Ray Tracing Help

This document describes what you need to do in order to be able to write the ray tracer. In particular, how you can use what you have written with a bit of extra math to figure out where and if a ray intersects a triangle. I ordered it that way for a reason because the first step is to find where the ray intersects the plane of the triangle and then determine if that is actually in the triangle. The method described here isn't necessarily optimal, but it should be easy for you to understand and implement given what you have already written.

Definitions:
r0, r1 : These are the points on the ray. The ray starts at r0 and goes through r1. So that you can render the same image at many resolutions, we are going to scale the rays for the pixels on the screen. You should probably make r0 to be (0.5, 0.5, -1). Altering how big the z component is will alter the perspective of the image drawn. Also note that it is negative. The raster is at the plane z=0 and everything you are drawing has positive z. So each pixel might be given by the ray r0=<0.5,0.5,-1>, r1=<x/width,y/height,0>, where x,y is the point on the raster. Keep in mind that the calculations of x/width and y/width need to be done in floating point or you get zero. Your geometry should then almost all sit inside the positive unit cube with one corner at the origin.

t0, t1, t2 : These are three points on a triangle. We are going to be more precise about these points this time. They need to be arranged in a particular manner so that if we walked from t0 to t1, then to t2, then back to t0 our left hand side would always be towards the inside of the triangle. We also define these triangles as single sided. The cross product (t1-t0)×(t2-t0) gives us a normal that points to the side that we want the triangle to be facing.

L : is the point for the light.

Each of these points is actually a 3 vector with an x, y, and z component.

Process:
Given a ray and a triangle we want to see where they intersect, if they intersect. To do this we first find where the ray crosses the plane of the triangle and we do that by expressing the plane and the ray as equations. To get the plane of the triangle we define n=(t1-t0)×(t2-t0), so n is the normal vector to the triangle. It can be normalized if you want, but doesn't have to. Note that this is a vector cross product. If you don't know what that is, come talk to me or look it up, it isn't very complex and you can write a simple function to do one on two vectors. Other parts of the math will want a normalized normal vector so it might be useful to keep that. A point p is in the plane if (p-t0)·n=0. That is the dot product of the normal and the vector from one point on the triangle to the point. If we break this into components, that gives us

$$(px - t0x) * nx + (py - t0y) * ny + (pz - t0z) * nz = 0$$

Now, we want to find out where the ray intersects that. We can describe the ray as a parametric equation such that it is at r0 when the parameter, s, is zero and r1 when it is one. So r(s)=<r0x+s*(r1x-r0x), r0y+s*(r1y-r0y), r0z+s*(r1z-r0z)>. If we plug this into our equation for the plane using these values for px, py, and pz, we get a single equation in s that we can solve.

$$(r0x + s*(r1x - r0x) - t0x) * nx + (r0y + s*(r1y - r0y) - t0y) * ny + (r0z + s*(r1z - r0z) - t0z) * nz = 0$$

$$s*((r1x - r0x) * nx + (r1y - r0y) * ny + (r1z - r0z) * nz) + (r0x - t0x) * nx + (r0y - t0y) * ny + (r0z - t0z) * nz = 0$$

$$s = -\frac{(r0x - t0x) * nx + (r0y - t0y) * ny + (r0z - t0z) * nz}{(r1x - r0x) * nx + (r1y - r0y) * ny + (r1z - r0z) * nz}$$

Just solve for that s and plug it back into r(s) to get the location of the point of intersection. If s<0 then they don't really intersect and we can throw it away.

So now you have a point, but we need to see if it is in the triangle. This is why we were picky about the ordering of the points in the triangle. We want to take the cross product of each leg of the triangle with the intersection point then dot that with the normal vector for the triangle. If this is non-negative for all three legs, the point is inside the triangle. For example check if n·(t1-t0)×(r(s)-t0)<=0 then we check the others:

n·(t2-t1)×(r(s)-t1)<=0 and n·(t0-t2)×(r(s)-t2)<=0. Note the directions of the vectors (how we are doing the subtraction) it is extremely critical because doing it backwards gives you the opposite sign.

For a triangle, this just about finishes the work. If the point is in the triangle we simply put the point r(s) in a pass-by-reference argument and return a value saying they intersected. The calling function deals with this differently depending on what we are looking for. You might also want your function to return the value of s. The reason for this is that when you check if something is blocking a light, your ray will be from the point on the triangle to the light source. For something to be between the point and the light, the value of s will have to be between 0 and 1, not just greater than 0. So you will pass the intersect function the triangle you want to check, the vectors r(s) that you just found and L.

For spheres:
Spheres can look really cool in ray traced graphics, even our simple ones here. Of course, much of the above has to be changed for that. We could make a sphere from a large number of triangles, but you will find that the number of triangles required to make that look good is quite large and will slow the program down.

Similar to the triangles, we want a function that can tell us if a given ray intersects a sphere and where it intersects. The sphere is defined by its center and radius. What we want to do here is make an equation for the distance between the ray and the center of the sphere as a function of our parameter s. We find the first s such that the distance is equal to the radius and that is the point of intersection. If the center of the sphere is point c, then the equation we want is

$$r = \sqrt{(cx - rx(s))^2 + (cy - ry(s))^2 + (cz - rz(s))^2}$$

$$r = \sqrt{(cx - r0x - s*(r1x - r0x))^2 + (cy - r0y - s*(r1y - r0y))^2 + (cz - r0z - s*(r1z - r0z))^2}$$

$$r^2 = (cx - r0x - s*(r1x - r0x))^2 + (cy - r0y - s*(r1y - r0y))^2 + (cz - r0z - s*(r1z - r0z))^2$$

$$r^2 = (\Delta cx - s*\Delta rx)^2 + (\Delta cy - s*\Delta ry)^2 + (\Delta cz - s*\Delta rz)^2$$

$$r^2 = (\Delta cx^2 - 2*\Delta cx*s*\Delta rx + s^2*\Delta rx^2) + (\Delta cy^2 - 2*\Delta cy*s*\Delta ry + s^2*\Delta ry^2) + (\Delta cz^2 - 2*\Delta cz*s*\Delta rz + s^2*\Delta rz^2)$$

$$r^2 = s^2*(\Delta rx^2 + \Delta ry^2 + \Delta rz^2) - s*2*(\Delta cx*\Delta rx + \Delta cy*\Delta ry + \Delta cz*\Delta rz) + (\Delta cx^2 + \Delta cy^2 + \Delta cz^2)$$

$$s^2*(\Delta rx^2 + \Delta ry^2 + \Delta rz^2) - s*2*(\Delta cx*\Delta rx + \Delta cy*\Delta ry + \Delta cz*\Delta rz) + (\Delta cx^2 + \Delta cy^2 + \Delta cz^2 - r^2) = 0$$

Note that substitution between the 3rd and 4th lines to simplify things. Also, note that this last night is simply a quadratic equation. You learned how to solve any equation of the format $a*x^2+b*x+c=0$. That is given by

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In our case, x is s and a, b, and c are the expressions given above. Note that the factor of -2 is part of b. You can store them in variables called a, b, and c in your code to simplify it. If the value under the square root is negative, then the ray doesn't intersect the sphere. If it is zero or positive though, this gives us the value of s that we can stick into the expression for r(s) to find the point of intersection. We always want to use the – in the ± because we want the smaller value of s which will be the near side of the sphere.

Once you have this, the normal vector is simply given by r(s)-c, the vector from the center of the sphere to the point. Dividing by the radius of the sphere normalizes it.