Einführung in C++ – Übung 4
Testatgruppe A (Isaak)

Rasmus Diederichsen

5. November 2014

Listing 1: CMakeLists.txt

```
1  #
       ##############################################################################
2  # Declare the minimum cmake version required
3  #
       ##############################################################################
4  cmake_minimum_required(VERSION 2.8)
5
6  #
       ##############################################################################
7  # The name of out project
8  #
       ##############################################################################
9  project(LSSR)
10
11 #
       ##############################################################################
12 # Now you would normally declare additional linker and include
       directories.
13 # We don't need this at this moment, just to show you how its done
14 #
       ##############################################################################
15
16 #link_directories("${CMAKE_SOURCE_DIR}/lib")
17 #link_directories("$ENV{HOME}/local/lib")
18
19
20 #
       ##############################################################################
21 # With cmake we can define additional compiler flags for different
22 # configurations. CMAKE_CXX_FLAGS are for the default case. The
       debug config
```

```
23 # can be used to generate debug symbols for gdb. The release option
       uses
24 # special optimization flags
25 #
       ################################################################################

26
27 set( CMAKE_C_FLAGS_RELEASE "-O3 -msse3 -Wno-unused -Wno-deprecated"
       )
28 set( CMAKE_C_FLAGS_DEBUG   "-g -Wall -DDEBUG -Wno-deprecated" )
29
30 #
       ################################################################################

31 # Find required libraries.Right now we need glut and OpenGL. The
       required
32 # options forces this packages to be present. For non-mandatory
       packages you
33 # can leave this flag out. The can then check if they were found by
       using
34 # IF(OpenGL_found) etc.
35 #
       ################################################################################

36
37 FIND_PACKAGE(OPENGL REQUIRED)
38 FIND_PACKAGE(GLUT REQUIRED)
39
40 if(OPENGL_FOUND)
41     link_directories(${OPENGL_LIBRARY_DIRS})
42     include_directories(${OPENGL_INCLUDE_DIR})
43 endif(OPENGL_FOUND)
44
45 if(GLUT_FOUND)
46     link_directories(${GLUT_LIBRARY_DIR})
47     include_directories(${GLUT_INCLUDE_DIR})
48 endif(GLUT_FOUND)
49
50 #get_cmake_property(_v VARIABLES)
51 #foreach(_v ${_v})
52 #   message(STATUS "${_v}=${${_v}}")
53 #endforeach()
54
55 #
       ################################################################################

56 # Variable for the sources of the binary. In larger projects it is
       often a
57 # good idea to use such variables because it is easy to add more
       source files
58 # and you can reuse it for several targets (maybe you want to build
       a library
59 # some day ;-)
60 #
       ################################################################################

61
```

```
set(VIEWER_SOURCES
    mainwindow.c
    main.c
    objio.c
)

#
    ################################################################################

# The executable fpr our project
#
    ################################################################################


add_executable(viewer ${VIEWER_SOURCES})

#
    ################################################################################

# External library dependencys
#
    ################################################################################


target_link_libraries(viewer ${GLUT_LIBRARY} ${OPENGL_LIBRARY})
```

Listing 2: objio.c

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

void set_null(float **vertexBuffer, int **indexBuffer, int *
    vertexCount, int *faceCount)
{
   *vertexBuffer = NULL;
   *indexBuffer = NULL;
   *vertexCount = 0;
   *faceCount = 0;
}

void loadObj(
      char* file,
      float** vertexBuffer,
      int** indexBuffer,
      int* vertexCount,
      int* faceCount)
{

   FILE *model_file = fopen(file, "r");
   if (model_file == NULL) {
      set_null(vertexBuffer, indexBuffer, vertexCount, faceCount);
      return;
   }

   char vertex_format[] = "v␣%f␣%f␣%f";
   char face_format[] = "f␣%d␣%d␣%d";
   char line_read[50];

```

```
30      *vertexCount = *faceCount = 0;
31
32      while (fgets(line_read, 50, model_file) != NULL)
33      {
34         switch(line_read[0])
35         {
36            case '\n':
37            case '#': break;
38            case 'f': (*faceCount)++; break;
39            case 'v': (*vertexCount)++; break;
40            default: /* invalid format */
41                     set_null(vertexBuffer, indexBuffer, vertexCount,
                              faceCount);
42                     return;
43
44         }
45      }
46
47      *vertexBuffer = malloc(*vertexCount * 3 * sizeof(float));
48      *indexBuffer = malloc(*faceCount * 3 * sizeof(int));
49
50      /* indices for both arrays */
51      int i, j, line, error = 0;
52      i = j = line = 0;
53
54      rewind(model_file);
55      while (!error && fgets(line_read,50,model_file) != NULL)
56      {
57         line++;
58         printf("%d: %s\n",line,line_read);
59         int read;
60         switch (line_read[0])
61         {
62            case '\n':
63            case '#': break;
64            case 'v': read = sscanf(line_read, vertex_format,
65                                    (*vertexBuffer)+i,
66                                    (*vertexBuffer)+i+1,
67                                    (*vertexBuffer)+i+2);
68                      i += 3;
69                      if(!read) error = 1;
70                      break;
71            case 'f': read = sscanf(line_read, face_format,
72                                    (*indexBuffer)+j,
73                                    (*indexBuffer)+j+1,
74                                    (*indexBuffer)+j+2);
75                      j += 3;
76                      if(!read) error = 1;
77                      break;
78            default: error = 1; break;
79         }
80      }
81      if(error)
82      {
83         fprintf(stderr, "Error. No valid .obj file.\nError at line %d
               : %s\n",line,line_read);
84         set_null(vertexBuffer, indexBuffer, vertexCount, faceCount);
```

```
85          return;
86      }
87      fclose(model_file);
88  }
89
90  void freeObj(float **vertexBuffer, int **indexBuffer)
91  {
92      free(*vertexBuffer);
93      free(*indexBuffer);
94  }
```