'A Tutorial on Phong Lighting'

Colby Rush

April 24th, 2014

1 Overview

This tutorial resides at http://www.antongerdelan.net/opengl/phong.html. This tutorial outlines a technique used to imitate B.T. Phong's illumination modelling system. It starts with a quick description of how Phong's model works to calculate light intensity for a given point. Afterwards, an explanation of surface normals and how they affect the light reflecting off them is described, as well as describing how normals are calculated. It then describes how to combine both the fragment and vertex shader to interpolate values for the light intensity, with a few different types of light interaction (ambient, diffuse, specular) all being eventually added to fully light the shape according to the Phong model.

2 Key Points

The first key point lies in the description of Phong's model. It is composed of 3 reflection approximations that only look good when combined, but with the upside of being quick to render with current shaders. The main points of Phong's model are:

- Light is calculated for objects already known to be visible
- Light from a surface to the eye is modelled specularly
- Surface smoothness is aproxximated per-surface with specular reflectivity
- Surface roughness is approximated per-surface with diffuse reflectivity
- We don't care about object-to-object reflections

- Don't care about refraction or shadows ...

With all of these elements, total light intensity is relatively straightforward, calculating the value from surface(Diffuse light intensity + specular light intensity + ambient light intensity), which we use to color the surface appropriately.

The second key point is the description of how to calculate the normals for each shape's surface. Specifically, taking the cross product of two of it's edges. Flat surfaces are straightforward: each vertex has the same normal. For curved faces, we calculate each normal as an average of the surrounding face's normals. This will end up giving the illusion of a curved surface, which is good enough. See terms and equations for the diagram given.

The third key point involves description of the shader's used in calculating and processing the position and normals for each vertex. A summary of the code reveals the vertex shader simply interpolates the positions and normals, and outputs them to the fragment shaders, which control the actual coloring of that segment of the shape face. The fragment shader, once getting this information, uses it, along with the given light source and relevant intensities (specular, diffuse, ambient), to calculate the light intensity for each of the three attributes.

The fourth and final key point involves a detail on how each of these light attributes are calculated, simplified for clarity. The ambient light is simply a coloring of everything it touches, to the effect of a dark gray, since it's value is so low that it doesn't really affect anything, but is still there. The diffuse intensity is similar, but will increase/decrease in intensity as the face is facing towards/away the light, respectively. To calculate this, we find the line between the face and the light source, and use it to determine this intensity. Finally, for the specular intensity, we find the angle of both the viewer to the face being lit, and the angle of the face and the light source. If the angles are the same, intensity! If perpendicular, no intensity.

3 Terms and Equations

I will be putting each new term/equation on it's own paragraph from here on out.

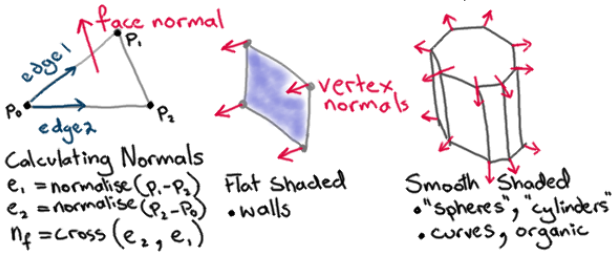
Global illumination is a grouping of algorithms that take into account light all over a scene, including from a light source (direct illumination) and light rays that may reflect from other surfaces (indirect illumination). This is in comparison to Phong's model, which only takes one reflection into account, not surface-to-surface. (See key points)

I didn't know the distinction between ambient, specular, and diffuse light

until reading this. Ambient light is the general light that is entering a scene, while diffuse light is the light captured by surface, and specular light is the light that directly reflects off of a surface.

Surface normals are the actual direction a surface is facing, while vertex normals are the directions a vertex is facing, determined by the surface normal it is associated with. Surface normals of a curved surface are determined by taking many small segment samples of the curved surface, and averaging them out.

Figure 1: How to calculate normals (from tutorial)



One of the things that I found interesting to learn about, while not pertient to the actual tutorial's methods, was a quick history of the older methods for lighting. Many of the papers are easily linked at the page given, so I won't bother with putting them into the bibliography, instead only focusing on the tutorial's description of them. But learning about the differences between per-vertex, per-facet, and per-fragment lighting calculations (interpolated, not interpolated, smaller segments made per-vertex, respectively) was neat.

indent

Shaders of both vertex and fragment type are functions that handle the calculations needed to make for light intensity per fragment for each shape. See key points.

4 Conclusion

This tutorial was a nice crash course into handling the more complicated lighting element present in openGl, not getting anything complicated involved. It's a simple triangle draw, and the author walks you through adding each different type of light until the finished product, while giving a nice amount of background to the theory. The code is concise, and the simple diagrams and math behind it make it easy to see how it works, both mathmatically and physically. It even threw in a little bit about surface

properties, which helped me in my other assignment in this class a little bit.

But the reason I picked this tutorial is the method he describes and explains calculating surface normals. It was a very straightforward explanation, but the hand-drawn diagram really helped it click for some reason. The way he combined it into his 'shader' functions made it seem very straightforward to put it into my own code, and let me understand what was actually happening instead of just grabbing someone else's code and hope it worked with mine. Finally, the three-step process of adding ambient, diffuse, then specular made it very easy to see the steps instead of just one function that did all three at once.

In conclusion, this paper was a very nice hand-hold through some of the more intimidating aspects of creating a realistic scene using the Phong technique for modelling light. While the math behind it was a little intimidating to think about, the author made it really simple to understand and apply to more than just his given example, as well as giving a basic lecture-level quick rundown of basic lighting model evolution and techniques.