1. Problem Statement:

The purpose of this assignment is to give students experience with low-level polygon rendering operations. Specifically, your task is to create a program to display convex polygons using the Phong shading illumination model and a scan-line rendering algorithm. Since this is a large and complex task, we are breaking it into two parts as described below.

Homework #4

Your first task will be to read an input file containing information about the L light sources for your scene, material properties for the M objects in the scene, and finally the coordinates and surface normals for P convex polygons.

To keep things simple, each of these pieces of information is identified in the file with the keyword “Light”, “Object” or “Polygon” followed by a sequence of data values. Light descriptors contain the light number, color values between 0..255, and light direction values between 0..1. Object descriptors contain the object number, color values between 0..255, and Phong lighting coefficients. Polygon descriptors contain the object number, the number of points in the polygon, followed by 3D coordinates and surface normals. To keep things simple, you can assume that all of the polygon coordinates have been projected into screen coordinates with x between [0..499] and y between [0..499]. For example:

```
Light 0 R G B X Y Z
Light 1 R G B X Y Z
...
Object 0 R G B Ka Kd Ks Kp
Object 1 R G B Ka Kd Ks Kp
...
Polygon 0 4
Px Py Pz Nx Ny Nz
Px Py Pz Nx Ny Nz
Px Py Pz Nx Ny Nz
Px Py Pz Nx Ny Nz
Polygon 0 4
Px Py Pz Nx Ny Nz
Px Py Pz Nx Ny Nz
Px Py Pz Nx Ny Nz
Px Py Pz Nx Ny Nz
...
```
Once you have read the Light and Object information above into arrays in your program, you must use this information to calculate the (r,g,b) color value for each of the polygon endpoint using the Phong shading illumination model. For each (r,g,b) color channel, you will need to calculate the sum of ambient, diffuse and specular terms as follows:

\[
R = \text{object.Ka} \times \text{ambient.R} \times \text{object.R} \quad \leftarrow \text{use ambient light}
\]
\[
+ \text{object.Kd} \times \text{NdotL} \times \text{light.R} \times \text{object.R} \quad \leftarrow \text{sum over all lights}
\]
\[
+ \text{object.Ks} \times \text{pow(RdotV, object.Kp)} \times \text{light.R} \quad \leftarrow \text{sum over all lights}
\]

\[
G = \text{object.Ka} \times \text{ambient.G} \times \text{object.G} \quad \leftarrow \text{use ambient light}
\]
\[
+ \text{object.Kd} \times \text{NdotL} \times \text{light.G} \times \text{object.G} \quad \leftarrow \text{sum over all lights}
\]
\[
+ \text{object.Ks} \times \text{pow(RdotV, object.Kp)} \times \text{light.G} \quad \leftarrow \text{sum over all lights}
\]

\[
B = \text{object.Ka} \times \text{ambient.B} \times \text{object.B} \quad \leftarrow \text{use ambient light}
\]
\[
+ \text{object.Kd} \times \text{NdotL} \times \text{light.B} \times \text{object.B} \quad \leftarrow \text{sum over all lights}
\]
\[
+ \text{object.Ks} \times \text{pow(RdotV, object.Kp)} \times \text{light.B} \quad \leftarrow \text{sum over all lights}
\]

All of the information you need to perform these calculations will be in your Light, Object and Polygon arrays, but you will need to do some work to calculate NdotL and RdotV. Remember the R vector is calculated using the L and N vectors. You can also assume that the direction to the viewer V = (0,0,-1).

In the data file all surface normals and the light direction vectors have been normalized to unit length. This will simplify your NdotL and RdotV calculations. For this program, you can assume that the ambient light color is (255,255,255).

As you are adding together (r,g,b) color values from different sources, make sure your output values stay in the range [0..255] or you will see some very strange looking wrap around artifacts. One popular option is to do all of your (r,g,b) color calculations above using floats in the range [0..1] and then convert this color value to [0..255] just before display.

Once you have completed the Phong shading calculation for each polygon endpoint, your program should print these (r,g,b) values to the screen and also display the polygon endpoints in the correct color on the screen. To do this, store the colors in an image[500][500][3] array and use the glDrawPixels() command to display this 3D array on the screen (see render.cpp for implementation details).
Homework #5

Once you have successfully calculated the (r,g,b) colors for all of the polygon endpoints, your next task is to extend this program to fill in the interior of each polygon using a scan-line rendering algorithm.

The classic way to implement technique this is to loop over all edges of a polygon and calculate all of the (x,y) intersection points for each edge using a simple line drawing algorithm. At the same time, you need to calculate the interpolated surface normals (Nx,Ny,Nz) along the line, and perform Phong shading to calculate (r,g,b) color values for each of these points. All of this information should be stored in the scan-line data structure. For example, after processing three edges AB, BC, and BA in the polygon below, you should have calculated and stored (x,y), (Nx,Ny,Nz) and (r,g,b) information for each of the 12 scan-line end points below.

![Diagram of a polygon with scan-line rendering](image)

After all of the edges are processed, you then traverse the scan-line data structure and fill in the (r,gb) color values along each scan-line. Because the surface normals and (r,g,b) color values on either end of the scan-line are typically different, this will require more normal interpolation and Phong shading calculations as you move horizontally across the scan-line.

Your input file will contain P polygons, so you will need to repeat this scan-line rendering process for each polygon, and store your (r,g,b) color values in the image[500][500][3] array. When you have finished processing the input file, you can use glDrawPixels() to display this image on the screen. Congratulations! You have implemented a polygon rendering program from end to end.

2. Design:

Your first task will be to work out read the input data file. Since the “Light” and “Object” descriptors are fixed length, you could read the first word into a string, and
based on its value read either 7 light parameters or 8 object parameters into your program. The “Polygon” descriptors begin with a fixed length line with 2 parameters, followed by 6 N values that describe the N points in the polygon. You should verify that you can read all of this in correctly before working on Phong shading or scan-line rendering.

3. Implementation:

There are a number of sample programs in the OpenGL Examples folder that you are encouraged to read, and borrow from as you wish to create your program. In particular, you will want to look at “render.cpp” and “render2.cpp” to see how the image[500][500][3] array is used and displayed, and how a simple scan-line data structure can be implemented. The scan-line algorithm in render2.cpp interpolates just the (r,g,b) colors of polygon endpoints, and does not interpolate normals or perform any Phong shading calculations.

4. Testing:

Test your program to check that it displays your polygons correctly. You should probably do this incrementally, adding polygons to your input one at a time, and adjust the locations, sizes, and orientations of objects until you have completed the model. Finally, save a sample screen capture of your model to include in your project report.

5. Documentation:

When you have completed your program, write a short report using the project report template on the class website. Include a number of screen captures of your final geometric object, and your source code. Save this report in a PDF file to be submitted electronically.

6. Project Submission:

In this class, we will be using email project submission to make sure that all students hand their programming projects and labs on time. Send your completed project with two attachments, the project report and the source code, to the GTA lcweaver(at)email.uark.edu and CC the instructor jgauch(at)uark.edu. The dates on your email will be used to verify that you met the due date above. All late projects will receive reduced credit (10% off per day for three days, and zero credit after three days).