TU BRAUNSCHWEIG
Dr. Georgia Albuquerque
Institut für Computergraphik
Matthias Überheide

April 20, 2018

Real-time Computer Graphics, Summer 2018
Assignment 2

Present your solution to this exercise on Thursday, May 03, 2018.

The exercises are going to take place in the CIP pool, room G40 in Mühlenpfordstraße 23. Please make sure your solutions compile and run on the CIP pool computers. Note that you need a y-number, which can be obtained at the Gauß-IT-Zentrum, to use the computers. If for some reason you are not able to attend the exercise, you may send your solution to ecg@cg.cs.tu-bs.de instead.

This exercise and the corresponding version of the framework can be found on the lectures website (`http://www.cg.cs.tu-bs.de/teaching/lectures/ss18/ecg/`).

Theoretical Tasks

2.1 OpenGL Draw Calls (20 Points)

Given a VertexArrayObject `VAO` and an IndexArrayBuffer `IBO` of length 9. Explain how many and which primitives are drawn by the following OpenGL commands.
a)

```
glBindVertexArray(VAO);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
glDrawElements(GL_LINES, 6, GL_UNSIGNED_INT, (void*)0);
```

b)

```
glBindVertexArray(VAO);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
glDrawElements(GL_POINTS, 6, GL_UNSIGNED_INT, (void*)0);
```

c)

```
glBindVertexArray(VAO);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
glDrawElements(GL_TRIANGLE_STRIP, 6, GL_UNSIGNED_INT, (void*)0);
```

d)

```
glBindVertexArray(VAO);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (void*)0);
```

e)

```
glBindVertexArray(VAO);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
glDrawElements(GL_TRIANGLES, 12, GL_UNSIGNED_INT, (void*)0);
```

Practical Tasks

2.2 Load a 3D mesh from an OBJ file (30 Punkte)

Implement the missing methods in `OBJWavefrontFile.cpp`. The purpose of this class is to open a Wavefront OBJ file (`http://en.wikipedia.org/wiki/Wavefront_.obj_file`) containing vertex, texture coordinate, normal and face data for a 3D mesh object and to import data into a vertex list, a texture coordinate list, a normal list and a list of vertex indices used to form faces (defined as triangles). OBJ allows the use of different indices for vertex coordinate, texture coordinate and normal. However, rendering with OpenGL requires one index per corner. Take a look at `OBJMeshGenerator` in `OBJWavefrontFile.cpp` and implement the missing methods to create such vertex definitions. To avoid duplicates use a map to keep everything memory efficient. As OBJ is not restricted to triangle faces the mesh generator should employ a method to triangulate polygons (at least quads). Experiment with other OBJ files.

2.3 Draw geometry using Vertex Array Objects (30 Points)

In this exercise you will upload geometry to the GPU. Similar to the test triangle buffers need to be created. To add some colors the normals will be used as source for color. If you were unable to create your own object loader you can use the prepared loader results in the `test` namespace. Implement the missing parts within the classes `MeshObject` and `ColoredMeshObject`. In order to render the monkey later on, you need to setup a buffer structure that can be rendered by OpenGL. We will use a *Vertex Array Object* (or short VAO) for this, as presented in the lecture.
In order to set up the VAO properly, we need to organize the data it will hold. Since we have vertex indices available by `faces` in `OBJWavefrontMeshDataGenerator`, we can render the data as a *Vertex Buffer Object* (VBO) indexed by a *Index Array Object*.
First fill internal buffers `vertexData` and `faceData`. Then create and upload the data to VBO and IBO:

- First, put your data into `vertexData` (`ColoredMeshObject` should add the vertex colors after the coordinates).
- Generate a VAO organizing all data and its layout. Use `glGenVertexArrays()` to generate a new Vertex Array Object and store its OpenGL address in the classes member variable, which is already defined.
- Bind your generated VAO to activate it for further configuration (setup of used buffers).
- To hold the actual data, we need a VBO. Generate a new VBO using `glGenBuffers()` and store its handle.
- Bind the new generated VBO as a `GL_ARRAY_BUFFER`.
- Upload the data to the VBO.
- Upload the data to the IBO.
- Enable the defined vertex attribute using `glEnableVertexAttribArray()`.

Now that the geometry is set up properly, it needs to be rendered somehow. Implement the missing parts in `OGLMeshObject::render` to achieve this.