Mesh rendering

1 Vertex stream

It will be easiest to render the mesh in triangle soup mode and using an index buffer. To get the index buffer contents, concatenate the rows of the triangle table (each row contains 3 unsigned integers).

You'll likely need two buffers for the vertex data: one for coordinates, one for normals. To get the contents of these buffers, just place the data describing consecutive vertices into an array (use `GLfloat*` as the array type) and use that array as an argument to `Buffer` constructor to create a buffer and send the data to the GPU. Remember that locations and normals are 3D vectors, so the arrays should be of size three times the number of vertices. See the sample code for how this works.

You'll have to compute normals yourself as described below. The normals will be stored in another buffer. Both buffers need to be attached to a vertex array.

2 Vertex normals

Use the area-averaged normals for the vertices. Every *vertex* will have its normal. To compute the normals, first allocate a V-dimensional vector (call it S) of 3D vectors, where V is the number of vertices of the mesh. Initialize these vectors to zero. Then, go over all triangles of the mesh. For every triangle Δ with vertices a, b and c, compute its normal N_Δ using the cross product of vectors running along two edges (for any triangle use the same indexing of vertices to make sure that normals point in consistent direction). Then, add N_Δ to the entries of S corresponding to the triangle's vertices, i.e. $S[a]$, $S[b]$ and $S[c]$. After the loop terminates, S will contain the area-weighted vertex normals for every vertex. Make sure your implementation is not quadratic – some of the input meshes are large.

3 Orientation

You can assume that the triangles of the model are consistently oriented. This means that one of the following is true:

(i) All triangles appear clockwise when looked at from the outside of the object

(ii) All triangles appear counterclockwise when looked at from the outside of the object.

This means that the mesh is back face culling-friendly: if you specify vertices of every triangle in a consistent order (e.g. a-b-c if the respective row of the triangle table is 'a b c') then you will be close to gettin the correct image with back face culling turned on. If it does not work (you see the back shell of the model rather than the front shell), swap *two* vertices in each triple of vertices in the stream defining a triangle. For example, instead of using the order a-b-c for triangle Δabc, use order b-a-c. This will change the orientation of the triangle. Alternatively, you can cull front faces instead of back faces with `glCullFace(GL_FRONT)` (note that front and back are defined by the sign of the cross product used for culling purposes, so front are not always faces that are in front – it depends on how triangles are oriented).

Because the triangles are oriented consistently, either normals of all triangles are going to point to the outside or all normals are going to point to the inside. If you see that your light source is not working as expected, it may mean you are using normals that point to the inside: negate the normals to get outward pointing ones.

If your method is is consistent over all triangles and it works for one of the input models, it will work for all intput models.