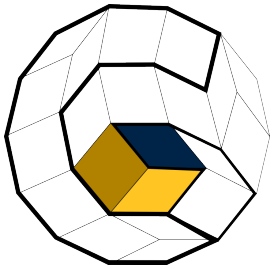
TU BRAUNSCHWEIG
Dipl. Inform. Martin Eisemann
Institut für Computergraphik
Lorenz Rogge (rogge@cg.tu-bs.de)

May 17, 2011



Echtzeit Computergraphik SS 2011
Assignment 5

In this exercise we extend the existing objLoader to import vertex normals and texture coordinates and create a shader that is able to use and display textures. Please present your solution to this exercise on Monday, May 23rd, 2011.

5.1 Advanced objLoader (30 Points)

In the last assignment, you implemented a way to reconstruct normals from vertex data only. Of course this is not a very clever way.
The *Wavefront* OBJ format specifies many more attributes than vertex coordinates and faces using these vertices. Extend the existing `objLoader` to import vertex normals and texture coordinates. To reduce the file size, vertices are only defined once, but may be combined with several different attributes, when used in a face definition. These attiributes are indexed themselves, but *do not* use the same indices as the vertices. So when importing a face definition, one must be careful to use the correct vertex/vertex-normal/texture-coordinate combination. When encountering vertices using *different* normals and/or texture coordinates, clone these vertices and use their copies in further face definitions. Make sure that for every vertex/vertex-normal/texture-coordinate triplet there is one instance, that may be indexed for rendering.
Having a loader like this, it is now possible to load vertex attributes from an .obj file and still be able to use this data in vertex arrays referenced by a single index array.
Additionally allow the objLoader to load quads as faces. Split these quads up into two triangles directly when importing them.
So 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)`

5.2 Textures in OpenGL and GLSL (20 Points)

First, implement the method `loadTextureData()` to load a texture from a file into a simple array of `unsigned char`. This data can them be passed to OpenGL texture object later on. You may use *OpenCV* to do this job, since it provides methods for loading several image file types such as png, jpeg and so on. (Hint: `cvLoadImage()`).
When having the texture data successfully loaded, implement the setup and use of textures in your project `Ex05`. Setup three different textures using different internal filtering methods like `GL_NEAREST`, `GL_LINEAR` and one using autogenerated MipMaps. For all these textures use the exact same texture data imported previously. These textures may be switched during rendering using the keys `1`, `2` and `3`.
Create a vertex and a fragment shader handling texture coordinates and a texture uniform for shading the rendered object. Use very simple lighting in your shader for now (omnidirectional light at the camera's position).

Complete the following steps ...

- Load texture data from a file. (here: "trashbin.png")
- Initialize an OpenGL texture object.
- Set up texture parameters for filtering and clamping.
- Define a texture uniform in your shader and get it's location in your C++ code.
- Bind and pass the texture to the shader before rendering an object.

...and you should see a picture like this:

Figure 1: Fully textured trashbin using texture coordinates.