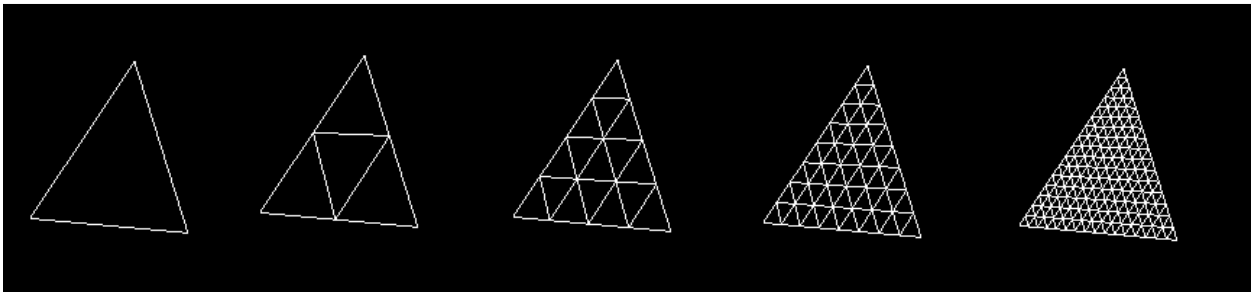
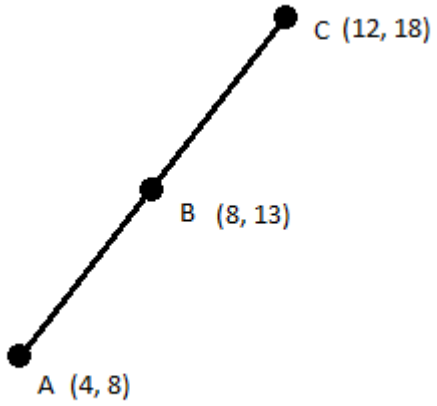
Gouraud Shading & Phong Shading

1. Draw a sphere with polygons
   One way you can do it is to start with a platonic solid with triangular sides - an octahedron, for example. Then, take each triangle and recursively break it up into smaller triangles, like so: (only one side of the octahedron is shown here)



Once you have a sufficient amount of points, you normalize their vectors so that they are all a constant distance from the center of the solid. This causes the sides to bulge out into a shape that resembles a sphere, with increasing smoothness as you increase the number of points.

Normalization here means moving a point so that its angle in relation to another point is the same, but the distance between them is different. Here's a two dimensional example.

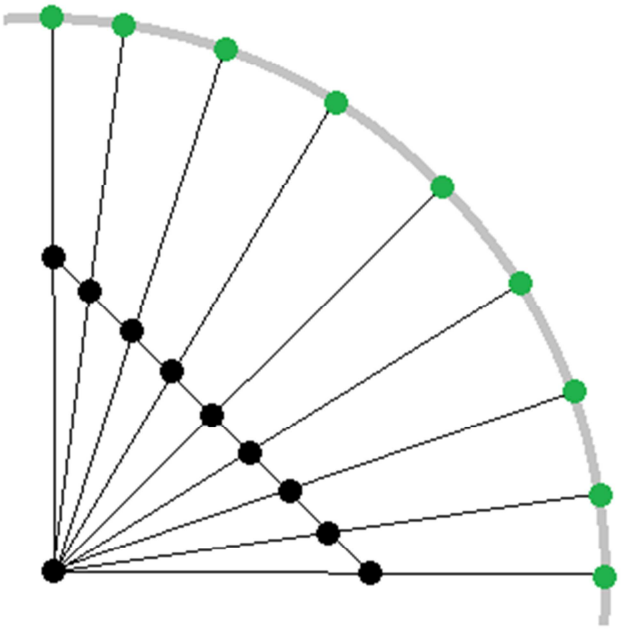


We can say that C is the normalized form of B with respect to A, with distance 12. We can obtain C with code like this:

```
#returns a point collinear to A and B, a given distance away from A.
function normalize(a, b, length):
    #get the distance between a and b along the x and y axes
```

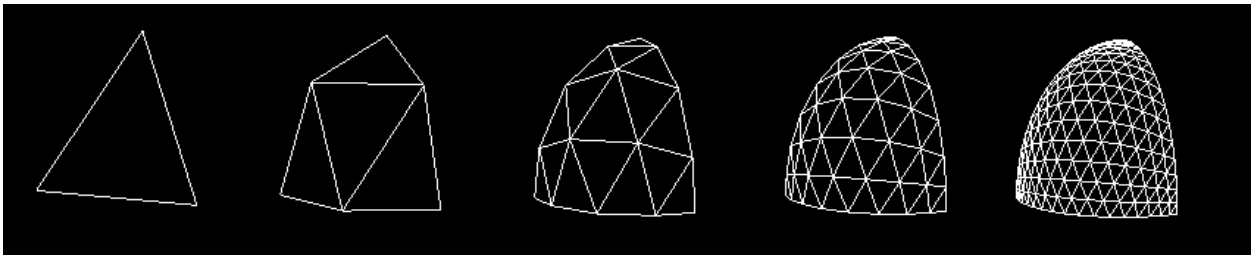
```
    dx = b.x - a.x
    dy = b.y - a.y
    #right now, sqrt(dx^2 + dy^2) = distance(a,b).
    #we want to modify them so that sqrt(dx^2 + dy^2) = the given length.
    dx = dx * length / distance(a,b)
    dy = dy * length / distance(a,b)
    point c =  new point
    c.x = a.x + dx
    c.y = a.y + dy
    return c
```

If we do this normalization process on a lot of points, all with respect to the same point A and with the same distance R, then the normalized points will all lie on the arc of a circle with center A and radius R.



The black points "bulge out" and become the green points on the circle.

This process can be extended into three dimensions, in which case you get a sphere rather than a circle. Just add a dz component to the normalize function.



2. Lighting Model
   When you have the skeleton of the sphere, you can apply the lighting equation to the Gouraud and Phong shading model. Calculate the intensity value for each pixel by interpolating either the light or the normal. You can assign the value of the parameters K_a, K_d, K_s, I_a, I_i and n yourself.

$$I_{total} = k_a I_{ambient} + \sum_{i=1}^{lights} I_i \left(k_d \left(\hat{N} \cdot \hat{L} \right) + k_s \left(\hat{V} \cdot \hat{R} \right)^{n_{shiney}} \right)$$

3. When you get the intensity value for each pixel, assign the intensities (total or specular) value to the glColor3f(r, g, b) depending on what color you want. Then draw each pixel with its color. Last, implement hidden surface removal and draw the shadow.

4. Other objects such as cube, cylinder and cone can be draw much easier using the similar method.

5. You are allowed to use glVertex3f() and OpenGL projection transformation functions such as glOrtho(), glFrustum() or gluPerspective() this time.