Project 3 Additional Information Sheet

Implementation Requirements

In this project, you have to set pixel RGB values via your own functions. You CAN NOT use glColorPointer/glVertexPointer etc. You may use a modified draw_pix primitive that includes a way to specify the color of a pixel or you may use your own pixel buffer as in the other demo, HelloGraphics.cpp. You are again required to use `glBegin(GL_POINTS)` only, meaning you cannot leverage openGL's line drawing functions because you need to implement linear interpolation yourself.

Rubric Overview

Along with the requirements outlined in project guideline, here's a rough list of things I will check during grading. Make sure the following are working and you should do well. (Point value associated with specific part is in [])

1. Phong Model [30]: Since you are not required to implement CVM (Camera viewing model) in this project, the scene could look abnormal due to the fact that you are using orthogonal projection to view the scene. You should include a UI method to turn on/off the phong model mode. Be aware that your UI will be used heavily in testing the Phong Model's implementation. More details on requirements below.

2. Gouraud Shading [20]: Your program should use Gouraud shading to compute intensity (color) of pixels whose intensity is not generated by Phong model. I recommend using your DDA algorithm from assignment 1, unless you found Bresenham simpler.

3. Painter's Algorithm [15]: Your program should use painter's algorithm (or z-buffering if you prefer) to remove surfaces not visible to the user.

4. Halftone [15]: Your program should have an option to turn on/off half-tone mode. When in half-tone mode, your program should fill the object surface using the half-tone algorithm described in class.

5. UI [15]: If your UI is functional without any issue (i.e. crashes when running some command, inconsistent with README.txt instructions), you should receive full credit on this part.

6. Manual [5]: Please name your manual "README.txt" . Keep your manual below 150 lines in length and 80 characters per line. It should explain how to use your program and briefly explain what you believe deserves extra credit. Keep the extra credit section below 25 lines total.

Detailed Requirements

Phong Model:

$$I_{\mathbf{p}} = k_a I_A + \frac{I_L}{||\mathbf{f} - \mathbf{p}|| + K}\left(k_d(\vec{\mathbf{l}} \cdot \vec{\mathbf{n}}) + k_s(\vec{\mathbf{r}} \cdot \vec{\mathbf{v}})^n\right)$$

As you are aware, this equation encapsulates the relevant parameters to the phong model implementation that you will be implementing. None of these should be hard coded. The majority of these parameters are actually values that you will determine defaults for yourself or be taking from file directly.

Remember to normalize (make vectors length 1) all 4 of the vectors in the above equation! Note all faces will be triangles!

Parameters you determine default values for: (and have the ability to edit through your UI!)

These are listed in order of appearance from left to right in the above equation.

- $k_a I_A$ You can actually combine these into one variable if you would like. The ambient color should also be settable in your UI.
- I_L The intensity of the light source.
- K The "distance" from the light source to the face. This should be large enough that the ||**f** - **p**|| term is comparatively small most of the time.
- $\mathbf{f}$ The location of the eye. After a large amount of deliberation, I decided that I will not require you to implement the camera viewing model. However, I still require that you provide the ability to set the location of the eye. Now it can be moved anywhere and should be the same for all 3 views.
- $\vec{\mathbf{l}}$ You will technically calculate the light vector by normalizing the vector pointing from **p** to the light source, but I mention it here because you need to have a default location for the light source and provide the ability to move it. Note that this is decoupled from the *K* parameter in the denominator.
- k_s The color of the light source.

Parameters you retrieve from file and then calculate:

- $\mathbf{p}$ This is the location of a point on a polyhedron's face. You will receive the vertices of each polyhedron, but will need to calculate a location to associate with a given pixel for determining the color via the Phong Model. Do this with linear interpolation; I recommend tracking this data as you do your Gouraud Shading for the diffuse color.
- k_d The diffuse color of a particular point on a polyhedron's face. Calculate this via Gouraud Shading.
- $\vec{\mathbf{n}}$ The normals for particular points on faces are the source of the most computation in this model. Calculate the normal for each face using the right hand rule on the vertices of the faces. You will only need the outward facing normal, I will not use open shapes in grading. Use the special cases of the phong model described in class if viewing the back of a face.
- $\vec{\mathbf{r}}$ The reflection vector. Once you have the normal vectors, this is relatively easy to calculate.
- $\vec{\mathbf{v}}$ The view vector. Simply normalize the vector that results from subtracting points **f** and **p.**
- n The specularity constant. I chose to specify specularity per face to introduce you to the idea that graphics engines often track data per vertex and per face. These are simply read from file and I will not require that your UI is able to change them.

Example File

(I provide comments after each line, the comments do not appear in real files)

1 # Number of polyhedra (integer)

4 # Number of vertices (integer)
0.0 0.0 0.0 # Vertex 1 (floating point)
0.1 0.0 0.0 # Vertex 2
0.0 0.2 0.0 # Vertex 3
0.0 0.0 0.3 # Vertex 4
255 255 255 # Color for Vertex 1 (White at maximum brightness) (integer)
255 0 0 # Color for Vertex 2 (Red, no green, no blue)
0 255 0 # Color for Vertex 3 (Green)
0 0 255 # Color for Vertex 4 (Blue)
4 # Number of Faces (faces are always going to be triangles) (integer)
1 3 2 # Face 1 defined by 3 vertices, this being the first, third and second (integer)
1 2 4 # Face 2
1 4 3 # Face 3
2 3 4 # Face 4
1 # Specularity for Face 1 (integer)
25 # Specularity for Face 2
3 # Specularity for Face 3
1500 # Specularity for Face 4

As you can see above, no normals are provided. You will calculate these yourself. The faces are defined with the right hand rule in mind.

For the first face, the normal points downwards. Calculate this by taking

(face vertex 2 - face vertex 1) x (face vertex 3 - face vertex 1)

For this calculation, remember face vertex 2 is not the second vertex defined in the file. As shown by the ordering “1 3 2”:
Face 1 vertex 1 = vertex 1
Face 1 vertex 2 = vertex 3
Face 1 vertex 3 = vertex 2

(0.1i - 0) x (0.2j - 0) = -0.02k

This vector is then “normalized” to become -k = (0, 0, -1) in terms of only x, y, and z.

Extra Credit

The following are published extra credit opportunities for this project.

1. Display non-plainer/intersecting objects properly [up to 3]
2. Multiple viewports still! [up to 2]
3. Animations! [up to 3]
4. If your UI is stunning! [up to 5]
5. Cabinet/Cavalier/Arbitrary Parallel Projections [up to 5]
6. Perspective Projections! [up to 8]
7. Camera Viewing Model implementation [up to 8]

If you've done anything else extra, please document it in README file so we can assess it and try to give you more points!

Remember that this is meant to be done after everything else in the project! Extra credit is not meant to take the place of any aspect of the project!

Helpful Notes

There are a lot of parts in this program, keep in mind that graphics processing is a pipeline, so you should establish which process to be done first, this would make programming easier.

It would be easier if you test each part separately, Ex. for Gouraud shading, you should generate some simple picture to see if it's working before you put it in your processing pipeline.