

# Ramamoorthi and Hanrahan - “An Efficient Representation for Irradiance Maps” An Implementation In OpenGL

Anderson Liu

The paper by Ramamoorthi and Hanrahan describes a process that allows for an analytical approximation of the lighting distribution of a given environment map by precomputing 9 parameters representing the first 9 moments of the lighting.

The paper goes into great detail as to how to implement such a process. First, I started by computing and declaring some constants. I computed  $\hat{A}_l$  as specified in the paper for  $0 \leq l \leq 2$ . We only need to compute  $l \leq 2$  because of how quickly  $\hat{A}$  falls off. I then compute the real spherical harmonics coefficients  $Y_{l,m}$  for  $l \leq 2$ ,  $-l \leq m \leq l$ . Although  $Y_{l,m}$  depends on the coordinates  $(x, y, z)$ , there are some constants that can still be precomputed so I store these in an array.

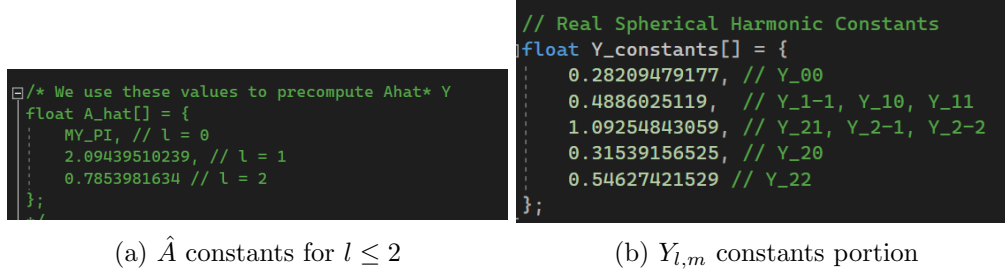


Figure 1: Initializing constants

The irradiance in terms of  $\theta$  and  $\phi$   $E(\theta, \phi)$  is given by:

$$E(\theta, \phi) = \sum_{l,m} \hat{A} Y_{l,m}(\theta, \phi) L_{l,m} \quad (1)$$

The paper expands this formula in order to get an equation for the irradiance  $E$  in terms of the normal  $n$ :

$$E(n) = n^T M n \quad (2)$$

where  $M$  is some matrix of coefficients.

They then define the matrix  $M$  in terms of some sets of coefficients  $c_n$  and  $L_{l,m}$  which represent the lighting coefficients.

The  $c$  coefficients are found by multiplying  $\hat{A}_l$  by the constant coefficients of  $Y_{l,m}$  and so we can precompute these constants.

$$M = \begin{pmatrix} c_1 L_{22} & c_1 L_{2-2} & c_1 L_{21} & c_2 L_{11} \\ c_1 L_{2-2} & -c_1 L_{22} & c_1 L_{2-1} & c_2 L_{1-1} \\ c_1 L_{21} & c_1 L_{2-1} & c_3 L_{20} & c_2 L_{10} \\ c_2 L_{11} & c_2 L_{1-1} & c_2 L_{10} & c_4 L_{00} - c_5 L_{20} \end{pmatrix}$$

$$c_1 = 0.429043 \quad c_2 = 0.511664$$

$$c_3 = 0.743125 \quad c_4 = 0.886227 \quad c_5 = 0.247708 \quad (12)$$

(a) The matrix  $M$  described in the paper

```
// For finding irradiance E = summation ( Ahat * L_coeff * Y)
// we can simplify by precomputing Ahat * Y
Float c[] = {
    0.4290427654, // A_2 * Y_22
    0.51166335396, // A_1 * Y_1m / 2
    0.74312386829, // A_2 * Y_20 * 3
    0.88622692544, // A_0 * Y_00
    0.24770795609, // A_2 * Y_20
};
```

(b)  $c_n$  coefficients

Figure 2: Finding values for  $c_n$

The most complicated process is finding  $L_{l,m}$ . Here we have to compute a surface integral by iterating through all the pixels of the environment map. The integral is as follows:

$$L_{l,m} = \int_{\theta=0}^{\pi} \int_{\phi=0}^{2\pi} L(\theta, \phi) Y_{l,m}(\theta, \phi) \sin \theta d\theta d\phi \quad (3)$$

To find this integral, we loop through all pixels of the light probe and sample the color at the pixel in order to find  $L(\theta, \phi)$ . In this case, we can separate the channels to compute the final matrix  $M$  needed per color channel.

To find  $\theta$  and  $\phi$ , I followed Paul Debevec's notes. Basically, instead of having the coordinates be from  $[0, width]$ , we normalize the coordinates and solve for  $u, v$  which are in the range  $[-1, 1]$ . Then, the radius  $r$  is simply  $\sqrt{u^2 + v^2}$  and  $\theta$  and  $\phi$  can also be solved.

$$\theta = \pi r, \quad \phi = \text{atan2}\left(\frac{v}{u}\right) + \pi \quad (4)$$

This can then help us find  $x, y, z$  which is needed to compute  $Y_{l,m}(\theta, \phi)$ . Where  $(x, y, z)$  is given by  $(\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta)$ .

For each pixel, for each color channel, we increment the 9 coefficients  $L_{l,m}$  given by  $l \leq 2$  and  $-l \leq m \leq l$  in order to get the entire coefficient.

There is still one part to talk about, though, and that is the solid angle  $\sin \theta d\theta d\phi$ , otherwise known as  $d\omega$ . Here, my process was to simply compute this as normal:

---

```
// compute sin(theta) dtheta dphi
// size = image_width
float sin_dtheta_dphi = sin(theta) * MY_PI / size * TWO_PI / size;
```

---

My reasoning for using this formula was simply because we're discretizing  $\theta$  and  $\phi$  based on the number of pixels, so since  $\theta$  goes from 0 to  $\pi$  and  $\phi$  goes from 0 to  $2\pi$ ,  $d\theta$  should be  $\frac{\pi}{width}$  and  $d\phi$  should be  $\frac{2\pi}{width}$ .

However, using this approach did not yield the same results as the ones produced by the paper. They were very similar, but not quite the same. I couldn't figure out what was causing this difference in results. In the end, I had to refer to the source code provided to see that in computing the solid angle, they multiply the expression I got by a factor of  $\frac{2}{\theta}$  and instead compute the solid angle like so:

---

```
// compute sin(theta) dtheta dphi
// size = image_width
float sin_dtheta_dphi = sinc(theta) * TWO_PI / size * TWO_PI / size;
```

---

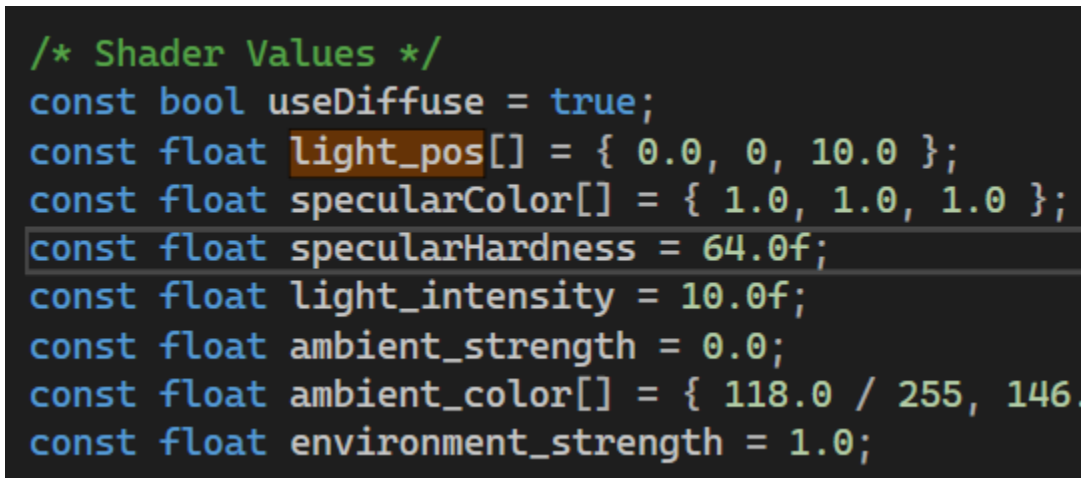
While I'm unsure as to why they do this, but I have two ideas that could be correct. First,  $\frac{2}{\theta}$  is some sort of scaling factor where the solid angle is inversely related to  $\theta$ . In other words, if  $\theta$  is small, then the light is more spread out.

Another possibility is that it has something to do with the coordinate system and the factor is something to do with the normalization of the coordinates. I'm basing this idea off the fact that 2 is the length of the range  $[-1, 1]$  of  $u$  and  $v$ , and then  $\theta$  would be the divisor that splits the range.

Then finally, after we compute all our coefficients, we can bundle them into matrices as specified by the paper to pass them to the shaders.

My shader is pretty simple, just taking the matrix  $M$  for each color channel and applying the irradiance formula mentioned above. I also set up a simple Blinn-Phong diffuse shader and added it to the irradiance.

The Blinn-Phong shader can be toggled on and off and other shader values can be changed around. Also, there are a few light probe images that can be used for the environment map.



```
/* Shader Values */
const bool useDiffuse = true;
const float light_pos[] = { 0.0, 0, 10.0 };
const float specularColor[] = { 1.0, 1.0, 1.0 };
const float specularHardness = 64.0f;
const float light_intensity = 10.0f;
const float ambient_strength = 0.0;
const float ambient_color[] = { 118.0 / 255, 146.0 / 255, 108.0 / 255 };
const float environment_strength = 1.0;
```

Figure 3: All the shader parameters

The algorithm itself works really well and is really fast at computing the lighting coefficients. I was surprised at how well it worked. But, there's a lot of things that could be done to improve this. The main thing is that this approach can be combined with shadows and ambient occlusion to create a more realistic scene. Other issues are that some light probes yield super high values and so it's hard to adjust the shader parameters correctly as I just have an environment strength variable that scales the values. A better way to handle this is probably to have some sort of normalization system so that everything doesn't just turn white.