Ray-tracer for triangle renderer - due 12/6.

No late pts until 12/10, but no LATE assignments after 12/12!

Your program will render a polygonized, triangle-based model read in from an input file like the second assignment. Test models provided from Assn 2, but your program should be general to work with other triangle-based models and will be tested to show that it can work on other models during grading. Your project is to interpret the data and render the model parsed as follows.

You will have a few choices to build up to the complete assignment.

BASIC RENDERER (50pts) The basic renderer is mandatory part to implement a flat shaded polygon render. The ray-tracer will follow from the lecture description, the Barycentric solution is optional but recommended. The illumination should follow the Phong lighting model discussed in class and should contain diffuse, specular highlighting, and ambient components.

The next choices below are optional but the count should aim to total 100 pts for full credit:

PHONG SMOOTHING (20pts) If you choose, the rendering tool will use two modes to draw the model. The starting/default mode is a flat shaded render and the second is smooth shaded, using Phong smoothing to render. To this end, you must compute vertex normals and average the normals (hint Barycentric coordinates makes this easy - but don't forget to re-normalize).

BUMP MAPPING (20pts) Choose to add a simple example of bump mapping. Note, adding a bump effect over the polygons does not need to use the UV layout of the OBJ file, but it can if you want. To accomplish this, at a minimum, you need to show a simple bump effect. You can be creative, or follow the following simple rule. For each polygon, add a small circular dimple in the middle of the triangle by modifying the normals near the center of each polygon accordingly.

TRIANGLE REMESH (20pts) For general model rendering, a single obj file is not always going to be composed of only triangles. If you choose this option, when your system parses a face with more than three sides, you will divide it into a set of triangles. If you choose this option, your program will be tested against OBJ files that have arbitrary side-count faces.

Implementation:

You should start from the OpenGL test code provided in previous assignments. To be clear, you will be use OpenGL only to turn pixels on using glVertex. This is the only OpenGL call your code should make to render.

My suggestion is that your algorithm follow the pseudo-code given in class for ray-tracing. You may "hard-code" information about the light source(s) as in position(s) and intensities. A default material can also be hard-coded.

A simple background color (maybe, black or white?) should be set when a ray doesn't hit any objects in the scene. To help you pick your camera angle (scene set-up), we provide test objects (Assn 2). Sizing like in Assn~2 will assure the new model sits in the window you have set up (i.e. rather than moving the camera).

For the keyboard interface, a simple switch for the render modes is controlled by the "x" button press for smooth shading and a "b" button for bump map added and removed. With each button press, you will also need to re-render the model.

HINTS: This program may run slowly, especially if unoptimized. To speed up the debug process, test a part of the image only and don't do the full blown image size until the very end. Of course, \*do\* test the full resolution before you turn it in and submit a snapshot of the image along with your code.

Credit will be broken down as above plus 10 points for neat code and documentation. A maximum of 120 pts is available by doing all the options as extra credit. Your program should run on any of the machines in lab.

Good Luck!