Rendering Notes

Aaron Trent

May 4, 2017

Collection of equations and algorithms for use in the Combustion Engine, as well as general reference for developers wishing to understand the final equations and how I came to them.

This is a **work in progress** document.

Contents

Part I

Theory

1 Physically Based Rendering (PBR)

Physically Based Rendering (or Physically Based Shading) is simply the technique of using real-world-like parameters to represent a material, for example **Roughness**, **Anisotropy**, **Metalness**, **Index of Refraction** (η), and so forth.

Combined with newer forms of shading such as Cook-Torrance, Oren-Nayar and global illumination techniques, PBR allows easier creation of photorealistic scenes. If photorealism isn't the goal, it at least makes it easier for artists to work with, rather than arbitrary values such as a "shininess" exponent for the old Phong and Blinn-Phong shading models.

1.1 Physical Correctness

By itself, PBR does not guarantee a realistic image. The equations and algorithms used to render the image must be physically correct. That is, they must obey the laws of physics to properly simulate light.

2 Microfacet Theory

Microfacet theory describes the surface of an object as being composed of incredibly tiny mirrors called microfacets. The microfacets are oriented in different directions based on surface roughness. The more smooth the surface, the more mirrors line up with the surface normal, forming a more mirrored macroscopic surface. Rougher surfaces are equivalent to the microfacets being oriented in ever more random directions, up until a point of entirely rough surfaces where the microfacets are totally randomly oriented and scatter light in all directions.

Since microfacets themselves are less intuitive, they are usually abstracted to become microgeometry, which is the microscopic ridges and crevices of a surface, where roughness and smoothness are as we know them traditionally.

At this abstraction level, it makes more sense that rougher surfaces are blurrier, while smoother surfaces are shinier.

3 Fresnel Effect

"Everything has Fresnel!" - Internet

If you haven't heard this before, you have now. Everything has Fresnel. However, what is Fresnel?

Put simply, the Fresnel effect determines how much light is reflected off of or transmitted into a surface. Reflected light forms specular highlights, while transmitted light becomes diffuse reflections or subsurface scattering.

For transparent materials, like glass, transmitted light is also refracted and passes all the way through it.

The Fresnel effect is governed by three primary variables (excluding any vectors):

1. The Internal Index of Refraction (η_i)
2. The Extinction Coefficient (k)
3. The External Index of Refraction (η_o)

where the internal IOR (η_i) is the index of refraction of the material itself, the extinction coefficient (k) is effectively how conductive the material is, and the external IOR (η_o) is the index of refraction of the material the object is contained in.

Normally, the external IOR is approximately 1.0, because the IOR of Air is approximately 1.0, so many forms of the Fresnel equations simply omit it.

However, if you wish to render an object as it appears underwater, the external IOR must be set to approximately 1.33 (the IOR of Water), or it will appear incorrect.

For metals, the interal IOR and extinction coefficient form a complex IOR in the form:

$$\eta + ik$$

where the color of the metal can be derived from it's wavelength-specific IORs.

4 Wavelength Response

4.1 Computer Color

Color in computer graphics is primarily RGB. Despite HSV/HSL/YUV/etc. color representations, RGB remains the primary way to represent color in computers. The reason for this is that the human eye only perceives Red, Green and Blue colors for its three cone type. The science of human color perception is quite in-depth and not necessary here, but the gist of it is that humans see a range of wavelengths for each primary color, with perception strength in roughly Gaussian curve shapes.

4.2 RGB Wavelengths and Human Eyesight

Here are the wavelengths used in Combustion:

$$\begin{aligned} \text{Red} &= 620\text{nm} - 740\text{nm} \\ \text{Green} &= 495\text{nm} - 570\text{nm} \\ \text{Blue} &= 450\text{nm} - 495\text{nm} \end{aligned}$$

where the human eye's response to individual wavelengths is this:

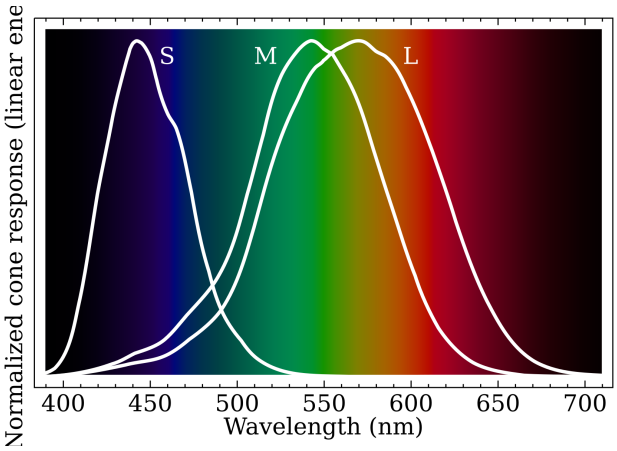
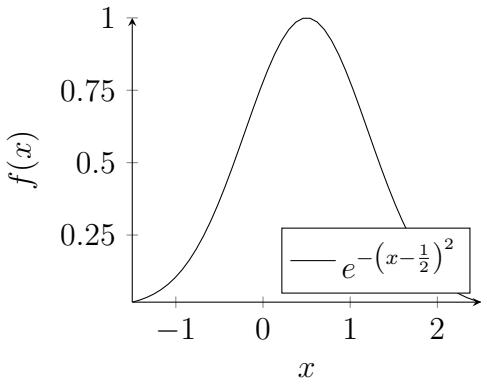


Figure 1: Human Eye Wavelength Response

which can be approximated using a modified Gaussian function like this:



to form:

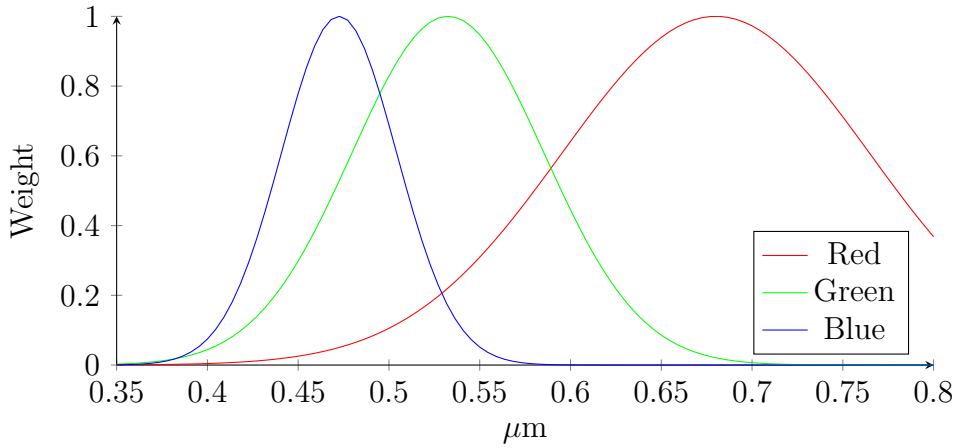


Figure 2: Weighted RGB Wavelength Response

which is similar to our perception of pure Red, Green and Blue. Note that they don't line up exactly, because there is a difference between wavelength and RGB color.

So by using this simple form to our advantage, we can perform a weighted average of any wavelength dependent optical properties like so:

$$V = \frac{\int_{w_{min}}^{w_{max}} f(x)\, p(x)\, dx}{(w_{max} - w_{min}) \int_{w_{min}}^{w_{max}} p(x)\, dx}$$

where $p(x)$ is:

$$p(x) = e^{-\left(\frac{x - w_{min}}{w_{max} - w_{min}} - \frac{1}{2}\right)^2}$$

or the original distribution scaled to the wavelength range like so:

$$p(\frac{x - w_{min}}{w_{max} - w_{min}}) = e^{-\left(x - \frac{1}{2}\right)^2}$$

Part II

Light Equations

5 Specular Reflectance

5.1 Phong

Phong and Blinn-Phong are less accurate than newer shading models, but are much cheaper to compute so they allow for greater performance at the cost of accuracy and ease of use.

5.2 Cook-Torrance

The Cook-Torrance equation is the core component for specular reflections today. It is flexible and offers much greater accuracy than older methods, at the cost of being more complex.

Variables for equations:

$N =$ unit surface normal
$V =$ view vector
$L =$ light vector
$H = \dfrac{V + L}{\|V + L\|} =$ halfway vector and microfacet normal
$\eta =$ Index of Refraction
$\alpha =$ surface roughness

It should be noted that these are ONLY specular reflections. That is, light which is reflected without diffusing into the material surface. For diffuse materials and how to mix diffuse and specular reflections, keep reading.

5.2.1 R. Cook and K. Torrance 1981

The original Cook-Torrance model, created by none other than R. Cook and K. Torrance in 1981, is originally defined as:

$$f_{spec} = \frac{DFG}{\pi\left(\omega_o \cdot n\right)\left(\omega_i \cdot n\right)}$$

where ω_o is the view direction and ω_i is the incoming light direction.

The idea was to take three terms representingseparate components of the surface model and combine them together to create an accurate result. These three components are the **Fresnel** term (as discussed in Section 7), the **Geometric Attenuation** function, and the **Microfacet Distribution** function.

The Geometric Attenuation function describes self-shadowing and self-masking of light from the microsurface of the material, like so:

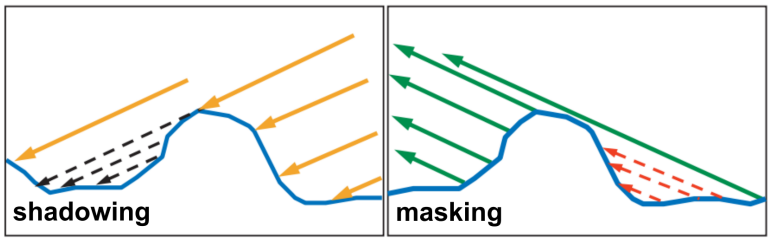


Figure 3: Shadow-Masking of light

For the original **Geometric Attenuation** function, G, these are combined into a single equation like so:

$$G_{Cook-Torrance} = min\left\{1, \frac{2\left(N \cdot H\right)\left(N \cdot V\right)}{\left(V \cdot H\right)}, \frac{2\left(N \cdot H\right)\left(N \cdot L\right)}{\left(V \cdot H\right)}\right\}$$

The **Microfacet Distribution**, D, is a probability distribution function (PDF) that describes the mean orientation of the microfacet normals. A well-known example of a PDF is the Standard or Gaussian Distribution. However, for a surface the roughness must be accounted for, so the distribution can be quite complex. In the original Cook-Torrance paper, the distribution function is the Beckmann Distribution.

D, the Beckmann distribution is defined as:

$$D_{Beckmann} = \frac{\exp\left(\frac{(n\cdot m)^2 - 1}{\alpha^2 (n\cdot m)^2}\right)}{\pi\alpha^2 (n \cdot m)^4}$$

where α is the surface roughness from $[0, 1]$.

Although discussed in more detail later, an approximation for the Fresnel term, F, was given in the original paper in the form of:

$$F_{Cook-Torrance} = \frac{1}{2}\frac{(g-c)^2}{(g+c)^2}\left\{1+\frac{[c(g+c)-1]^2}{[c(g-c)+1]^2}\right\}$$

where

$$c = \cos\theta = V \cdot H$$
$$g^2 = \eta^2 + c^2 - 1$$
$$\eta = \frac{1+\sqrt{F_0}}{1-\sqrt{F_0}}$$

and F_0 is defined as:

$$F_0 = \left\{\frac{\eta-1}{\eta+1}\right\}^2$$

where η is the Index of Refraction of the material. Depending on how you do things, F_0 can be avoided entirely by just using the raw IOR η.

Although slightly more accurate than Shlick's approximation (as discussed in section 7.2.1), it is more expensive to compute so it is largely unused for realtime rendering.

5.2.2 Walter 2007 (GGX)

Walter 2007 updates the Cook-Torrance model with a better set of equations that are commonly referred to as GGX.

GGX Cook-Torrance form:

$$f_{spec} = \frac{DFG}{4\left(\omega_o \cdot n\right)\left(\omega_i \cdot n\right)}$$

where G is split into two functions like so:

$$G = G_l(\omega_i)G_l(\omega_o)$$

Here, the Geometric Attenuation term is broken into two functions, collectively referred to as the Smith shadow-masking function. Each partial geometric attenuation function is the same, but is passed separate vector for light and view directions, then multiplied together.

Also note the replacement of π with just 4. I often see these used interchangeably around the internet, but they should remain distinct depending on how it is used. GGX works better with 4.

For the GGX equations, G_l is defined as:

$$G_l(v) = \chi^+\left(\frac{\omega_v \cdot \omega_g}{\omega_v \cdot \omega_m}\right)\frac{2}{1+\sqrt{1+\alpha^2\tan^2\theta_v}}$$

where ω_v is the view direction, ω_g is the unit surface normal, ω_m is the halfway vector, and ω_v is the incoming or outgoing direction.

The GGX Distribution function (D) is defined as:

$$D = \chi^+\left(n \cdot m\right)\frac{\alpha^2}{\pi\cos^4\theta(\alpha^2+\tan^2\theta)^2}$$

or simplified to:

$$D = \chi^+\left(n \cdot m\right)\frac{\alpha^2}{\pi\big(\left(n \cdot m\right)^2\left(\alpha^2-1\right)+1\big)^2}$$

with an anisotropic form defined as:

$$D = \chi^+\left(n \cdot m\right)\frac{1}{\pi\alpha_x\alpha_y}\frac{1}{\left(\frac{(x\cdot m)^2}{\alpha_x^2}+\frac{(y\cdot m)^2}{\alpha_y^2}+\left(n \cdot m\right)^2\right)^2}$$

as you can see, it takes two α (roughness) values and two anisotropy direction values (x and y) to control anisotropy.

6 Diffuse Reflectance

What light isn't directly reflected in the form of specular highlights is transmitted into the surface. However, most surfaces simply scatter the light below the surface and re-emit it in the form of colored diffuse light.

6.1 Lambertian Diffuse Model

For perfectly smooth surfaces, diffuse reflectance can be approximated by a single operation:

$$f_{diffuse} = N \cdot L = \|N\|\|L\| \cos\theta$$

where $\|N\|$ and $\|L\|$ are the magnitudes of the vectors. Consequently, because it is just $\cos\theta$ (when using normalized vectors), Lambertian diffuse has a reflectance curve like this:

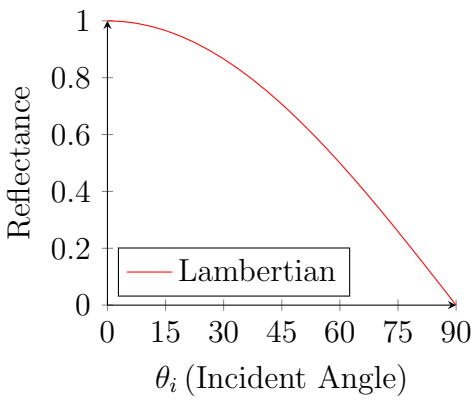


Figure 4: Lambertian reflectance

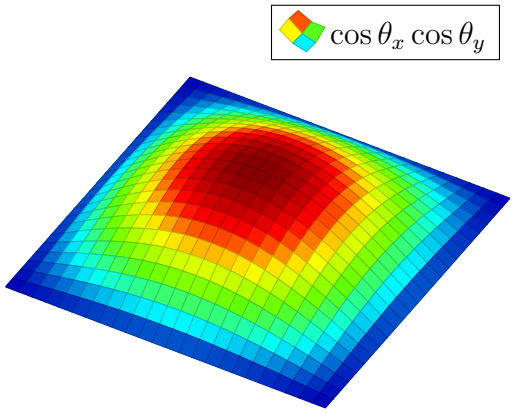


Figure 5: 3D plot

Lambertian is the most commonly used diffuse model mostly because of its simplicity. It's usually computed anyway for the specular component, so it's effectively free.

However, it does not account for surface roughness, so things get kind of weird looking in PBR heavy workflows.

You can see this effect more in Figure 6 on the next page.

6.2 Oren-Nayar Diffuse Model

Oren-Nayar is a "newer" (1994) set of equations based on empirical data that more accurately simulates surfaces of varying roughness.

Figure 6: Comparison of Lambertian and Oren-Nayar models

As you can see, Oren-Nayar does a much better job than traditional Lambertian diffuse. It more accurately simulates rough surfaces, and is therefore much more useful in modern PBR workflows.

The full form of the Oren-Nayar equation is defined as:

$$L_r = \frac{\rho}{\pi} \cdot \cos\theta_i \cdot (A + (B \cdot \max[0, \cos(\phi_i - \phi_r)] \cdot \sin\alpha \cdot \tan\beta)) \cdot E_0$$

where

$$A = 1 - 0.5\frac{\sigma^2}{\sigma^2 + 0.57}$$
$$B = 0.45\frac{\sigma^2}{\sigma^2 + 0.09}$$
$$\alpha = \max(\theta_i, \theta_r)$$
$$\beta = \min(\theta_i, \theta_r)$$

and ρ is the surface albedo (diffuse absorption), σ is the surface roughness, and ϕ_i and ϕ_r are the azimuth angles

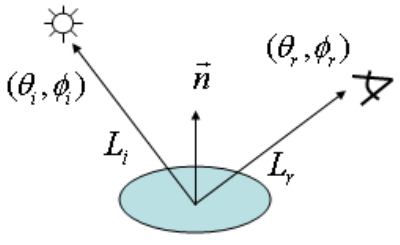


Figure 7: Reflectance Diagram

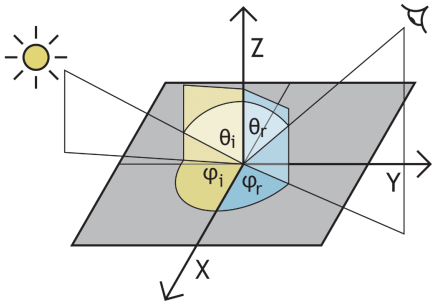


Figure 8: Angle Diagram

7 Fresnel Equations

The Fresnel equations determine how much light is reflected off of a surface, versus how much light is transmitted into it. Therefore, they are critical for correctly rendering any realistic material.

Most of these equations were taken from Memo on Fresnel equations by Sébastien Lagarde

The full Dielectric-Conductor Fresnel equations are as follows:

$$
\begin{aligned}
a^2 &= \frac{1}{2\eta_i^2}\left(\sqrt{\left(\eta_t^2 - k_t^2 - \eta_i^2\sin^2\theta\right)^2 + 4\eta_t^2 k_t^2} + \eta_t^2 - k_t^2 - \eta_i^2\sin^2\theta\right) \\
b^2 &= \frac{1}{2\eta_i^2}\left(\sqrt{\left(\eta_t^2 - k_t^2 - \eta_i^2\sin^2\theta\right)^2 + 4\eta_t^2 k_t^2} - \eta_t^2 + k_t^2 + \eta_i^2\sin^2\theta\right) \\
R_s &= \frac{a^2 + b^2 - 2a\cos\theta + \cos^2\theta}{a^2 + b^2 + 2a\cos\theta + \cos^2\theta} \\
R_p &= R_s\frac{a^2 + b^2 - 2a\sin\theta\tan\theta + \sin^2\theta\tan^2\theta}{a^2 + b^2 + 2a\sin\theta\tan\theta + \sin^2\theta\tan^2\theta}
\end{aligned}
$$

where

$$
\begin{aligned}
\eta_t &= \text{Surface IOR } (t \text{ for transmitted}) \\
k_t &= \text{Surface Extinction Coefficient} \\
\eta_i &= \text{External IOR } (i \text{ for incoming})
\end{aligned}
$$

Furthermore, this gives us two reflectance values, R_s and R_p, for parallel and perpendicular polarization of light. If light polarization is not important, these can be average together into a single reflectance value like so:

$$R = \frac{R_s + R_p}{2}$$

or if polarization is important, these values should be linearly interpolated like so:

$$R = (1 - \omega_t)\, R_s + \omega_t R_p$$

where ω_t is the polarity weight in the domain $[0, 1]$, so that 0 is fully perpendicular polarized light, and 1 is fully parallel polarized light. A weight of $\frac{1}{2}$ is equal to the first form where they are averaged together.

As mentioned in the Fresnel Effect theory section, the external IOR plays a very important role for scene that are not in air or vacuum. In air and/or vacuum, the external IOR is usually assumed to be 1 so the equation can be simplified, but for underwater scenes or something more exotic this parameter is absolutely necessary.

7.1 Metals

Metals are unique because they have no diffuse reflections, and therefore their entire color is derived from the Fresnel effect on varying wavelengths of light.

As you can see below in Figures 9 and 10, Gold behaves very differently depending on wavelength. It absorbs blue-ish light and reflects more red-ish light, producing its characteristic yellow color.

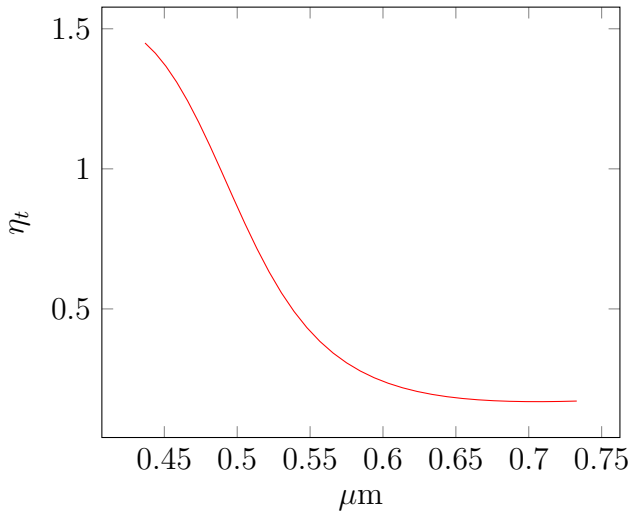


Figure 9: Gold η_t

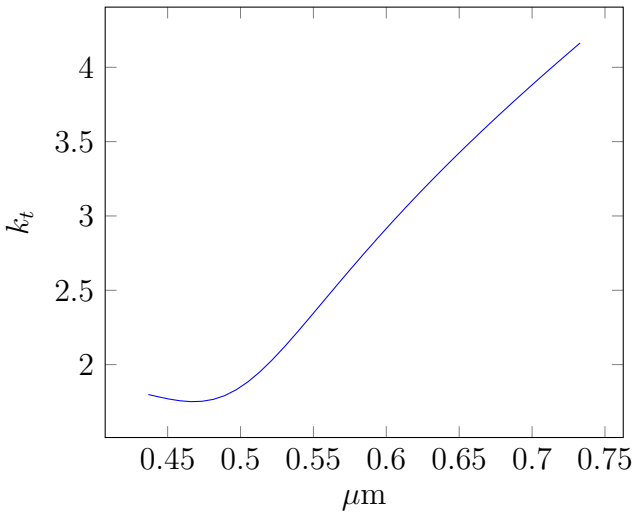


Figure 10: Gold k_t

Consequently, because metals have different behaviors for different wavelengths of light, their color is created by the varying reflectance values at a given angle.

As you can see below in Figures 11 and 12, Gold and Copper are given their unique colors by only the Fresnel effect, and reflecting more white light at greater grazing angle.

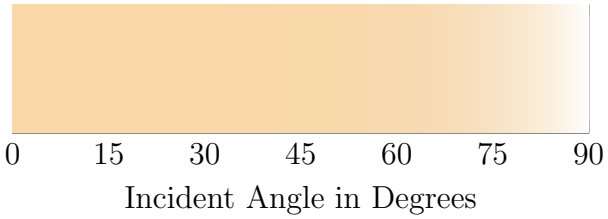


Figure 11: Gold (Unweighted)

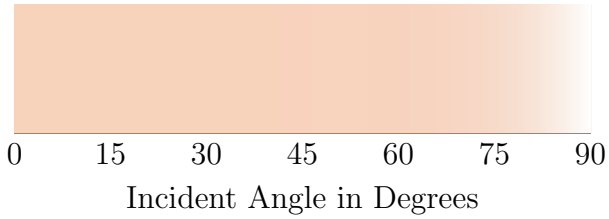


Figure 12: Copper (Unweighted)

You can further see how it affects individual spectra of light in Figure 13 on the next page.

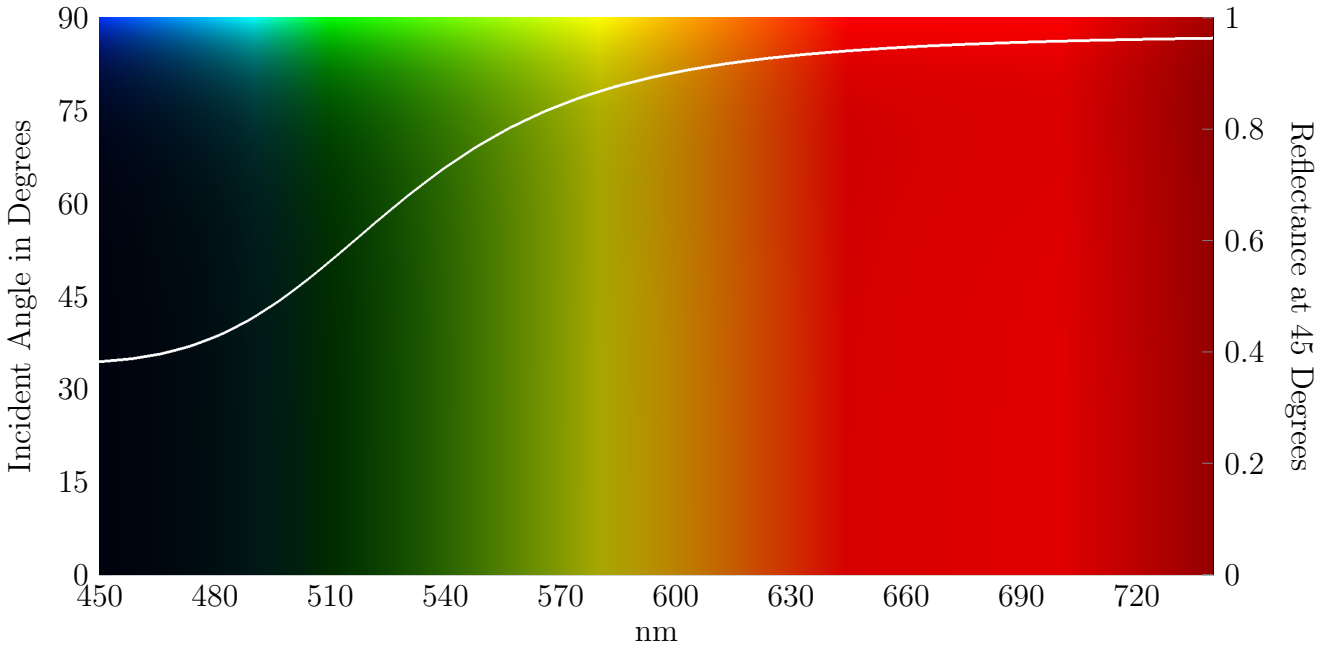


Figure 13: Gold Spectral Reflectance (R^3 for emphasis) at varying incident angles. The white line indicates reflectance at 45 Degrees.

It's important to understand the relationship all of these have together so you aren't tempted to ignore them for performance. For most of these, precomputed texture lookup tables can be used instead of computing the Fresnel effect for every fragment.

An example of such a lookup table for Copper is below in Figure 14.

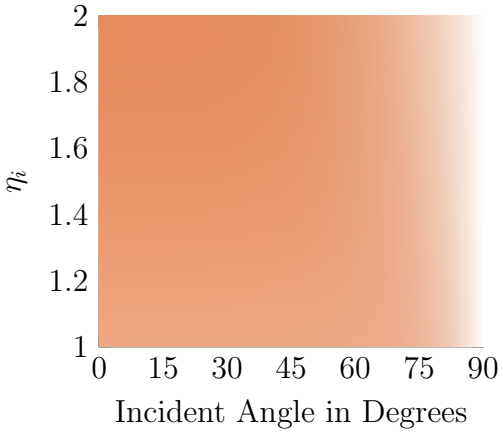


Figure 14: Copper Reflectance LUT with varying angles and external η_i

Where the LUT can be accessed with the following GLSL code:

TODO GLSL CODE

The only drawback of this method is that every metal needs its own lookup table texture, but unless you're going for as accurate as possible they don't need to be large, and can be automatically generated from the refractive index data sets.

Additionally, this implies that materials can be **ONLY** purely metallic or purely dielectric. This is true. However, this can be solved by simply thinking about it physically, and layering the materials. For example, a thin layer or grease over some steel surface could be represented as a linear interpolation of the metalic steel and the dielectric grease layer.

Multiple layers can be mixed together using linear interpolation. You can read more about that in Section 8 on mixing layers.

7.2 Dielectrics

Dielectric materials are all non-conductive materials, such as plastics, glass, most liquids, wood and basically everything non-metal.

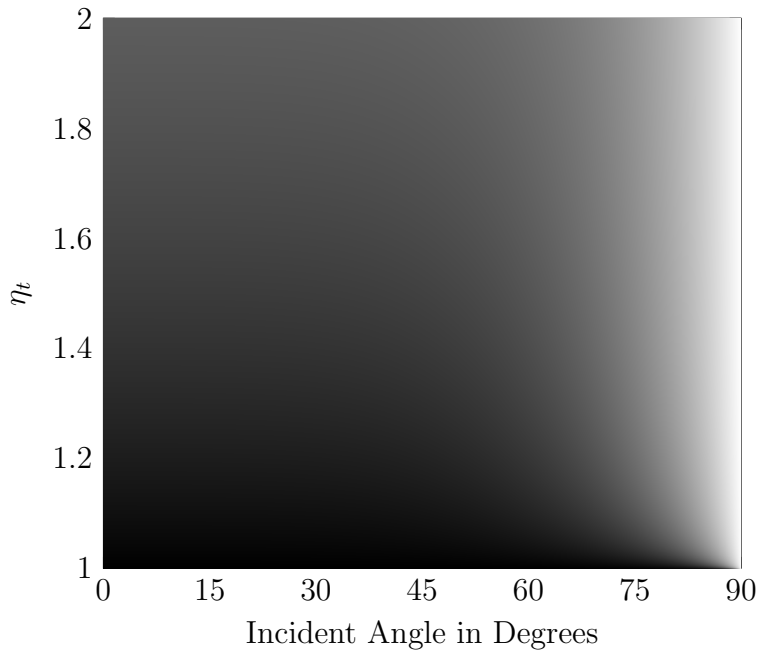


Figure 15: Dielectric Reflectance for varying η_t and incident angles

For dielectric materials, the extinction coefficient (k_t) term in the Fresnel equation is **always** set to zero. This is because it simply doesn't conduct absorbed light across its surface. Instead, dielectric materials absorb the light photons into the atoms and molecules that make up the material, and then usually re-emit the energy as more photons with a wavelength determined by the material. This is how all dielectric materials are colored.

7.2.1 Schlick's Approximation

For dielectric materials, there exists a decent approximation of the Fresnel equations that is very simple. Simple enough it can be run directly on the GPU without impacting performance too much.

Schlick's Approximation of the Fresnel Equations is defined like so:

$$R = F_0 + (1 - F_0)\,(1 - \cos\theta)^5$$

where F_0 is defined as:

$$F_0 = \frac{\eta_i - \eta_t}{\eta_i + \eta_t}^2$$

where θ is the incident angle, and η_t and η_i are the surface and external IORs, respectively.

Part III

Material Composition

8 Mixing Layers

Multiple layers of materials can be mixed together using linear interpolation like so:

$$L_{final} = (1 - \omega_{thickness}) (L_{bottom}) + \omega_{thickness} L_{top}$$

where L_{bottom} can be an L_{final} from another layer mix, and $\omega_{thickness}$ is the thickness of the layer in the domain $[0, 1]$

Part IV

Rendering

9 Pixels

Pixels are the physical component to any image, meaning they have discrete size in real life. This discreteness means that every pixel lies on an whole integer grid the size of the image, and any single pixel can only be a single value (usually a color). This is an important distinction because when mapping 3D points onto a 2D image, you must choose the nearest whole integer (x, y) coordinate, losing some accuracy.

Essentially, any scene rendered must be divided into a discrete grid and sampled with whole integer locations.

Supersampling and multisampling alleviate this somewhat by taking multiple samples per pixel in slightly difference sub-pixel locations, then averaging their values together.

10 Framebuffers

A framebuffer is a simple collection of images used to store the rendering result. Most framebuffers contain a color component, for the colors that will actually be shown, and a depth component, to store the depth of a rendered object from the camera, which is used for hiding geometry behind other geometry.

Additionally, a stencil buffer may be present, which allows storing an outline of sorts of rendered geometry with a given numeric value. Any pixel that geometry touches is given the desired value in the stencil buffer.

10.1 Blending

An important part of rendering to a framebuffer is color blending, usually alpha blending. For framebuffers with a color component that contains an alpha/transparency channel, for geometry to show up behind transparent or translucent geometry, the color values must be blended together in a way that makes sense, and emulates the behavior of light as it is filtered and/or transmitted through the mediums.

11 Transformations

By themselves, all 3D geometry is pretty useless to a rendering engine. How is it supposed to take those points in 3D space and show them on a screen?

If you've ever used OpenGL/DirectX/Anything before, you'll know the answer to that is to do it yourself.

Most rendering pipelines have what is generally referred to as a **Vertex Shader**, which is tiny program that accepts a single point in 3D space, and is expected to turn that into something that can almost be displayed on the screen.

Almost always, the vertex shader is:

$$V_{clip} = M_{projection} \cdot M_{camera} \cdot M_{object} \cdot V_{local}$$

where

$$\begin{aligned} V_{world} &= M_{object} \cdot V_{local} \\ V_{camera} &= M_{camera} \cdot V_{world} \\ V_{clip} &= M_{projection} \cdot V_{camera} \end{aligned}$$

As you can see, all vertex transformations go through a few stages, using a few different transformation matrices.

The first stage is to take the raw vertex positions, which is usually whatever is given in the mesh, and transform it based on the mesh's position in the 3D world. This is known as transforming the model-space vertex to world-space.

The second stage is to transform it to how the camera sees it using the camera matrix M_{camera}, which is usually just some translations and rotations based on where you want the camera to look.

The third stage is the most important: Projection. Projection is the act of taking the final 3D coordinates and calculating their 2D positions, usually using some form of perspective distortion. The projection stage does all the hard work of transforming the vertices into something that can be rendered in 2D.

11.1 Normals

Alongside the vertex positions, most vertices also have normal vectors which are essential for shading. Like with positions, normals need to be transformed into world-space as well, but because vector math is tricky, it needs to be done using sort of an odd matrix.

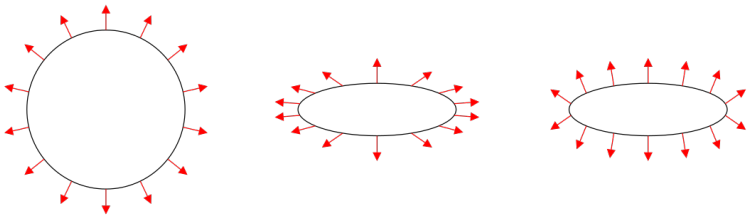


Figure 16: Normal Transformations

As you can see in the above figure, in the center image, simply applying the raw M_{object} transformation matrix to the normal vectors would squish and distort the normals, which is bad. What we really want is the right-most illustration, which preserves the normals.

As it turns out, doing this is as simply as transforming the normal vectors by the inverse transpose of the M_{object} matrix.

That is to say:

$$M_{mit} = \left(M_{object}^{-1}\right)^T$$

and then:

$$N_{world} = M_{mit} \cdot N_{local}$$

11.2 Precomputing Matrices

Since the vertex shader is evaluated across all vertices, and matrix multiplication is somewhat expensive, it's usually best to pre-compute the transformation matrices, like so:

$$M_{mvp} = M_{projection} \cdot M_{camera} \cdot M_{model}$$

and then:

$$V_{clip} = M_{mvp} \cdot V_{local}$$

which should improve floating point precision if the matrices are precomputed on the CPU, and improve performance by performing fewer matrix multiplications in the vertex shader.

12 Rasterization

TODO