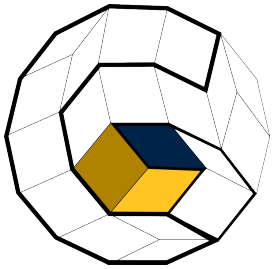
TU BRAUNSCHWEIG
Dr.-Ing. Martin Eisemann
Institut für Computergraphik
Lorenz Rogge (rogge@cg.tu-bs.de)

June 4, 2012



Echtzeit Computergraphik SS 2012
Assignment 7

Present your solution to this exercise on Monday, June 11th, 2012.

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 rogge@cg.tu-bs.de instead.

In this assignment you will extend the already existing version of the OBJLoader to a more flexible version. This version will then be able to load mesh definitions including texture coordinates. Using a mesh like this, you will be able to render a textured object in the second task of this exercise.

7.1 Advanced OBJ Loader (20 Points)

To be able to render properly textured meshes, one has to import this information from a given file. The *Wavefront OBJ* format is able to define various vertex attributes for a mesh. So in this task the `ObjLoader` of exercise 3 has to be extended to support texture coordinates as a vertex attribute.
Like a vertex normal, a texture coordinate is defined in a given .obj file by a line starting with the token `vt` followed by two floating point values. These are then indexed within a face definition in the same file.

In the last exercises, we used .obj files where the vertex index and the vertex normal index was identical. Thus a face definition like `f 0 1 2` would properly define, which vertex positions and which normal vectors should be used for rendering. Unfortunately, this is not always the case. Especially when using texture coordinates. Neighboring triangles, that share at most two vertices, could be textured differently. This would require to use the same vertex position for these triangles, but *different* texture coordinates. This is why the full definition of *Wavefront OBJ* uses a different way to index values for a face definition.
Valid face definitions in a file could be:

- `f v0 v1 v2 (v3)`
- `f v0//n0 v1//n1 v2//n2 (v3//n3)`
- `f v0/t0/ v1/t1/ v2/t2/ (v3/t3/)`
- `f v0/t0/n0 v1/t1/n1 v2/t2/n2 (v3/t3/n3)`

Where `v0...v3` are vertex indices, `t0...t3` are (possibly different) texture coordinate indices, and `n0...n3` are normal indices, that may be completely different again.
Also, as you can see in the format examples above, a face may also consist of *four* vertices, which means the face is a quad. Whenever importing a quad face, simply subdivide it into two triangles. Your VAO then only has to care about rendering of triangular faces.

Look into the provided `ObjLoader.cpp` and complete the missing parts within the method `loadObjFile()`. Read in all values from the file, as you did in exercise 3, but this time use local

storage lists instead of directly inputting the data into a `MeshData` container. You will need to reformat the data before doing this.
When it comes to reading in a face definition, enable the loader to import any of the four definition formats mentioned above. When you have imported a face that uses four vertices instead of three, create two triangular faces from this quad face and store it in your local face list.

Once you finished importing the file, you need to reformat the indexing, which will be used for rendering. Like described above, the used vertex index is not necessarily the same as the vertex normal index or the texture coordinate index. There has to be a unique index for each triplet of these attributes. To when going through your list of faces, you can check if a index-triplet has been used before or is new. If it is new, simply add the vertex position, normal and texture coordinate to the lists within a `MeshData` container and put their index in these lists into the index-list of the container. If, however, the triplet has been used before, just add the previously used index to the index-list and your done.

When you have imported everything properly and you passed this information to the `MeshObj`, you will have to create a `VBO` for the texture coordinates, upload the values to the GPU and enable the proper vertex attribute for texture coordinates in the `VAO` of your `MeshObj`.

7.2 Using Textures (20 Points)

To texture a rendered object properly, you need to pass the texture itself to your shader program and set texture coordinates as vertex attributes. In the previous task you've already setup everything to import texture coordinates and pass them as vertex attributes in form of a VBO to your GPU.
Now you need to create a texture structure, that you can upload to your graphics card and adapt your shader to retrieve the texel information for each rendered fragment.

Figure 1: Textured trash bin shaded by 3 light sources