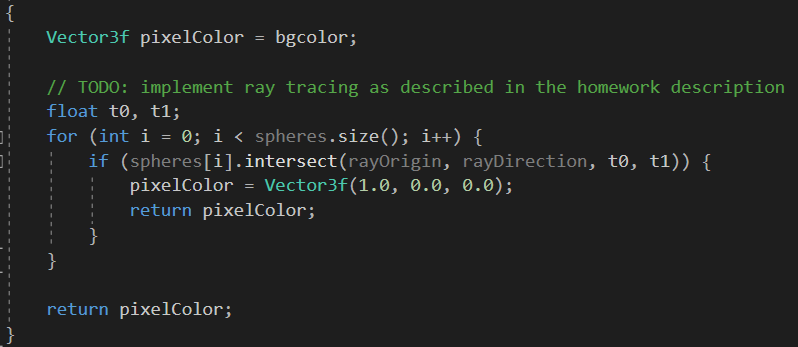
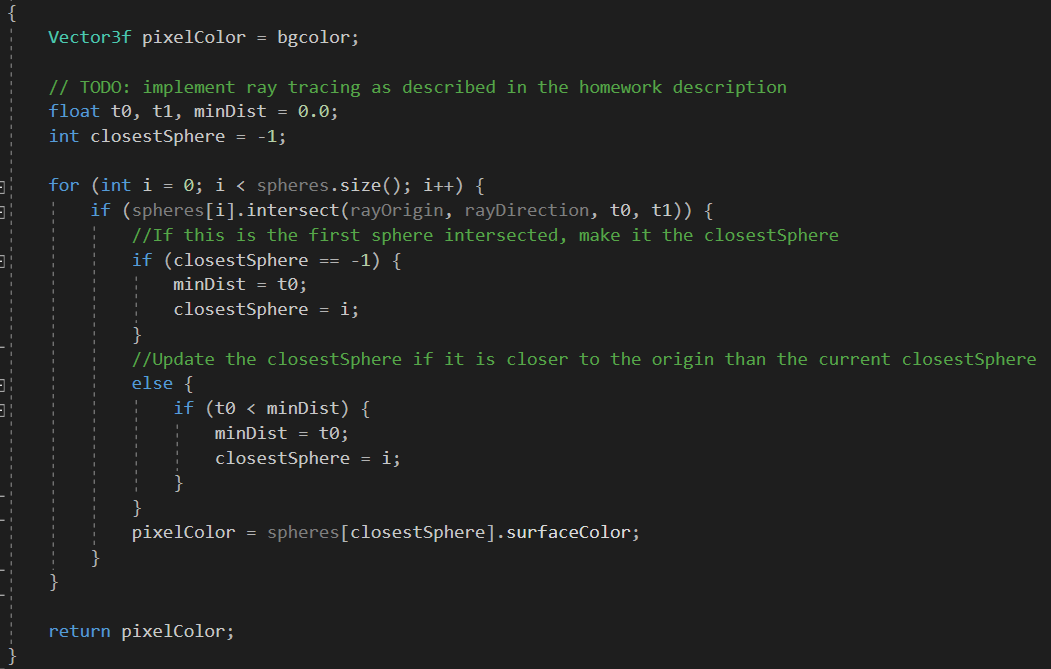
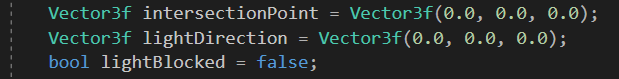
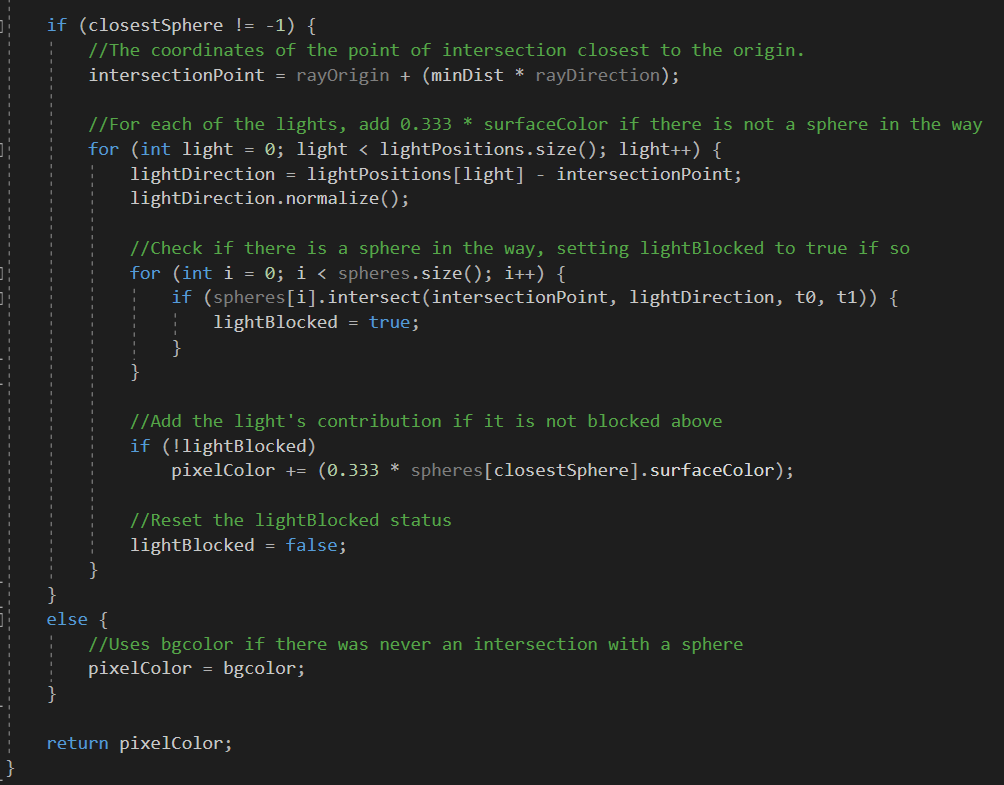
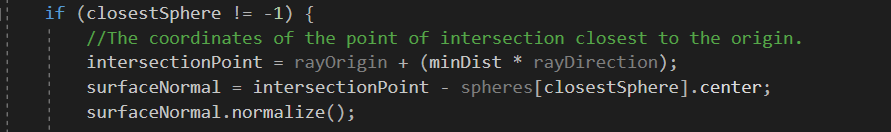
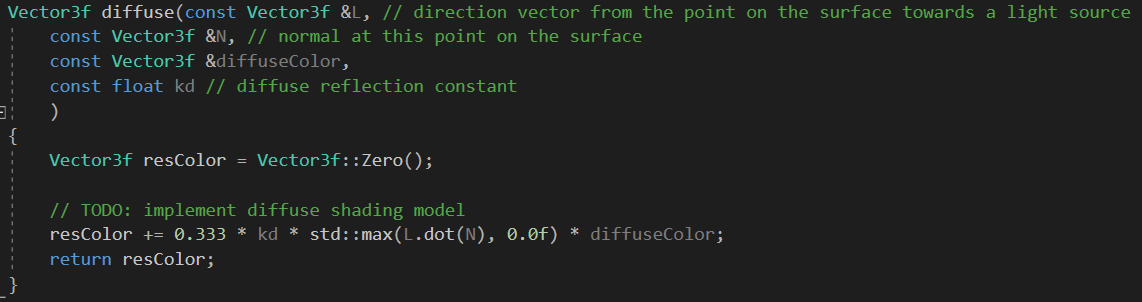
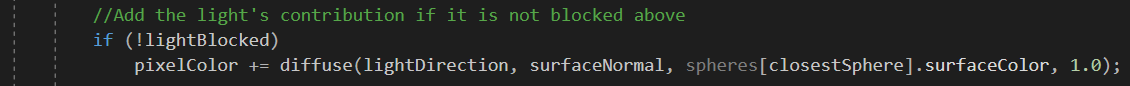
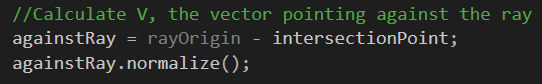
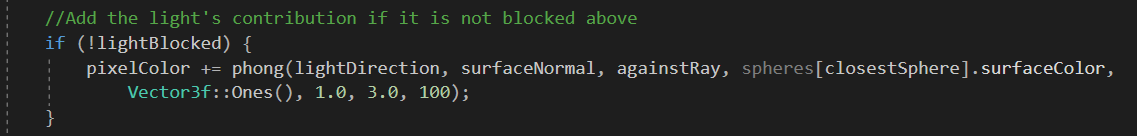
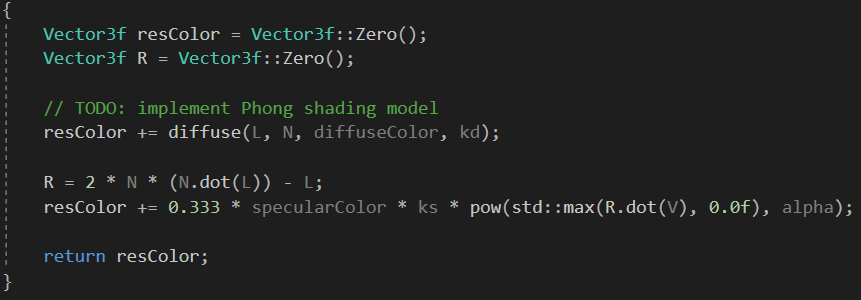
**Part 1:**

* For the first part, I started with a simple for loop that goes through each of the spheres, returning red if the ray intersects with any of them. Returns bgcolor otherwise
* To return the colors of the spheres intersected, I had to determine which sphere was in front. I did this by keeping track of the smallest t0 returned by intersect() and the sphere associated with it. After going through all spheres, I return the color of the closest sphere. 

**Part 2:**

* Using the same for loop from above to find the closestSphere, I then add 3 new variables to find the effect of the light on the spheres: 
* Once I know the closestSphere and the closest distance minDist (which is essentially the point visible from origin), I store the location of where the ray hit the closestSphere in intersectionPoint. Using this point as the new origin, I check if a ray from the point to each light intersects a sphere. If there are no intersections, the light contributes 0.333 \* surfaceColor to the pixel, creating shadows. 

**Part 3:**

* To implement the diffuse shading, the only additions I had to make to part 2 was a way to compute the surface normal at point P, and to put in the equation described in the pdf. For starters, I defined a new variable surfaceNormal: 
* I then computed the surface normal using the point P (intersection point) and the center of the sphere in question (closestSphere): 
* I now have L (lightDirection), N (surfaceNormal), diffuseColor (spheres[closestSphere].surfaceColor), and kd (1). With this information, I simply define diffuse using the equation in the pdf: 
* And finally, I call diffuse whenever I increment pixelColor, which is when there are no spheres blocking the light: 
* Next, for the complete Phong shading model, I am only lacking the parameter V, which I define as againstRay and compute below: 
* I now have every parameter needed to pass to phong, which is defined as the summation of the diffuse and specular components for each light. I do this similarly to previous steps: I just add what is returned by phong to the pixelColor
* To define phong, I first get the diffuse component by calling diffuse with the appropriate parameters. I then find the specular component by first finding the reflection vector R as defined in the lecture slides, and then plugging it into the equation (also defined in the slides): 
* And these were the only changes needed to implement phong shading. My final and complete implementation of trace is below: 