Lab 7 - Smooth vs Flat Shading

You may work in pairs on this assignment. To receive credit, push your code to your git repository by 11:59PM on March 27th. Be sure to send me an email with your partner's name and which repository the code is pushed to.

Flat and smooth shading are used to describe how the normals of a mesh are used to shade a model. When flat shading, each face of a mesh uses its true normal in lighting calculations which results in a very faceted look. When smooth shading, the normal at each vertex is an average normal of all the faces that are constructed using that vertex. When using the averaged normal in lighting calculations, the mesh appears curved, or smoothed.

In this lab, you'll be loading mesh data from an obj file and shading it using both flat and smooth shading. An obj file represents an indexed face set, which is a list of vertices followed by a list of faces. Each face is defined by three indices that reference the previously defined vertices to define a triangle. You will need to iterate over this data, compute normals and organize the data into OpenGL buffers so you can switch between a smooth shaded render and a flat shaded render by pressing the spacebar.

Part 1 - Normals For Flat Shading

Just like last lab, we'll need normals in order to calculate our shading equations. This time we can't just hard-code a list of normals like we did for the cube. Instead, we need to calculate the normals based on the three points that make up each triangle. In the provided code, we use an obj loader library to read in the mesh data, positions and indices, that represent a list of triangles. The initializeFlatModel method initializes your model for flat shading, but leaves out the normal calculation. It's your job to calculate the normal of each triangle. To do so, pick a single point of any given triangle and calculate the two vectors from that point to the other two points of the triangle. Take the cross product of those two points and normalize the result to get the normal. Once your positions and normals array are populated, the mesh is rendered using glDrawArrays with a draw mode of GL_TRIANGLES (see renderModelFlat).

Part 2 - Normals For Smooth Shading

To implement smooth shading on your model, you'll need to rewrite the initializeSmoothModel method. The provided code currently implements flat shading (minus the normal calculation), and it does so using glDrawElements rather than glDrawArrays. glDrawElements uses an index buffer to specify the construction of primitives. The major benefit of glDrawElements is your mesh data can be represented more compactly by not duplicating vertices. Instead, a vertex can be defined a single time and its index can be referenced multiple times in an index buffer when specifying which points define triangles. When using flat shading, we unfortunately cannot avoid duplicating vertex data because we need to specify a different

normal for each triangle that a point is used in, so there is no benefit to using glDrawElements. With smooth shading, though, any given point only needs a single combined normal of all triangles that are using it, so glDrawElements works great. Follow the steps below to specify normals for a smooth shaded model.

a. Specify each vertex position only once

Change initializeSmoothModel to only append each vertex position once. You will essentially be copying the vertex position data from the data structure that the obj loader uses into your own positions array.

b. Initialize normals to (0,0,0)

For every vertex position, you'll also need a corresponding normal. Initialize your normals array with the same number of normals, but set them all to (0,0,0).

c. Combine normals

Loop over every triangle in the mesh and compute its normal. Add the triangle's normal to each of it's vertex normals.

d. Copy indices

Copy over the indices from the obj loader's data structure, into your own indices array.

d. Normalize normals

After iterating over all your triangles to calculate their normals, loop over the vertex normals and normalize them. Now your normals are all smoothed.

Part 3 - Extra Credit

If you didn't get the extra credit from Lab 6, you can still get extra credit for it now by implementing it in Lab 7 (you can't get the extra credit twice!).

a. Specular (5 pts extra credit)

To calculate specular reflections we'll need to calculate the view direction (a unit vector that points toward the camera) and a reflection direction (a unit vector of our light direction reflected about the normal). We'll also need a shininess variable, which is a floating point or integer value. The higher the shininess value the shinier the surface (0 would be not shiny and all, 128 would be very shiny). Take the dot product of the view direction and reflection direction vectors, clamp the value to between 0 and 1 and raise it to the power of your shininess variable.

b. Camera space light (5 pts extra credit)

Try transforming your light position so the uniform variable is actually specified in camera space. Invert your view matrix to get a camera space to world space transform and use it transform your light position. Now as you move the camera around your light will always be shining on what you're looking at.

Things to notice

The only difference between smooth and flat shading is how the normal data of your mesh is specified. This means that we don't need to make any changes to our shaders. By just changing the configurations of the data that we upload to our position and normal buffers we can see a dramatic difference in how the model renders.

Recommended Reading

Phong shading
Smooth Shading Video
glDrawElements