CSCI441 Project 2 Technical Write up

Alexander Alvarez

May 2020

1 Introduction

As with most ray tracing techniques, reflection and refraction both aim to mimic the path of light in our physical universe. Luckily, these behaviors have been studied and formulated for many more years than computer graphics have been around, and so translating these laws of light over into ray tracing is relatively easy, given a fundamental understanding of the laws themselves.

2 Reflection

Reflection is a rather simple technique to implement. In Figure 1, the incoming ray of light is represented by vector I, while the normal of the reflecting surface is represented by N, and the reflected ray is represented by vector R. I and N are all we need to find R, as it is visually apparent in Figure one that all we must do is 'flip' I around N.

To do this, we may break down vectors I and R into their respective axis-aligned parts A and B (pictured in Figure 2). When we do this, we get
$I = A + B$
$R = A - B$
Thankfully, we know that $B = cos(\theta) * N$ when θ is the angle between N and I. When we plug this in for B, we get
$I = A + cos(\theta) * N$
$R = A - cos(\theta) * N$
We can rearrange the first of these equations to find A, which we can then plug into the second equation.
$A = I - cos(\theta) * N$
$R = I - cos(\theta) * N - cos(\theta) * N$
$R = I - 2cos(\theta)N$
We've now found R! Although, trigonometry is rarely the most efficient way of computing directions in computer graphics. Thankfully, we can substitute in $N * I$ for $cos(\theta)$, since we know that this is the relation between dot products and angles. We finally come to our final equation:

$$R = I - 2(N * I)N$$

Implementation from here is pretty straightforward. We use the direction of the incoming ray for I, and the normal of the hit for N. Our new ray will have its origin at the position of the hit, and we should set it's direction to R. Now we can recursively cast our new ray, and find the next hit. If the surface is another reflection, we are able to recurse again, until we have either hit a non-reflective surface, or we've reached a predetermined maximum number of reflections. This is well illustrated in Figure 3. We shade our final hit appropriately, and then use this color as our pixel's color. Done!

Well, maybe not quite, because often perfect reflection may not be the desired result, so we can keep going! If our reflecting surface has a reflection value $r \in [0, 1]$, where 0 is no reflection and 1 is total reflection, we can add the two sources of color together for each pixel. For our object's final color C, we can simply use the formula $C_{final} = (1 - r)C_{object} + (r)C_{reflection}$, taking care that we are correctly shading the colors in their respective locations before adding them together. We can even apply this formula recursively as well, to reflect off of multiple surfaces with multiple different values of r.

3 Refraction

Refraction is, admittedly, a much more complex technique, focused around *Snell's Law*. Refraction is an inherent property of light passing from one transparent medium to another, in which the angle of the light is bent according to 3 properties: θ, and each of the medium's *refractive indices*, n_1 and n_2. Snell's Law represents the relationship between these variables, and is written as
$sin(\theta_2)/sin(\theta_2) = n_1/n_2$
This relationship is best illustrated in Figure 1, where our ultimate goal is finding vector T. Using this equation, we can find
$T = (n_1/n_2)(I + cos(\theta_1)N) - Ncos(\theta_2)$
First, we can identify and replace of $cos(\theta_1) = N * I$, and $n_1/n_2 = n$. Using trig identities, we can also replace $cos(\theta_2) = \sqrt{1 - sin^2(\theta_2)}$. Our equation now looks like
$T = n(I + (N * I)N) - N\sqrt{1 - sin^2(theta_2)}$
From Snell's Law, we can find that
$sin(\theta_2) = (n_1/n_2)(sin(\theta_1) = n * sin(\theta_1)$
$sin^2(\theta_2) = n^2 * (1 - cos^2(\theta_1)$
Finally, we can reduce our equation down to its final form:
$T = nI + N(ncos(\theta_i) - \sqrt{1 - n^2 * (1 - cos^2(\theta_1))})$
Unfortunately, there is no way to completely reduce out of the square root, so this is what we have to work with. Also unfortunately, just because we now have our formula, doesn't mean we're done yet. When we implement this formula, we must first consider whether we are entering or exiting the surface that we are refracting, as $n_1 = 1$ will work when entering the surface, while $n_2 = 1$ will work when exiting the surface, with the other refractive index being that of the surface itself (as air has a refractive index of 1). Next, we must consider the

surface's *Total Internal Reflection.*

Total internal reflection is the amount of light that cannot escape out of the surface due to the size of θ_2 and the angle of its exit, well demonstrated in Figure 5. The angle at which light cannot escape the surface is referred to as the critical angle, anything larger than this angle will result in no exit ray. This is actually easily enough to check, as this will occur whenever the contents within the square root are negative, and will therefore never yield a real number. So, what happens when we hit this angle?

We do know that light must be conserved. This means that the total amount of light we send in must also be sent out. Then, if the light cannot be refracted, it must be reflected. The amount of light reflected can also be calculated using $n_1, n_2, \theta_1, and \theta_2$, using the *Fresnel equations.* Using the Fresnel equations, we can represent the total amount of reflected light as
$R1 = n_1 * cos(\theta_1) - n_2 * cos(\theta_2) / n_1 * cos(\theta_1) + n_2 * cos(\theta_2)$
$R2 = n_2 * cos(\theta_1) - n_1 * cos(\theta_2) / n_2 * cos(\theta_1) + n_1 * cos(\theta_2)$
$R = \frac{R1^2 + R2^2}{2}$
When the critical angle is reached, $R = 1$, otherwise, we use the Fresnel equations. Due to the conservation of light, $T + R = 1$, so the amount of light refracted is $T = 1 - R$. This results in familiar behaviour, such as seeing reflections in puddles or lakes or windows, which is demonstrated well in Figure 5. Almost there! We now can recursively cast these rays to find the color of our pixel, using the formula $C_{final} = (R)C_{reflection} + (T)C_{refraction}$. Note that we now must cast two rays instead of one, as we did with reflection, but the recursive nature stays intact. The refracted ray will have to refract at least once more to exit the object, but either ray may continue to reflect and/or reflect until the final hit is found, where appropriate shading should be applied, and the color sent back up in the recursion tree. Done!

4 Figures

Figure 1:

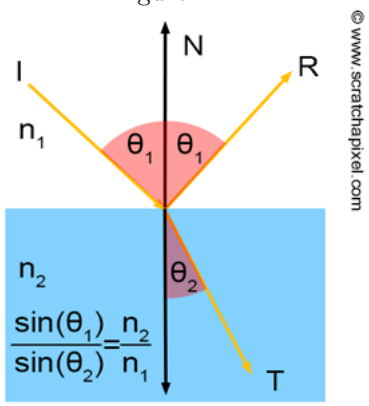

Figure 2:

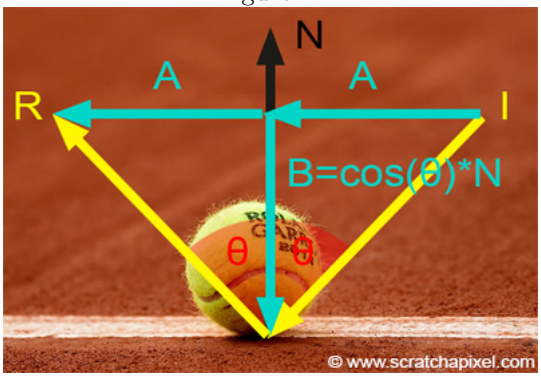

Figure 3:

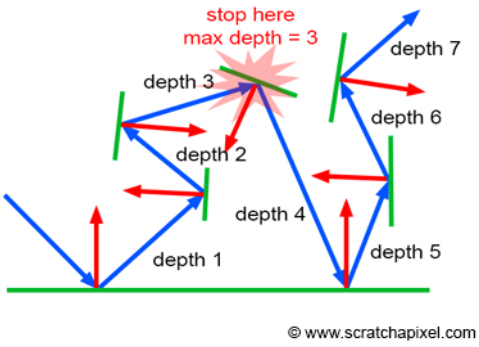

Figure 4:

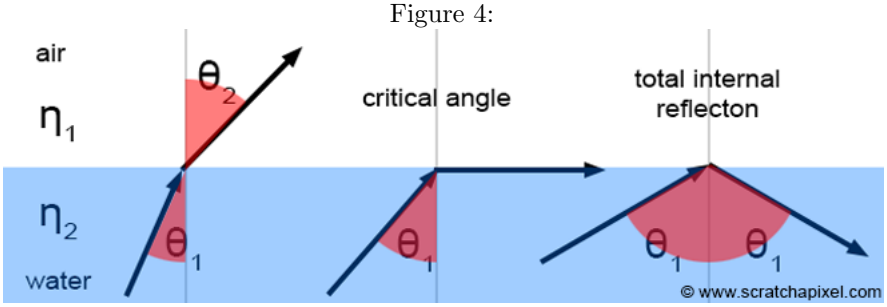

Figure 5: