M. Stamminger, C. Weber *Erlangen,* ***Monday, 27.10.2014***

Computer Graphics - Programming Exercises

Assignment 2 [6 Points] (Lighting)

In this assignment, you have to implement the Phong lighting model using shaders. As you should know from the lecture, lighting computations can be done per triangle (Flat Shading), per vertex (Gouraud Shading) or per pixel (Phong Shading). You will implement the last two versions.

The scene to be illuminated is the sphere from assignment 1, however, there is an implementation of the drawSphere-function in the current program skeleton in case you did not manage assignment 1 (if you want to use your code, don't forget that you have to supply a normal for each vertex). Beside this, the whole shader compiling stuff is also implemented as well as the definition of user-defined variables which are handed to the shaders. Hence, there is actually no need for you to edit the main program file, you only have to write the code for the shaders in the given .glsl files. It is recommended that you make yourself familiar with the code in the .cpp files since an understanding of this code will help in subsequent assignments.

There are two light sources in the scene, one directional light D and a point light P. The light D has a constant light direction which is available in the shaders as the variable `vec3 D_LightDir`. The point light P rotates around the sphere and is visualized as a yellow dot in the scene. Its current position is available in the shaders as variable `vec3 P_LightPos`. The position of the camera can be accessed in the shader by the variable `vec3 cameraPos`. The remaining information you need for the lighting computation, the vertex position and its color, are available via the built-in variables `vec4 gl_Vertex` and `vec4 gl_Color`.

a) Implement Phong lighting using Gourad shading in the file Gouraud.glsl . Since this shading model computes a color value for each vertex , you only need to write a vertex shader, the rest is done in the fixed function pipeline. You can assume that there is no ambient light in the scene and the light intensity is always (1,1,1,1). The formula for a reflected color value L_r known from the lecture then simplifies to :

$$L_r = \sum_{i=1}^{\#lights} I_{diff} + I_{spec}$$

where $I_{diff} = k_{diff}(\mathbf{n} \cdot \mathbf{l_i})$ and $I_{spec} = k_{spec}(\mathbf{r_i} \cdot \mathbf{v})^s$. For the material parameter k_{diff}, take the color of the vertex. The specular parameter k_{spec} and the shininess parameter s can be chosen as you like. However, implement the lighting such that the light D only produces a diffuse lighting effect and the point light P produces a diffuse AND a specular effect! See the solution program to see how the final result should approximately look like.
Perform the computation of the required vectors in **world coordinates**, i.e. don't transform the input vectors with the modelview matrix prior to the computations. The final color value should be assigned to the built-in variable `gl_FrontColor` and don't forget to transform each vertex with the modelview and projection matrix before you eventually assign it to the output variable `gl_Position`.

b) Now implement the same with Phong shading. Since this model computes a color value per pixel, you have to implement a vertex AND a fragment Shader in the files Phong_VS.glsl and Phong_FS.glsl respectively. In order to provide the necessary (interpolated) values to the fragment shader, you have to use varying variables. These are special variables which have to be defined in both shaders. The value you assign to such a varying variable in the vertex shader is interpolated in the rasterization step and then available in the fragment shader. The final color value should be written to the built-in output variable `gl_FragColor`.

Additional Information

- The GL shading language (GLSL) is much like C, however, the specifications of built-in variables and data types change from version to version. We use version 1.2 of GLSL in the assignments. Also note that we use the GLEW library (GL extension wrangler) to get the necessary extensions required for using shaders.
- GLSL provides a lot of functions. Here is a list of functions you might find useful in this assignment, look them up in online references to get detailed information: **normalize,reflect, max, dot, pow**.
- You might wonder why positions in 3D space (such as the built-in vertex shader attribute `gl_Vertex`) are stored in 4D vectors. The 4th component is the homogeneous coordinate w, which you will learn later in the lecture. For now, you can ignore this component. In order to access only the first 3 components of a 4D vector `vec4 a` you can use swizzling: `vec3 b = a.xyz`
- If there are syntax errors in your shader code, you won't get any errors during the compilation of the main program with `make`. This is because the shaders are not compiled until you run the main program, compiler errors will then be written to the console. This also means that you do not have to re-compile the main program after you made changes in you shader code.
- The callback function for keyboard events was extended by the functionality to switch between the shading modes. If you press F1, no lighting is used (default). F2 activates the use of your Gouraud shader and F3 of your Phong shader. You can also increment/decrement the angular speed of the point light source with +/-.

Good Luck!

Your source code will be copied from your handin directory on:
Monday, 03.11.2014 14:00 pm
all subsequent changes cannot be taken into account!