Shading and Light Design

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. All the functions that you need to write are in yourCode.h. You do not need to touch any other file !

Precisely, the functions you will write are called for each vertex and the resulting color will be mapped on the model and interpolated using Gouraud shading.

Under Linux : Compile the program (”g++ *.cpp -l GL -l GLU -l glut -I . ”). Launch it with ”./a.out”. Under Windows : Use the setup from your last exercise. You can simply replace the source files.

To get started, try the different keys and play with the program. Try to understanding why the debug mode (key 0) shows a colorful head - the function direct.

When working on the exercises, be sure to also verify the description of each task in the corresponding code part of yourcode.h

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 : :dot-Product(v1,v2) ;”

To place the light at the current camera position press l. To add a second light, use L. For this exercise, we will always assume to have white light sources.

Lambertian Model

First, we will define a simple diffuse surface. The main part of the formula is $Kddot(N, L)$,, where L is the direction of the point to the light and N its normal. Use the vector Kd to lookup the coefficients. Do not forget that light from behind/below the surface (with respect to the normal) should never illuminate it.

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

Implement a Lambertian Model in the function diffuseOnly function.

Phong Model

Implement Phong model in function phongSpecularOnly, which introduces view-dependent effects (specularities). Its main formula looks like this : $Ksdot(R, V)^s$, where R is the reflection vector (the reflected vector from the light at the plane defined by the surface's normal) and V is the vector from the surface point towards the camera position. Make use of the std :vector Ks, to lookup the coefficients.

Blinn-Phong Model

Implement the Blinn-Phong model in function blinnPhongSpecularOnly, which also takes view-dependent effects (specularities) into account. Its main formula looks like this : $dot(H, N)^s$, where H is the unit vector exactly between the view direction V and the direction towards the light. s is the specular exponent (in the code called shininess) and influences the size of the highlight. Use the vector Ks to lookup the coefficients.

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

What impact does s have ?

If, for an even exponent, a light behind the model illuminates it, you might have to fix your code !

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 the toon-shading function *toonShadingNoSpecular* and look at the indications in the code. Then complete the function *toonShadingOnlySpecular*.

FIGURE 1 – Toon shading (left without, right with specularities

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 is used for the navigation, pressing *space* will launch this function instead of a click). This interaction will trigger the function *userInteraction...*. We will use these functions to define a new light position, based on different criteria. To switch the interaction mode (hence, the userInteraction function that is called), use the "M" key. Please also read the indications in the source code.

Placing the light with the mouse

The function userInteractionSphere will be used to place the light at the closest intersection of a ray through the clicked position and a sphere of radius 1.5, centered at (0,0,0). Pay attention to really choosing the closest intersection - which can potentially 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 exactly a Lambertian (diffuse) 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 (there are several options). Test if it is "intuitive" to place the light using your method and potentially think of alternatives.**

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.**

Optional

You can present the results of the optional part during your project presentation at the end of the course to gain a small bonus. You can get the full points without working on these questions. Do you have even better ideas to control the light ? Do you have ideas for other illumination models (not necessarily realistic) ?