Prof. Dr. F. Sadlo, M.Sc. Y. Agapov, M. Tabachnik *Heidelberg, January 08, 2020*

Exercise Sheet 9

Assignment 9.1 HelloGL [7.5+3 Bonus Points]

This exercise introduces the basics for the use of OpenGL, by means of the OGL4Core framework. You can download the framework code directly from the project page[1]. Please use the appropriate `src` (not the `bin`) version for your platform. The given code skeleton from Moodle contains the OGL4Core framework for Ubuntu, where the code for this exercise is located in `OGL4Core/Plugins/MyPlugins`. The framework provides many simplifciations for using OpenGL, of which we will use particularly its functionality for accessing the graphics hardware. This exercise provides the necessary techniques for the development of standalone programs.

After extracting the code, you can list all plugins with `./ogl4coreGLUT64d -show`. The given plugin is located in `Plugins/MyPlugins`. Compile the plugin code (for Linux, simply type `make` in the `Plugins` folder). Start the Plugin `HelloGL` with `./ogl4coreGLUT64d -rid` n, where you should replace n with the appropriate plugin number. For now, a blue background and a gray ground should be visible. All necessary functions are located in `HelloGL.cpp`, unless specified otherwise.

a) **Model View Projection**
As discussed in the lecture, the rendering of three-dimensional geometry with local / global coordinates is influenced by a number of transformations. Aside from the so-called model matrix (positioning of the objects) and projection matrix ("mapping" of 3D coordinates to "2D" normalized device coordinates), there is also the view matrix, which defines the camera. In this task, you should calculate the camera position with regard to the user input, as well as the viewing direction of the according view matrix. For this, alter the function `HelloGL::AssembleTransformMatrices`. With the keys *w, a, s*, and *d*, as well as the mouse (left mouse button + drag), you can alter the viewing direction. [*3.5 Points*]

Hint: The buttons *w, a, s*, and *d* should change the camera position horizontally (forward, backward, or sideward, respectively). The mouse should change the viewing direction. Note that there are different approaches as to which center of rotation you use. Preferably, implement a FPS-like tracking, i.e., with rotation around the camera.

b) **Drawing the Geometry**
Define the necessary data for a *two-dimensional* geometry of a tree. The z-component is always 0. For this, you need to define the vertex, index, and color data in the appropriate OpenGL buffers. The data should be described in the function `GenerateTreeMesh`. The exact geometry is left to your imagination (see comments in the source code for further help). Once the geometry is in the main memory,

[1] `https://www.vis.uni-stuttgart.de/projekte/ogl4core-framework`

the necessary OpenGL buffers have to be allocated. This should be done in the function `GenerateTree`. If you have difficulties with the used syntax, you can, for example, look at the definition of the ground area. [*4 Points*]

c) **BONUS TASK: Instanciated Geometry and Billboarding**
Extend the drawn tree with instanciated calls. For this, alter the function call in `HelloGL::Render`. The positions of the trees are already generated by the skeleton and are available in the GPU memory (see `GenerateTreePositions`).

Extend the shader in `HelloGL/resources/Shaders/tree-vertex.txt` by a "billboarding" effect. Here the tree should be rotated in such a fashion that the normal of the (flat) geometry will always point toward the camera. Remember to only rotate the tree around its vertical axis! [*3 Points*]

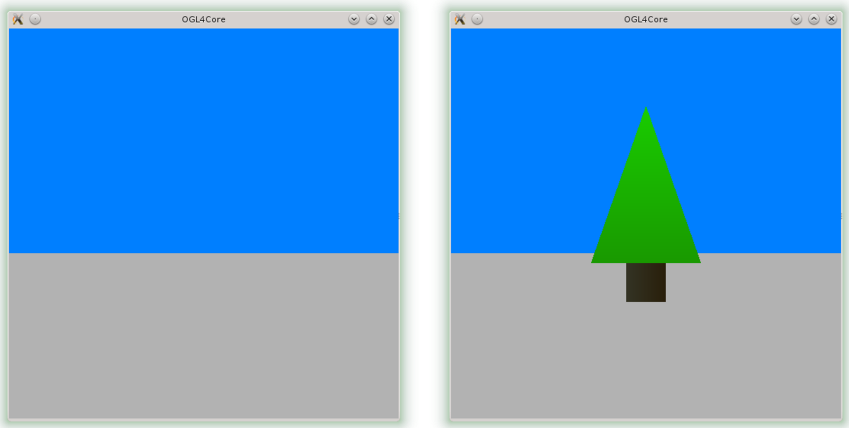

Figure 1: Starting geometry (left) and exemplary tree (right).

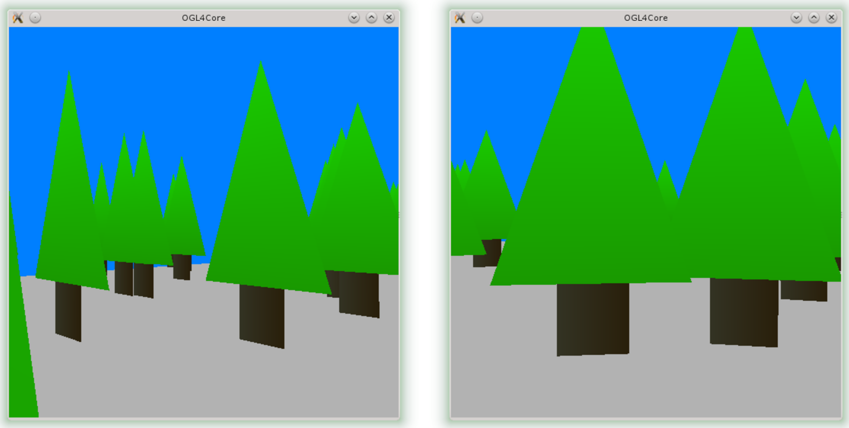

Figure 2: Forest without (left) and with billboarding (right).

Submission: January 21, 2020, 14:15 CEST, via Moodle