M.Stamminger, C.Weber *Erlangen,* ***Monday, 24.11.2014***

Computer Graphics - Programming Exercises

Assignment 6 [6 Points] (Texturing)

In this assignment, the scene will be augmented with textures. You will find various 2D images in the `data` folder which you can use as textures for earth, moon, saturn and the background (however, you are free to use your own images if you have nicer ones). For the background, there is a function `drawCube` in the skeleton, which draws the six faces of a cube in a way that the inner part is front-facing. Such a cube can be drawn around the entire scene and textured with a seamless background image showing some distant stars to create the illusion of being in outer space.

a) The first thing to do is to get the jpg-images into the OpenGL texture format. Qt offers a very convenient way to load images from a file via the class `QImage`. Along with that, the function `QGLWidget::convertToGLFormat` might be useful for you, have a look at the respective references to get more information. You will find a function `initTexture` in the program skeleton to assist you with the generation and initialization of an OpenGL texture.

b) In order to be able to glue the textures on the geometry, OpenGl needs to know a texture coordinate for each vertex. Augment the two geometry-generating functions `drawSphere` and `drawCube` with the assignment of texture coordinates using `glTexCoord2d`. While this is quite straight-forward for the cube faces, the texture coordinates for the sphere need a more elaborate mapping which you should know from the lecture.

c) Draw the textured geometry by binding the according texture prior to each draw call. Once a texture is bound to OpenGL and the vertices have texture coordinates, you can easily perform the mapping in a shader. In the vertex shader, you can access the texture coordinate of the vertex with the built-in variable `gl_MultiTexCoord0`. Use the provided `varying` to interpolate the texture coordinate; In order to get access to the 2D texture in the fragment shader, there is a variable `uniform sampler2D tex` which is provided the appropriate value (the texture currently bound to texture unit 0). You can get a specific value of this texture using the built-in function `texture2D`.
Implement the shader in the files `Light_and_Tex_VS.glsl` and `Light_and_Tex_FS.glsl` respectively. As the name implies, not only the texturing but also the lighting computations should be performed. The color value of a fragment should no longer come from the actual color attribute but from the texture. However, there should be three different output styles:

- Earth, moon and saturn should be applied both, texturing and lighting. But take care to apply the specular term only to reflective surfaces, e.g. water.
- The background shouldn't be affected by lighting, just the raw texture color should be asigned.
- The rings of saturn shouldn't be textured, just lighting should be applied.

Do not write a shader for each of the styles, you can put it all in one shader program. Just encode the type of the geometry in a color attribute since it is not required for the planets and the background (the colors come from the textures). However, since we use only the sun as light source, you may want to use the ambient color for each planet as well.

Additional Information

- When you draw the cube in a wide range around the scene, don't forget to extend the far plane accordingly so that the background is not clipped.

Implementation Guidelines

Note that the `bindUniforms` functions was extended by one argument (the texture ID).

Good Luck!

Your source code will be copied from your handin directory on:
Monday, 01.12.2014 14:00 pm
all subsequent changes cannot be taken into account!