  *   Created on: Nov. 04 2018
5  *       Author: Thomas Wiemann
6  *
7  *   Copyright (c) 2018 Thomas Wiemann.
8  *   Restricted usage. Licensed for participants of the course "The C++
9  *   Programming Language" only.
10 *   No unauthorized distribution.
11 */
12 #include "mainwindow.h"
13 #include "plyio.h"
14
15 int main(int argc, char *argv[])
16 {
17     /* Our SDL_Window ( just like with SDL2 without OpenGL) */
18     SDL_Window *mainWindow;
19
20     /* Our opengl context handle */
21     SDL_GLContext mainContext;
22
23     if (!createSDLWindow(&mainWindow, &mainContext))
24     {
25         return -1;
26     }
27
28     /* Init model struct with default values */
29     Model* model = (Model*) malloc(sizeof(Model));
30     model->numFaces = 0;
31     model->numVertices = 0;
32     model->indexBuffer = NULL;
33     model->vertexBuffer = NULL;
34
35     /*+++++*/
36     /* TODO: Load data from PLY file into model */
37     /*+++++*/
38
39     loadply("models/arrow.ply", model);
40
41
42     /* Call main rendering loop */
43     mainLoop(mainWindow, model);
44 }
```

```

45     /* Free resources */
46     cleanupSDL(mainWindow, mContext);
47
48     /*++++++++++++++++++++++++++++++++++++++++*/
49     /* TODO: Free model data                      */
50     /*++++++++++++++++++++++++++++++++++++++++*/
51     freeModel(model);
52     return 0;
53 }

```

main.c

```

1  /*
2  *  plyio.c
3  *
4  *  Created on: Nov. 04 2018
5  *      Author: Thomas Wiemann
6  *
7  *  Copyright (c) 2018 Thomas Wiemann.
8  *  Restricted usage. Licensed for participants of the course "The C++
9  *  Programming Language" only.
10 *  No unauthorized distribution.
11 */
12 #ifndef __PLYIO_HPP__
13 #define __PLYIO_HPP__
14
15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <string.h>
18
19 #include "model.h"
20
21 /**
22 * @brief Loads a ply-File and creates vertex and index buffer arrays.
23 *
24 * @param file      The name of the ply-File.
25 * @param model     A pointer to model struct that will contain a
26 *                  representation of the loaded file if parsing
27 *                  was successful.
28 */
29 void loadply(
30     char* file, Model* model);
31
32 /**
33 * @brief Frees the allocated memory of the model
34 *
35 * @param model     A pointer to a model struct that will be freed
36 *                  after this method
37 */
38 void freeModel(Model* model);
39
40 #endif

```

plyio.h

```

1  /*
2  *   plyio.c
3  *
4  *   Created on: Nov. 04 2018
5  *       Author: Thomas Wiemann
6  *
7  *   @edited Henrik Gerdes, Manuel Eversmeyer
8  *
9  *   Copyright (c) 2018 Thomas Wiemann.
10 *   Restricted usage. Licensed for participants of the course "The C++
11 *   Programming Language" only.
12 *   No unauthorized distribution.
13 */
14 #include "plyio.h"
15
16
17 void loadply(char *file, Model *model)
18 {
19     FILE* fp;
20
21     fp = fopen(file, "r");
22     char buffer[81] = {0};
23     int canOpen = 0;
24     int isPly = 0;
25     int numVertex = 0;
26     int numFace = 0;
27
28     if (fp == NULL)
29     {
30         canOpen = 1;
31         printf("Fehler beim Laden der der Datei: %s\n", file);
32     }
33
34     //Check Ply
35     fgets(buffer, 80, fp);
36     if (strcmp(buffer, "ply\n") == 0)
37     {
38         isPly = 1;
39     }
40     //Parse header
41     while (strcmp(buffer, "end_header\n") != 0)
42     {
43         fgets(buffer, 80, fp);
44         sscanf(buffer, "element vertex %i", &numVertex);
45         sscanf(buffer, "element face %i", &numFace);
46         //printf("OUT: %s", buffer);
47     }
48
49     // Set 0 Model if any error
50     if (!isPly || canOpen || !numFace || !numVertex)
51     {
52         printf("If Falsche\n");
53         model->numVertices = 0;
54         model->numFaces = 0;
55         model->vertexBuffer = 0;
56         model->indexBuffer = 0;

```

```

57     }
58     else
59     {
60         // Allocate memory
61         float* vertexBuffer = (float*) malloc(3 * numVertex * sizeof(float)
        );
62         int* indexBuffer = (int*) malloc(3 * numFace * sizeof(int));
63
64         if (vertexBuffer == NULL || indexBuffer == NULL)
65         {
66             printf("Interner Fehler: Speicher konnte nicht reserviert
        werden.\n");
67             exit(1);
68         }
69
70         // Save info to model
71         model->vertexBuffer = vertexBuffer;
72         model->indexBuffer = indexBuffer;
73         model->numFaces = numFace;
74         model->numVertices = numVertex;
75
76         // Read all Vertex
77         fread(vertexBuffer, sizeof(float), 3*numVertex, fp);
78         int count;
79         char buf;
80         // Read all Index
81         for (count = 0; count < model->numFaces; count++)
82         {
83             fread(&buf, sizeof(unsigned char), 1, fp);
84             fread(&indexBuffer[3 * count], sizeof(int), 1, fp);
85             fread(&indexBuffer[3 * count+1], sizeof(int), 1, fp);
86             fread(&indexBuffer[3 * count+2], sizeof(int), 1, fp);
87         }
88         // fread(indexBuffer, sizeof(int), 3*numFace, fp);
89     }
90
91     fclose(fp);
92
93 }
94
95
96 void freeModel(Model* model)
97 {
98     free(model->indexBuffer);
99     free(model->vertexBuffer);
100
101     model->numFaces = 0;
102     model->numVertices = 0;
103 }

```

plyio.c

```

1  /*
2  *  mainwindow.c
3  *
4  *  Created on: Nov. 04 2018
5  *      Author: Thomas Wiemann
6  *

```

```

7  * Copyright (c) 2018 Thomas Wiemann.
8  * Restricted usage. Licensed for participants of the course "The C++
   * Programming Language" only.
9  * No unauthorized distribution.
10 */
11
12 /* C library includes */
13 #include <stdio.h>
14
15 /* Project includes */
16 #include "mainwindow.h"
17
18 int createSDLWindow(SDL_Window** mainWindow, SDL_GLContext* mainContext)
19 {
20     /* Initialize SDL's Video subsystem */
21     if (SDL_Init(SDL_INIT_VIDEO) < 0)
22     {
23         printf("Failed to init SDL\n");
24         return 0;
25     }
26
27     /* Create our window centered at 512x512 resolution */
28     *mainWindow = SDL_CreateWindow("SDL Main Window",
29                                     SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
30                                     1024, 768, SDL_WINDOW_OPENGL);
31
32     /* Check that everything worked out okay */
33     if (!mainWindow)
34     {
35         printf("Unable to create window\n");
36         printSDLError(__LINE__);
37         return 0;
38     }
39
40     /* Create our opengl context and attach it to our window */
41     *mainContext = SDL_GL_CreateContext(*mainWindow);
42
43     /* Set our OpenGL version.
44        SDL_GL_CONTEXT_CORE gives us only the newer version, deprecated
45        functions are disabled */
46     SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,
47                         SDL_GL_CONTEXT_PROFILE_CORE);
48
49     /* 3.2 is part of the modern versions of OpenGL, but most video cards
50        whould be able to run it */
51     SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 3);
52     SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 2);
53
54     /* Turn on double buffering with a 24bit Z buffer.
55        You may need to change this to 16 or 32 for your system */
56     SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
57
58     /* This makes our buffer swap synchronized with the monitor's vertical
59        refresh */
60     SDL_GL_SetSwapInterval(1);
61
62     /* Init GLEW */
63     #ifndef __APPLE__

```

```

59     glewExperimental = GL_TRUE;
60     glewInit();
61 #endif
62
63     SDL_GL_SwapWindow(*mainWindow);
64
65     /* Init OpenGL projection matrix */
66     glClearColor(0.0, 0.0, 0.0, 1.0);
67     float ratio = 1024 * 1.0 / 768;
68     glMatrixMode(GL_PROJECTION);
69     glLoadIdentity();
70     glViewport(0, 0, 1027, 768);
71     gluPerspective(45, ratio, 1, 10000);
72
73     /* Ender model view mode */
74
75     glMatrixMode(GL_MODELVIEW);
76
77     return 1;
78 }
79
80 void mainLoop(SDL_Window *mainWindow, Model *model)
81 {
82     int loop = 1;
83     int i, a, b, c, index;
84
85     /* Set camera position and direction */
86     glLoadIdentity();
87     gluLookAt(0.0, 0.0, -750.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0);
88
89     if (model && model->indexBuffer && model->vertexBuffer)
90     {
91
92         while (loop)
93         {
94
95             glClear(GL_COLOR_BUFFER_BIT);
96
97             SDL_Event event;
98             while (SDL_PollEvent(&event))
99             {
100                 /* Check if window has been closed */
101                 if (event.type == SDL_QUIT)
102                 {
103                     loop = 0;
104                 }
105             }
106
107             for(i = 0; i<model->numFaces; i++)
108             {
109                 index = i * 3;
110                 a = 3 * model->indexBuffer[index];
111                 b = 3 * model->indexBuffer[index+1];
112                 c = 3 * model->indexBuffer[index+2];
113
114                 glBegin(GL_LINE_LOOP);
115                 glVertex3f(model->vertexBuffer[a], model->vertexBuffer[a
+1], model->vertexBuffer[a+2]);

```

```

116         glVertex3f(model->vertexBuffer[b], model->vertexBuffer[b
117             +1], model->vertexBuffer[b+2]);
118         glVertex3f(model->vertexBuffer[c], model->vertexBuffer[c
119             +1], model->vertexBuffer[c+2]);
120         glEnd();
121     }
122     /* Bring up back buffer */
123     SDL_GL_SwapWindow(mainWindow);
124 }
125 }
126
127 void cleanupSDL(SDL_Window* mainWindow, SDL_GLContext* mainContext)
128 {
129     /* Delete our OpenGL context */
130     SDL_GL_DeleteContext(mainContext);
131
132     /* Destroy our window */
133     SDL_DestroyWindow(mainWindow);
134
135     /* Shutdown SDL 2 */
136     SDL_Quit();
137 }
138
139 void printSDLError(int line)
140 {
141     const char* error = SDL_GetError();
142
143     if (error != "")
144     {
145         printf("SLD Error : %s\n", error);
146
147         if (line != -1)
148         {
149             printf("Line : %d\n", line);
150         }
151         SDL_ClearError();
152     }
153 }

```

mainwindow.c

```

1 INF-C++ Blatt03
2
3 @author Henrik Gerdes, Manuel Eversmeyer
4
5 Aufgabe3.1:
6
7 Vorteile von CMake:
8 Breite Unterstuetzung:      Viele IDEs nutzen es
9
10 Plattformunabhaengigkeit:  Anders als Make laeuft CMake auf jedem System,
11                             da es nur C(C++) als Vorraussetzung hat. (Make unterstuetzt Win
12                             offiziell nicht)
13
14 Metabuildsystem:           CMake liefert entweder ein Makefile oder
15                             Buildanweisungen fuer VisualStudio, XCode oder Eclipse und weitere.

```

```

16
17 Erweiterbarkeit:      Unterstuetzung von Makros
18
19 Anpassbarkeit:      Unterstuetzung verschiedener Compiler und
20      waehlt die je nach Dateieindung
21
22 Modular/Rekursiv:    Man kann CMake anweisen Abhaengigkeiten zu
23      Bilbliotheken aufzuloesen. Dazu sucht CMake nach weiteren
24      CMakeLists.txt Dateien und fuehrt diese aus
25
26 Wichtige Funktionen:
27 Automatisierte Erzeugung von Makefiles und Projekten fuer viele
28 unterschiedliche Entwicklungsumgebungen und Compiler
29 Verschiedene Ziele: ReleaseVersion fuer Windows/Linux/Mac,
30 Debug version. Errechtbar mit verschiedenen Flags
31 Auflösen und suchen von Abhaengigkeiten. Kann auf dem
32 Nutzerrechner verschiedene Verzeichnisse nach Biblotheken durchsuchen
33
34 Aufbau der CMake-Projekte:
35 -Minimum CMake Version
36 -Projektname
37 -Verzeichnisse die gelinkt oder inkludiert werden muessen
38 -Pfad in dem nach lokalen cmake modulen gesucht wird
39 -Compiler-Flags
40 -Suche nach benoetigte Bibliotheken
41 -Aufzaehlung der Source-Dateien in einer Variable
42 -Erzeugung einer Executeable
43 -Abhaengigkeiten von externen Bibliotheken
44
45 Unterschied In-Source und Out-of-Source:
46 Bei In-Source builds werden die Source-Datei und Executeable-Dateien im
47 selben Verzeichnis gespeichert. Bei Out-Source builds werden die
48 Source-Datei und Executeable-Dateien in seperaten Verzeichnissen
49 gespeichert.
50
51 In-Source builds haben gegenueber Out-Source builds so gut wie keine
52 Vorteile, daher verwendet man wenn moeglich Out-Source builds.

```

3_1.txt