Elmar Eisemann

Appearance Control and Shading

Practical Exercise

Introduction

Get used to the code

In modern graphics pipelines, it is possible to use shaders - small programs that modify the appearance of an object. Today, we will work in a simulated framework, which will allow us to write pseudo shaders. The function *computeLighting* will be used to produce a vertex shader. The result of this fucntion will define the appearance of each vertex of the model. The model will, hence, be colored based on the result of this function. Precisely, this function will be called automatically for each vertex and the resulting color will be mapped on the model. The function receives a position, a normale, and a light number (there will be several light sources, but please only compute the result for the light whose index is provided - the accumulation will be done automatically as well), and, finally, you receive an index of the vertex, which will allow you to identify the current vertex if needed (this will only be of importance when working with local materials later).

Compile the program ("g++ *.cpp -l GL -l GLU -l glut -I . "). Try the different modes with 1-4. For the moment, this will only affect the color of your model. Try outputting the normal as a color. Do you understand the output ?

Let there be light !

In this exercise, we will derive a tool for artists to light a surface and define the appearance of the model.

Reminder : Illumination models

For a start, look at Vec3D.h and get used to the functions (dot product, cross product, additions, normalisations etc.). Notice that some functions are STATIC, meaning that you need to write something like "float t = Vec3Df : :dotProduct(v1,v2) ;"

To place the light at the camera press l. The light position is stored in *LightPos[0]*, the camera position in *CamPos*.

Lambertian Model

First, we will define a simple diffuse surface. The main part of the formula is $dot(N, L)$,, where L is the direction of the point to the light and N its normal.

Also compare : `http://en.wikipedia.org/wiki/Lambertian_reflectance`

Implement a Lambertian Model (suppose the surface has a reflectance of 1**).**

Blinn-Phong Model

Implement this model, which also takes view-dependent effects (specularities) into account. Its main formula looks like this : $dot(H, N)^s$, where H is the vector exactly between the view direction to the point on the surface and the direction towards the light. s is the specular exponent and defines the shape of the highlight.

More can be found here : `http://en.wikipedia.org/wiki/Blinn-Phong_shading_model`

What impact does s have ?

For an even exponent, what do you realize when the light is behind the model ?

How can you fix this problem ?

Combine Lambertian and Blinn-Phong.

Toon shading

Toon shading does give the impression of looking at a comic drawing. In our case, all you need to do is quantize the illumination. The image below illustrates a possible result.

Implement toon shading. Deal with specularities independent of the rest (quantize them separately and add the white areas in the end).

FIGURE 1 – Toon shading

Also try out the what happens if you use the camera position as an additional light. This light is special and will not illuminate the scene, but transform the result of *computeLighting* to zero, if a vertex' theoretical illumination under a lambertian model is weak (lower than a threshold t).

What do you see ? What effect does t have ?

Change the global variable *BackgroundColor* to better see the result.

And there was light !

Now, we will focus on the control of the illumination. For an artist, it can be very difficult to control the light positions (just look at the credits of any modern movie, you have hundreds of people working on the light placements !).

Our goal is to simplify this work and provide the artist with a tool to efficiently place lights in an indirect manner.

We will use the function *userInteraction* in the following. This function receives, where the user has *clicked* (as the mouse button was used for the navigation, pressing *space* will launch this function instead of a click). The position, normal and the index of the clicked vertex (the last will also be stored in *selectedIndex*) will be provided.

Our goal is to use this function *userInteraction* to move the light (to this extent, we will change the global variable *LightPos[0]*).

Placing the light with the mouse

Place the light on a sphere of radius 1.5, centered at (0,0,0). The light position on the sphere should project to the screen position that was clicked by the user. Pay attention to choosing the intersection point closest to the observer - which can even be behind the observer !

Placing the light according to shading

This time, the new light position should be chosen such that the light produces a exactly a lambertian shading of zero at the clicked location. (Use the mode Toon shading to verify your solution). There are several possibilities to achieve this goal ! The implicit condition is that $dot(L, N)$ should be zero, which leaves some degrees of freedom. **Come up with a solution that you find appropriate and test if the light behaves "intuitively".**

Placing the light based on specularities

Placing a specularity is not easy because its position depends on the view AND the light. **Find a solution to make sure that the specularity is centered at the clicked location.**

Multiple light sources

Use the combination *shift+L* to add lights in the scene. To start, reduce the power of the light by multiplying the result in *computeLighting* with a small constant, e.g., 0.4. To restart from scratch with all your light sources, press *shift+N*.

Lighting like a pro

Place light sources around your model to produce effects like : Rim light, Back light etc.

How can you simulate a large light source ?

Controlling the parameters

We would like to add colored sources, which will be possible by using the array *LightColor*. It will contain the RGB triplet for each light source, which represents its color.

Complete the function *keyboard* and add the keys for R, G, B, r,g,b. Capital letters should augment, small letters reduce the corresponding values in LightColor by a small constant, e.g., 0.1. The affected light should be the current, which is stored in a global variable *SelectedLight* and can be changed by the + and - keys. The selected light can be determined on the screen by its border color.

Change your *computeLight* function again to take the light color into account (the index of the light corresponds to the source to evaluate).

Change the functions that control the light placements to only modify the current light.

Try to reproduce the lighting shown in the screen shot below by making use of your light-placing functions.

FIGURE 2 – This scene uses a rimlight in blue, a pink light, a white light and a specularities (with an exponent of 2.0)

Try to work with negative light. Or decide on a per-light basis whether to use specularities or not.

Material editing

In this exercise, we will also allow the editing of the surface materials. Again, we will make use of the function *userInteraction*.

For each selected vertex, we will fill values in the array MeshMaterial. *computeLight* will then chose the shading type (specular, lambertian, toon, based on this user input).

Add the possibility to define materials. Use this solution to render the eyes and lips specular, while keeping the remaining surface diffuse (see image below).

Additional exercises

Do you have even better ideas to control the light ? What would be a good interface to choose light and materials ? Do you have ideas for other illumination models (not necessarily realistic) ?

FIGURE 3 – Eyes and lips are specular, the rest is not