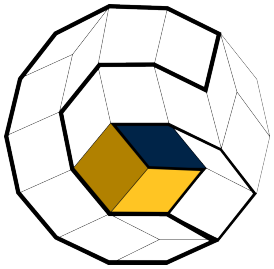
TU BRAUNSCHWEIG
Dipl. Inform. Martin Eisemann
Institut für Computergraphik
Lorenz Rogge (rogge@cg.tu-bs.de)

May 24, 2011



Echtzeit Computergraphik SS 2011
Assignment 6

In this exercise we improve the rendering routine of our MeshObj class. We replace the existing glBegin() - glEnd() routine with efficient VertexBufferObjects (VBO) for rendering. In the second task we use multi texturing to render a earth with clouds and emissive lighting on the night side. Please present your solution to this exercise on Monday, May 30th, 2011.

6.1 Speed up rendering with VertexBufferObjects (20 Points)

In the last assignment, you further improved the mesh import and the data arrangement in a indexable vertex array. We now want to use this preformatted data to speed up our rendering process.
Adapt the `setData()` and the `render()` routines of `MeshObj` to the *new* header file. It now defines a Vertex Buffer Object (`mVBO`) and the indices used for rendering as an Index Array (`mIBO`) .
To init yout buffer objects, you need to get a new location from the OpenGL state machine by calling `glGenBuffers`. Bind your generated buffer objects as the appropriate type (`GL_ARRAY_BUFFER` or `GL_ELEMENT_ARRAY_BUFFER`) and copy your vertex and index data into the buffer objects using `glBufferData()`.
When rendering your object, you need to specify the vertex attribute arrays after binding the vertex buffer. Use `glEnableVertexAttribArray(index)` and `glVertexAttribPointer(index,...)` to specify which data goes into which attribute array. Notice, that the indices are already predefined by OpenGL:

- 0 - vertex position
- 2 - normal data
- 8 - texture coordinates

These are the internal addresses for the attributes accessible in your vertex shader by `gl_Vertex`, `gl_Normal`, and `gl_MultiTexCoord0`.Be sure not to use the index 0, which is not defined by OpenGL or any other index, since you might change some other attributes and creating *interesting* effects you were not looking for.
Now that your are done specifying the vertex attributes, bind your index buffer object `mIBO` and render the indexed data with the command `glDrawElements` as triangles.

6.2 Multi-Texturing (20 Points)

Extend your program to handle not only one but multiple textures simultaneously. For this, the predefined structure `Texture` has to be initialized and filled with the needed data for every texture layer used in this assignment.
We want to display the planet earth with a simple color layer describing the actual earth's surface. An emissive texture is used to display city lights on the dark side not facing the sunlight. The view down to the surface is blocked by a cloud layer, which is given by a cloud texture and a mask texture describing, how dense the clouds are a different locations in the sky.
Alltogether, we need *four* textures - one for each information layer.
Complete the missing parts for initializing and loading these textures and pass them to your shader using multiple texture units. Activate them using `glActiveTexture()` and pass the different texture layers to the corresponding uniforms in your fragment shader.
This shader should render the earth with lambertian lighting, such that the brightness of the curved surface reduces with increasing viewing angle towards the surface normal (note, that this effect accounts for the lightsource also!). Remember that dense clouds block the view down to the surface (only clouds are visible) as well as the emitted light from the backside of the planet, while thin cloud layers block the light only partly. Where no clouds are visible, the light is not blocked at all and the earth's surface is clearly visible.
Use your shader the create this result and apply the emissive texture only on the sides of the globe, where no bright daylight is hitting the surface (the street lights are only turned on at night, right?!). Finally, ambient lighting is illuminating the whole globe a also little. Use a light color of $(0.2, 0.2, 0.2)$ for ambient illumination.
Place your *sun* fixed at $(1, 1, 1)$ in *world space* (which is quite close for a sun, but we're good for now) emitting plain white light.
As the result, you should see renderings like the following:

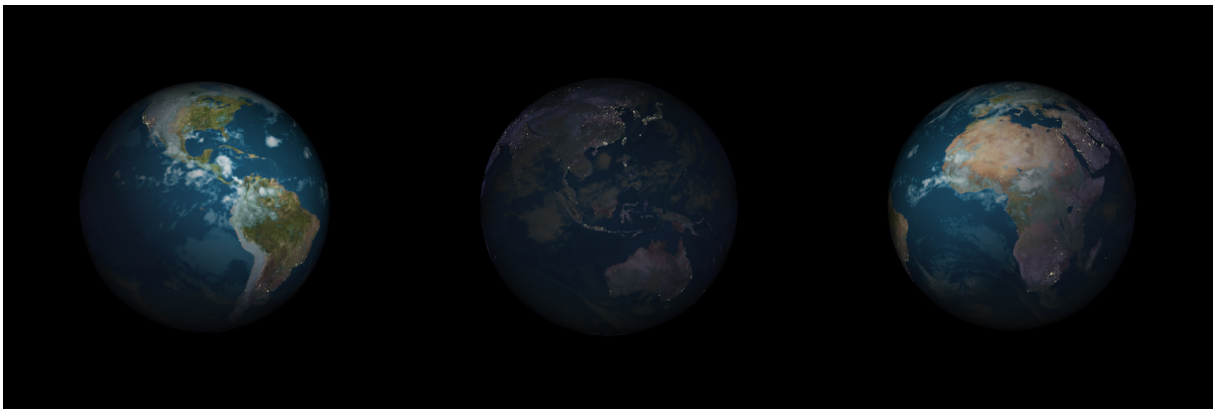


Figure 1: Three different views of the earth. Dayside (left), at night (middle), dawn(right).