Michael McQuade

Zhigang Deng

Computer Graphics

8 April 2021

Shading

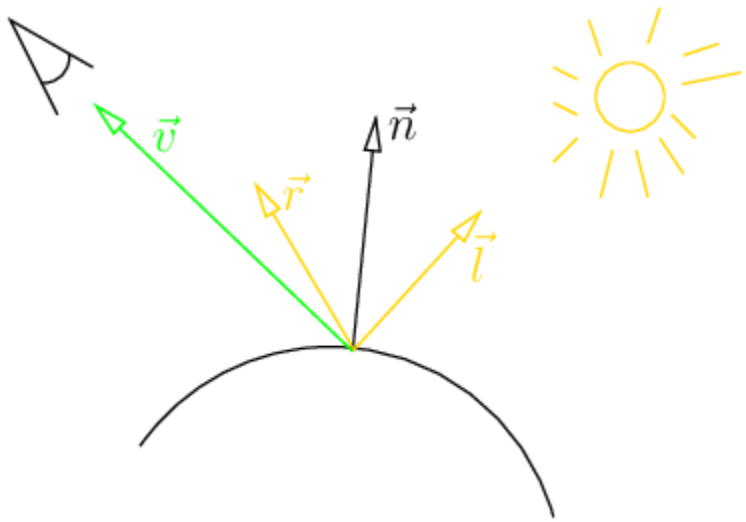
This report will cover the shading of the teapot using the Gouraud and Phong shading algorithms. In this assignment, several changes were added. First, there is now the idea of a material. In this case, a material is a vector containing values for ambient, diffuse, specular, and specular power. These values will be used as coefficients to scale the various lighting.

Ambient light can be thought of as a constant light that is applied evenly throughout the entire scene, so the coefficient passed in for the ambient portion of the material can be straight multiplied with the color values to get a new color value that has an ambient light value applied. In our case, that will only make the teapot darker, because we are starting with an all white teapot and we have a decimal value for ambient light, so it will always decrease the light evenly across the entire surface of the teapot.

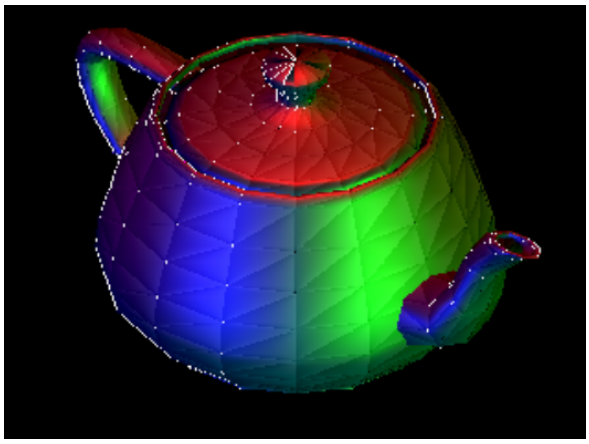
An example of ambient lighting evenly shading the teapot

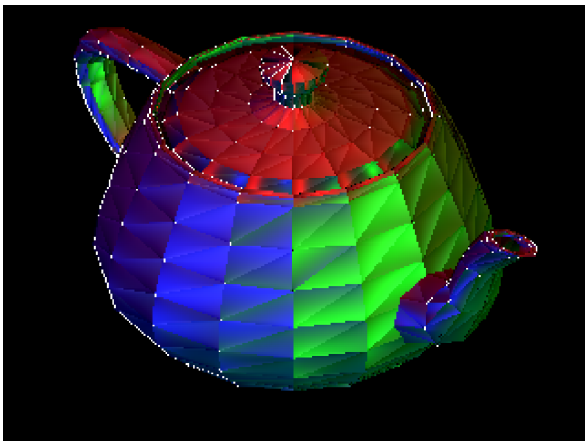
After getting ambient lighting worked out, it's quite a bit more work to get diffuse and specular lighting applied. In order to apply this lighting correctly, we need to apply a shader at each pixel level. That's because every pixel or fragment should reflect light differently than other pixels because of the direction of the viewer and the direction of the light source. In order to achieve this, shading needs to be done at the pixel level so it can take into consideration the overall worldspace coordinates or the barycentric coordinates, giving it a relative position within the current face. In OpenGL, this is done by using a *varying* keyword, which is a keyword for variables that change based on the relative location of the fragment inside the triangle, rather than being a constant value inside the shader.

In our case, our shader will receive the input of the color that it should start with, and the normal value. With this information we can calculate the color that the shader should output (or the fragment should reflect) by doing some calculations. These calculations involve: getting a normalized vector representing the position of the light source relative to the view, L. The normalized eye position, E. Finally, the normalized R, which is the reflected light ray. Below you can see a graphic representing these:

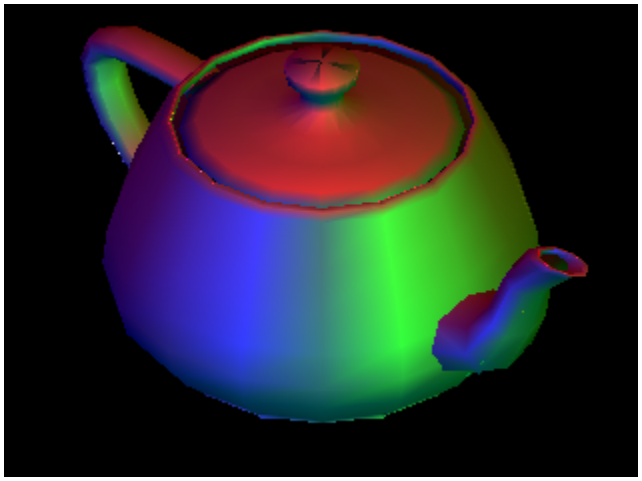


Some of the difficulties during this assignment involved starting too high level and trying to get this code to work without getting down to the per-pixel or per-fragment level of work. By trying to manipulate the colors in the actual drawTriangle method instead of in a separate shader method that was called lower in the pipeline, the triangles had interesting coloring results, but were undesirable.

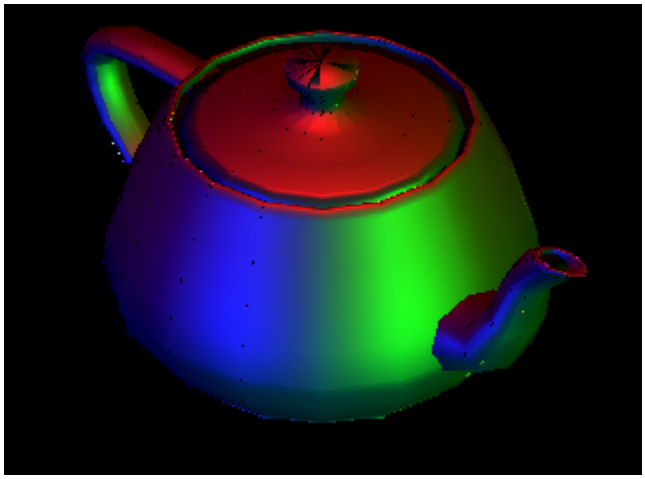




In the end, the desired result was achieved by extracting this logic into a shading function, interpolating the color values for gouraud and calling the previous assignments drawTriangle method with those color values. For phong, a new drawTriangle method was created which used a modified drawRasLine function that handled interpolating the normals across the triangles.



Final result of Gouraud



Final Result of Phong

Works Cited

Ruange. “Gouraud Shading and Phong Shading on Phong Reflection Model.” *GitHub*, github.com/ruange/Gouraud-Shading-and-Phong-Shading.

“Shaders.” *LearnOpenGL*, learnopengl.com/Getting-started/Shaders.

Ssloy. “Shaders for the Software Renderer.” *Tinyrenderer*, github.com/ssloy/tinyrenderer/wiki/Lesson-6:-Shaders-for-the-software-renderer.