Aleksandra Anderson
PA-4 Documentation

Part A:

Solve area of 2D triangle:

- To solve for the area of a 2D triangle, given its 3 vertices I will use the following equation. Where x/y denotes the axis and the trailing number represents which point.
  - Area = ½ $|x1(y2 - y3) + x2(y3 - y1) + x3(y1 - y2)|$

Solve area of 3D triangle:

- To solve for the area of the 3D triangle, I will start by creating 2 vectors from the 3 vertices.
  - V = p1 - p2
  - W = p1 -p3
- The equation to find the area is; Area = ½ $||V \wedge W||$
  - Python Code : area = ½ (norm(cross(V,W)))

Solve distance from line (p1 & p2) to point p3

- To solve for the distance from the line created by p1 and p2, to point p3 I will first need to create the line from p1 & p2. This can be done with the parametric equation of the line, l = p + tV where V is p2 - p1.
- From here, the distance is calculated using the W vector, where W = p3 - p1. The distance from p3 to the line is $||W||\sin(\alpha)$.
- Python has a linear algebra library, with functions that aid in this type of computation. To solve for the distance in my program, I used the cross() function and the norm() function to create the line and find the distance to p3.
  - D = cross(p1-p2, p1-p3) / norm(p1-p3)
  - Cross takes the cross product of 2 vectors
  - Norm will normalize the vector.

Solve distance from the bisector plane formed by p1 & p2 to point p3

- First, I will find the midpoint of the line created by the 2 points (p1 & p2)
  - Add each component of p1 with each component of p2. Then divide each by 2.
- Second, I will find the normal vector of the bisecting plane.
  - N = (q-p) / ||q-p||
- Next, I will find the bisecting plane equation and solve for the distance to p3
  - Bp = n · (x-n); where x is p3

Testing

- My program utilizes a triangle class, which holds the information pertaining to the triangle being evaluated. It holds each point, encapsulating the data necessary to identify the area and distances of the triangle. As I read in the file, I store the vertex information into a triangle object. This allows me to easily access the information associated with each vertex and triangle.

- To test, I have created my own input files, using the proposed formatting. After running my program, I worked the problems by hand to ensure that calculations were performed accurately. This is absolutely necessary as I am utilizing python functions to handle the computation; double checking the outputs ensures that the functions are being used correctly.

Part B:

Set Up

- Part B uses a modified version of the Triangle class mentioned above.
  - Get / Set the vertices
  - Get / Set normals
  - Get / Set V & W Vectors
- After reading in the first line, set the eye location & light source from the first 6 numbers in their associated arrays.
- Read in the remaining lines, storing their vertices, normal and vectors into their specified object
  - Store each object into a list to be evaluated later

Culling

- Find centroid
  - X = (x1 + x2 + x3) / 3
  - Y = (y1 + y2 + y3) / 3
  - Z = (z1 + z2 + z3) / 3
- Use eye location
  - This value is retrieved from the first line of the file.
- V = (e -c) / ||e-c||
- If (n · v) < 0
  - Triangle is back facing

Light Intensity

- Lighting intensity ranges from 0 (light coming from the back side of the facet) & 1 (perpendicular to the facet)
- The intensity is measured by |cos(x)|
  - X = angle of incidence
    - Dot product between the normalized light direction and the facet normal
  - `angle_of_incidence=np.dot(normalized_light,facet_list[i].get_normal())`
  - `final_intensity = abs(math.cos(angle_of_incidence))`

Testing

- To Test this program I used various types of numbers for input. This included negative numbers, integers, and floating point numbers.
- Check each output for accuracy, ensuring that the code I wrote is evaluating the expressions accurately.