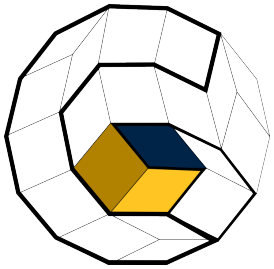
TU BRAUNSCHWEIG
Dipl. Inform. Martin Eisemann
Institut für Computergraphik
Lorenz Rogge (rogge@cg.tu-bs.de)

May 10, 2011



# Echtzeit Computergraphik SS 2011
## Assignment 4

In this assignment we learn the basic use of GLSL in shader programming. Please present your solution to this exercise on Monday, May 16th, 2011.

### 4.1 Normal reconstruction (10 Points)

The ObjLoader implemented so far only supports vertex data which is being index for rendering. To make object appear more smooth and to be able to light the scene correctly, normals attached to each vertex are very important. In this task implement a method for reconstructing normals from the loaded geometry data (only vertex data). To do so, you need to compute each faces' normal from the given data and calculate a vertex' normal from all surrounding faces and their normals. Having this, your are able to reconstruct normal data from geometry provided without any of this. To compute the correct normal direction, we assume the faces to be defined *counter-clockwise*. Complete the method `reconstructNormals` defined in the class `ObjLoader` to compute normals from loaded vertex data. Use the method to directly compute missing normals after loading an .obj file and store the normals in the predefined `Vertex` struct. As a last step, add the use of vertex normals to the render method of `MeshObj`.

### 4.2 Spotlight shader using GLSL (30 Points)

From the lecture you now know the basics of *GLSL*. We now want to use this technique to improve our rendering. For a simple start, we do not use materials or colors in this task, but concentrate on how to setup and use shaders in general. As an example we want to create a spotlight coming from the camera origin - just as a flashlight lighting the scene. This spotlight is defined by a direction, a color and two angles defining the size of the light cone sent out by our light source. The two angles define three regions. All pixels within a cone of angle `innerSpotAngle` are fully lit by the spotlight. Every pixel inbetween `innerSpotAngle` and `outerSpotAngle` should be lit with a linear fall-off from one to zero coming from the center, so that the edge of the spot appears smooth. All pixels with an angle towards the light source direction greater than `outerSpotAngle` are only lit by ambient light, which is constant for the whole scene $La = (0.2, 0.2, 0.2)$. The spotlight can be controlled using the keys R/F and T/G manipulating the inner and outer cone angles (this is already implemented).

In this task, complete the method `initShader()` by loading shader source files for a vertex and a fragment shader. Attach both shaders to a shader program `shaderProgram` and provide access to two uniform variables called `uniform_innerSpotAngle` and `uniform_outerSpotAngle` in your C++ code. These can be updated in every renderpass before rendering the scene, allowing to interactively control the lights parameters.

Finally create the needed shader source files for our vertex and fragment shaders.

In the vertex shader you need to compute the vector connecting a vertex and our light source. This vector is used to determine the angle towards the lights direction in the fragment shader. Also compute the view vector towards every vertex, just as explained in the lecture. This vector is necessary to compute correct lambertian lighting when looking at curved surfaces. Remember that the brightness of

a surface is related to the viewing angle between surface and camera. Therefore we also need the surface normal at each position. So pass the vertex-camera-direction, the vertex-light-direction and the vertex normal to the fragment shader.

In the fragment shader you need the interpolated values computed in the vertex shader and the two uniforms defining the two spotlight cutoff angles. Compute the angle between light direction and the directional vector between pixel and light source. Depending on that angle, compute the lighting of each pixel. If the pixel's angle is smaller than the `innerSpotAngle` apply full lighting, between `innerSpotAngle` and `outerSpotAngle` linearly cut off the lighting to zero. Let the spotlight always point in the viewing direction and add an ambient illumination term to the scene.

Save the final color for each fragment to `gl_FragColor` and your are done.

The result should look like this:

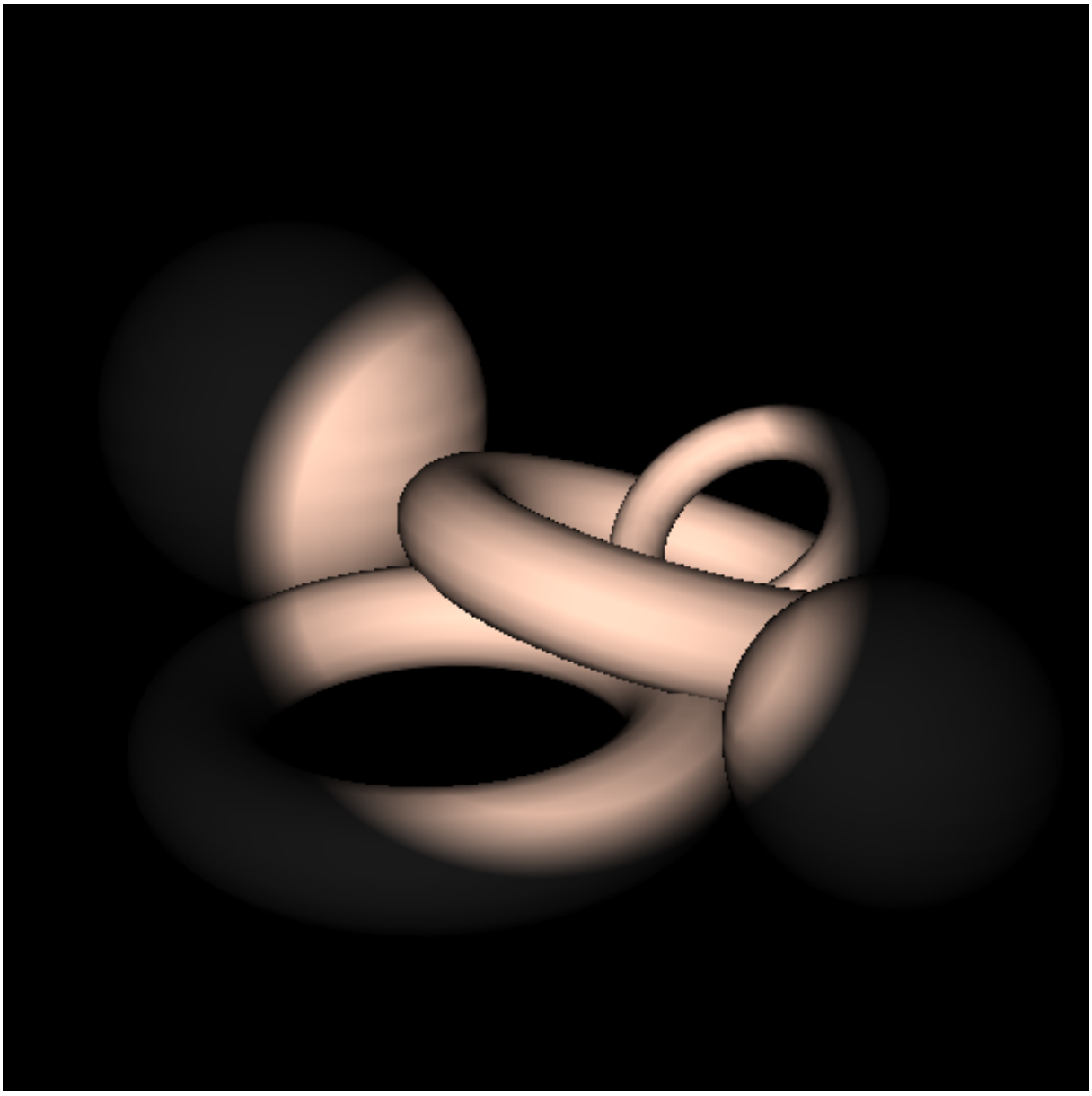


Figure 1: Exemplar view of the scene lit by a spot light.