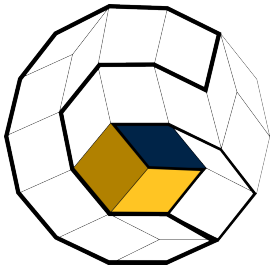
TU BRAUNSCHWEIG
Dr.-Ing. Martin Eisemann
Institut für Computergraphik
Lorenz Rogge (rogge@cg.tu-bs.de)



May 21, 2012

Echtzeit Computergraphik SS 2012
Assignment 6

Present your solution to this exercise on Monday, June 4th, 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 per-fragment-lighting shader of the last exercise to handle multiple light sources simultanously.

6.1 Multiple Lights (40 Points)

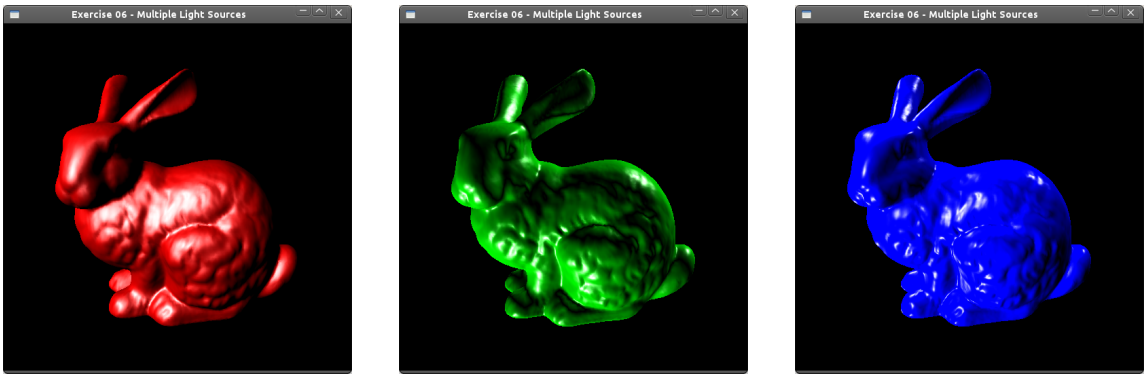
To render multiple light sources at once requires to upload every used light source's parameters to the shaer program. This can become quite confusing very quickly, since for every light source you need four uniform locations to upload the data to. Therefore the provided code (c++) now also contains a struct `UniformLocation_Light` containing these four uniform location adresses as `GLint`. A map of the these structs allows to quickly access a specific light source and then access its parameters within the light source struct. The map `uniformLocations_Lights` is already available in the code. Using `uniformLocations_Lights["lightID"].ambient_color` allows to access the uniform location adress of the ambient color component of a light source identified by the string `"lightID"`, for example.
Also the given code allows to toggle 10 different light sources, if set up, by using the number keys `0` through `9`. This will enable or disable a light source, by setting a boolean within the light sources struct (have a look at `toggleLightSource(unsigned int i)`).

In the last exercise you combined your light source data and the material information only within the vertex shader and then passed the final values to the fragment shader. In a case, where you have less vertices than fragment pixels, this would a more performant solution. With multiple light sources, however, you would need to pass a lot of data from vertex to fragment shaders. The GPU has only limited bandwidth to do this. For example the *GeForce GTX 295* in my office is only able to pass 60 float values from shader to shader. A single light source, defined by position, ambient, diffuse, and specular color component would already require 12 floating point values. Then one would be able to handle five light sources at most. But then there would be no bandwidth for vertex position, vertex normal and other important data. The specs of the *GeForce GTX 295*, however, state, that the fragment shader is able to handle 2048 floats a uniform inputs. This would allow to use more than 150 light sources without any problems. (You may want to check your GPU's limits by executing `glxinfo -l`)

So, using multiple light sources, it is a better idea to do most computations within the fragment shader program. You still might want to compute the vectors beween a vertex and every light source in the vertex shader and pass these values to the fragment shader using `in` / `out` variables.
Please extend your fragment shader to process multiple light sources simultaneously. Instead of having a single uniform `LightSource` you could now use an array of these structs. When shading a pixel, iterate through your active light sources and sum up the final color values for each light source to get your final fragment color.

You also need to extend your c++ program to initialize up to ten light sources (use different positions and colors) and to pass the enabled light sources to the shader program. Make sure not to upload any deactivated light sources. You also may want to use a uniform variable in your shaders, controlling how many light sources are active. That way, you will not need to cycle through every possible light source but only the active ones, to shade your pixels.

When you finished completing the code you should be able to toggle your different light sources and render views like the ones in figure 1.



(a) One light source (b) Two light sources (c) Six light sources

Figure 1: Exemplar renderings using different materials and light counts.