2. Übungsblatt - C++

Henrik Gerdes, Manuel Eversmeyer

11. November 2018

```c
/*
 *  main.c
 *
 *  Created on: Nov. 04 2018
 *      Author: Thomas Wiemann
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#include "mainwindow.h"
#include "plyio.h"

int main(int argc, char *argv[])
{
    /* Our SDL_Window ( just like with SDL2 wihout OpenGL) */
    SDL_Window *mainWindow;

    /* Our opengl context handle */
    SDL_GLContext mainContext;

    if (!createSDLWindow(&mainWindow, &mainContext))
    {
        return -1;
    }

    /* Init model struct with default values */
    Model* model = (Model*) malloc(sizeof(Model));
    model->numFaces = 0;
    model->numVertices = 0;
    model->indexBuffer = NULL;
    model->vertexBuffer = NULL;

    /*+++++++++++++++++++++++++++++++++++++++++++++++++*/
    /* TODO: Load data from PLY file into model */
    /*+++++++++++++++++++++++++++++++++++++++++++++++++*/

    loadply("models/arrow.ply", model);


    /* Call main rendering loop */
    mainLoop(mainWindow, model);

    /* Free resources */
    cleanupSDL(mainWindow, mainContext);

    /*++++++++++++++++++++++++++++++++++++++++++++*/
    /* TODO: Free model data                      */
    /*++++++++++++++++++++++++++++++++++++++++++++*/
    freeModel(model);
    return 0;
}
```

main.c

```c
/*
 *  plyio.c
 *
 *  Created on: Nov. 04 2018
 *      Author: Thomas Wiemann
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#ifndef __PLYIO_HPP__
#define __PLYIO_HPP__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "model.h"

/**
 * @brief Loads a ply-File and creates vertex and index buffer arrays.
 *
 * @param file      The name of the ply-File.
 * @param model     A pointer to model struct that will contain a
 *                  representation of the loaded file if parsing
 *                  was successful.
 */
 void loadply(
        char* file, Model* model);

/**
 * @brief Frees the allocated moemory of the model
 *
 * @param model     A pointer to a model struct that will be freed
 *                  after this method
 */
 void freeModel(Model* model);

#endif
```

plyio.h

```c
/*
 *  plyio.c
 *
 *  Created on: Nov. 04 2018
 *      Author: Thomas Wiemann
 *
 * @edited Henrik Gerdes, Manuel Eversmeyer
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#include "plyio.h"


void loadply(char *file, Model *model)
{
    FILE* fp;

    fp = fopen(file, "r");
    char buffer[81] = {0};
    int canOpen = 0;
    int isPly = 0;
    int numVertex = 0;
    int numFace = 0;

    if(fp == NULL)
    {
        canOpen = 1;
        printf("Fehler beim Laden der der Datei: %s\n", file);
    }

    //Check Ply
    fgets(buffer,80,fp);
    if(strcmp(buffer,"ply\n") == 0)
    {
        isPly = 1;
    }
    //Parse header
    while(strcmp(buffer,"end_header\n") !=0)
    {
        fgets(buffer,80,fp);
        sscanf(buffer, "element vertex %i", &numVertex);
        sscanf(buffer, "element face %i", &numFace);
        //printf("OUT: %s", buffer);
    }

    // Set 0 Model if any error
    if(!isPly || canOpen || !numFace || !numVertex)
    {
        printf("If Falsche\n");
        model->numVertices = 0;
        model->numFaces = 0;
        model->vertexBuffer = 0;
        model->indexBuffer = 0;
    }
    else
    {
        //Allocate memory
        float*  vertexBuffer = (float*)malloc(3 * numVertex * sizeof(float)
            );
        int* indexBuffer = (int*)malloc(3 * numFace * sizeof(int));

        if(vertexBuffer == NULL || indexBuffer == NULL)
        {
            printf("Interner Fehler: Speicher konnte nicht reserviert
                werden.\n");
            exit(1);
        }

        //Save info to model
        model->vertexBuffer = vertexBuffer;
        model->indexBuffer = indexBuffer;
        model->numFaces = numFace;
        model->numVertices = numVertex;

        //Read all Vertex
        fread(vertexBuffer, sizeof(float), 3*numVertex, fp);
        int count;
        char buf;
        //Read all Index
        for(count = 0; count<model->numFaces;count++)
        {
            fread(&buf, sizeof(unsigned char), 1, fp);
            fread(&indexBuffer[3 * count], sizeof(int), 1, fp);
            fread(&indexBuffer[3 * count+1], sizeof(int), 1, fp);
            fread(&indexBuffer[3 * count+2], sizeof(int), 1, fp);
        }
        //fread(indexBuffer, sizeof(int), 3*numFace, fp);
    }

    fclose(fp);

}


void freeModel(Model* model)
{
    free(model->indexBuffer);
    free(model->vertexBuffer);

    model->numFaces = 0;
    model->numVertices = 0;
}
```

plyio.c

```c
/*
 *  mainwindow.c
 *
 *  Created on: Nov. 04 2018
 *      Author: Thomas Wiemann
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++ Programming Language" only.
 *  No unauthorized distribution.
 */

/* C library includes */
#include <stdio.h>

/* Project includes */
#include "mainwindow.h"

int createSDLWindow(SDL_Window** mainWindow, SDL_GLContext* mainContext)
{
    /* Initialize SDL's Video subsystem */
    if (SDL_Init(SDL_INIT_VIDEO) < 0)
    {
        printf("Failed to init SDL\n");
        return 0;
    }

    /* Create our window centered at 512x512 resolution */
    *mainWindow = SDL_CreateWindow("SDL Main Window", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
                                   1024, 768, SDL_WINDOW_OPENGL);

    /* Check that everything worked out okay */
    if (!mainWindow)
    {
        printf("Unable to create window\n");
        printSDLError(__LINE__);
        return 0;
    }

    /* Create our opengl context and attach it to our window */
    *mainContext = SDL_GL_CreateContext(*mainWindow);

    /* Set our OpenGL version.
       SDL_GL_CONTEXT_CORE gives us only the newer version, deprecated functions are disabled */
    SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK, SDL_GL_CONTEXT_PROFILE_CORE);

    /* 3.2 is part of the modern versions of OpenGL, but most video cards whould be able to run it */
    SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 3);
    SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 2);

    /* Turn on double buffering with a 24bit Z buffer.
       You may need to change this to 16 or 32 for your system */
    SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);

    /* This makes our buffer swap syncronized with the monitor's vertical refresh */
    SDL_GL_SetSwapInterval(1);

/* Init GLEW */
#ifndef __APPLE__
```

```
    glewExperimental = GL_TRUE;
    glewInit();
#endif

    SDL_GL_SwapWindow(*mainWindow);

    /* Init OpenGL projection matrix */
    glClearColor(0.0, 0.0, 0.0, 1.0);
    float ratio = 1024 * 1.0 / 768;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, 1027, 768);
    gluPerspective(45, ratio, 1, 10000);

    /* Ender model view mode */

    glMatrixMode(GL_MODELVIEW);

    return 1;
}

void mainLoop(SDL_Window *mainWindow, Model *model)
{
    int loop = 1;
    int i, a, b, c, index;

    /* Set camera position and direction */
    glLoadIdentity();
    gluLookAt(0.0, 0.0, -750.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0);

    if (model && model->indexBuffer && model->vertexBuffer)
    {

        while (loop)
        {

            glClear(GL_COLOR_BUFFER_BIT);

            SDL_Event event;
            while (SDL_PollEvent(&event))
            {
                /* Check if window has been closed */
                if (event.type == SDL_QUIT)
                {
                    loop = 0;
                }
            }

            for(i = 0; i<model->numFaces; i++)
            {
                index = i * 3;
                a = 3 * model->indexBuffer[index];
                b = 3 * model->indexBuffer[index+1];
                c = 3 * model->indexBuffer[index+2];

                glBegin(GL_LINE_LOOP);
                glVertex3f(model->vertexBuffer[a], model->vertexBuffer[a
                    +1], model->vertexBuffer[a+2]);
```

```c
                    glVertex3f(model->vertexBuffer[b], model->vertexBuffer[b
                        +1], model->vertexBuffer[b+2]);
                    glVertex3f(model->vertexBuffer[c], model->vertexBuffer[c
                        +1], model->vertexBuffer[c+2]);
                    glEnd();
                }

                /* Bring up back buffer */
                SDL_GL_SwapWindow(mainWindow);
            }
        }
}

void cleanupSDL(SDL_Window* mainWindow, SDL_GLContext* mainContext)
{
    /* Delete our OpengL context */
    SDL_GL_DeleteContext(mainContext);

    /* Destroy our window */
    SDL_DestroyWindow(mainWindow);

    /* Shutdown SDL 2 */
    SDL_Quit();
}

void printSDLError(int line)
{
    const char* error = SDL_GetError();

    if (error != "")
    {
        printf("SLD Error : %s\n", error);

        if(line != -1)
        {
            printf("Line : %d\n",line);
        }
        SDL_ClearError();
    }
}
```

mainwindow.c

```
INF-C++ Blatt03

@author Henrik Gerdes, Manuel Eversmeyer

Aufgabe3.1:

Vorteile von CMake:
Breite Unterstuetzung:      Viele IDEs nutzen es

Plattformunabhaengigkeit:   Anders als Make laeuft CMake auf jedem System,
    da es nur C(C++) als Vorraussetzung hat. (Make unterstuetzt Win
    offiziell nicht)

Metabuildsystem:        CMake liefer entweder ein Makefile oder
    Buildanweisungen fuer VisualStudio, XCode oder Eclipse und weitere.
```

```

Erweiterbarkeit:         Unterstuetzung von Makros

Anpassbarkeit:           Unterstuetzung verschiedener Compiler und
    waehlt die je nach Dateiendung

Modular/Recursiv:        Man kann CMake anweisen Abhaengigkeiten zu
    Bilbliotheken aufzuloesen. Dazu sucht CMake nach weiteren
    CMakeLists.txt Dateien und fuehrt diese aus

Wichtige Funktionen:
Automatisierte Erzeugung von Makefiles und Projekten fuer viele
unterschiedliche Entwicklungsumgebungen und Compiler
Verschiedene Ziele: ReleaseVersion fuer Windows/Linux/Mac,
Debug version. Errechtbar mit verschiedenen Flags
Aufloesen und suchen von Abhaengigkeiten. Kann auf dem
Nutzerrechner verschiede Verzeichnisse nach Biblotheken durchsuchen

Aufbau der CMake-Projekte:
-Minimum CMake Version
-Projektname
-Verzeichnisse die gelinkt oder inkludiert werden muessen
-Pfad in dem nach lokalen cmake modulen gesucht wird
-Compiler-Flags
-Suche nach benoetigte Bibliotheken
-Aufzaehlung der Source-Dateien in einer Variable
-Erzeugung einer Executeable
-Abhaengigkeiten von externen Bibliotheken

Unterschied In-Source und Out-of-Source:
Bei In-Source builds werden die Source-Datei und Executeable-Dateien im
selben Verzeichnis gespeichert. Bei Out-Source builds werden die
Source-Datei und Executeable-Dateien in seperaten Verzechnissen
gespeichert.

In-Source builds haben gegenueber Out-Source builds so gut wie keine
Vorteile, daher verwendet man wenn moeglich Out-Source builds.
```

3_1.txt