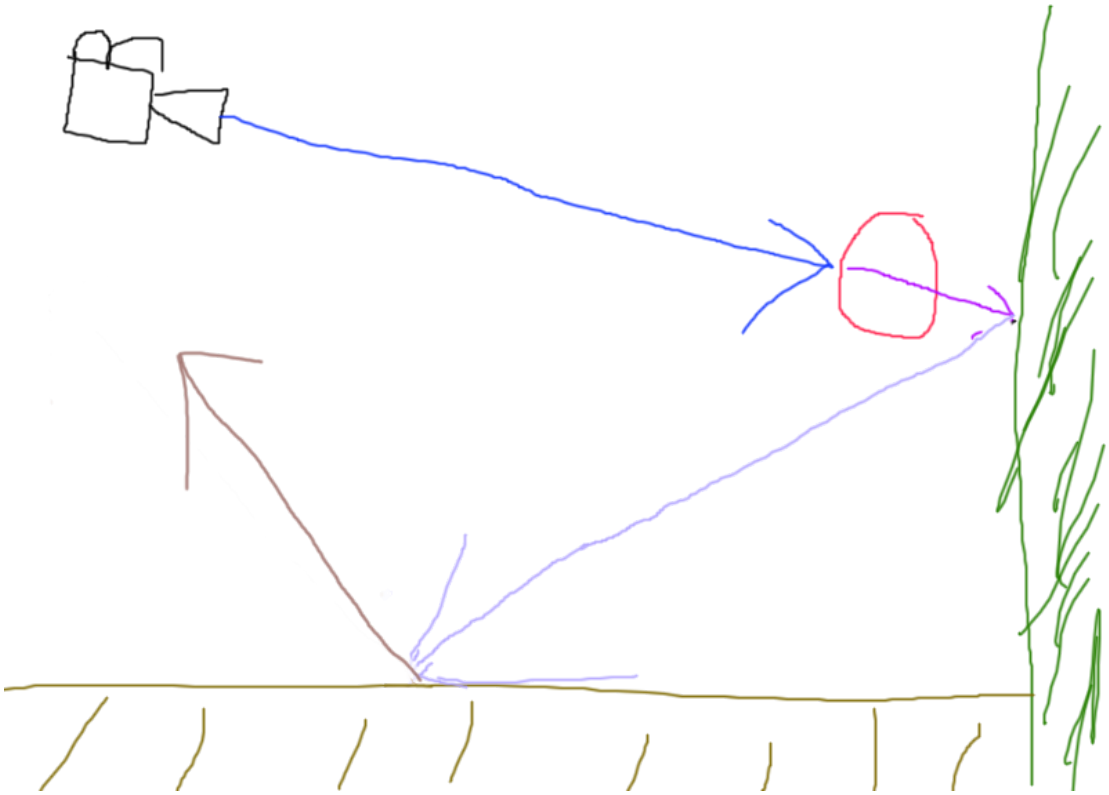
Global illumination or indirect illumination is the process where light is calculated based on all of the objects that a single ray of light is reflected from or bounces through. For objects that are transparent that means that the light ray would probably pass though the object, take the color of the object that it is passing though and add that color to the current light color and then dim it by some value since it probably wouldn't e a pure reflection as some of our light color/intensity would be absorbed by the material it hit. For surfaces that reflect the light ray, it would take the color of the object it hits, add that to a light ray and then bounce that colored light ray onto another surface. It would keep repeating this process until it's bounced a certain number of times probably specified in the shader. The more bounces probably means the more accurate light calculations but more computation time.

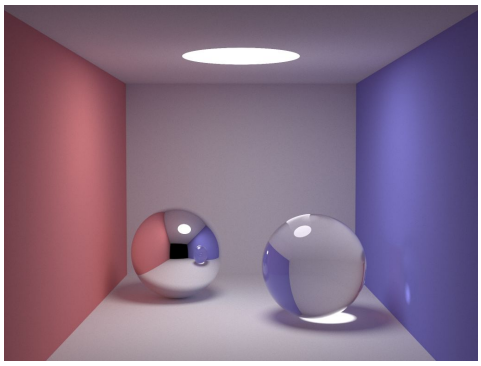
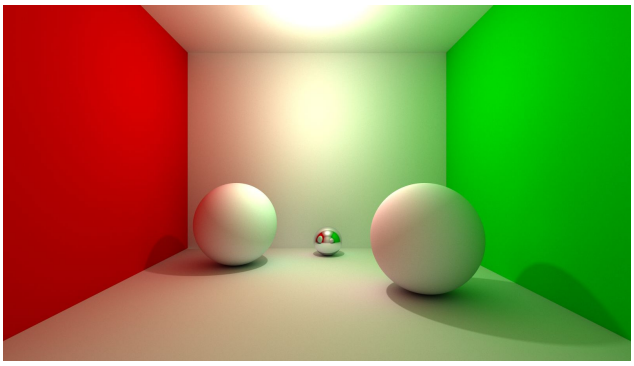
Here's a rough picture from how I understand this.



In this "scene" there are green and brown reflective surfaces, and a red transparent sphere.
The initial light ray is blue and everytime it passes through or hits something the color of the light ray changes a bit to cast the correct light color from the corresponding surface in which the light ray just hit. That way all objects are colored appropriately to the color of nearby objects. While this is a simple demonstration of how light would work in the scene, this could be vastly complex if we had a large amount of reflective surfaces.

In a shader you would probably be calculating a bunch of normals for reflection and refractions to determine the next direction of the light vector. How you would know all of the objects in the scene in the shader is a bit of a mystery but my best guess is that you would pass in a array of all of the object positions, use ray tracing to do some distance calculations, and then determine if the light ray intersects the object's position, probably using some signed distance functions. Using all of this with phong shading probably wouldn't be that hard as we would just probably use a while loop to calculate the color while we still needed to bounce around our scene. It would be interesting to see different ways in which we could go about this.

Here's a better example of how this would look using some images from the internet.
We can also observe that the reflective spheres do multiple reflections from the opposite sphere in the left photo. In the right photo we can notice that the floor takes a but of color from the nearby walls.

Team members:
Joshua Navarro (me)
Jenny Fullerton
Lisa Durand