# 2.15

# Reduction of Lighting Calculations Using Spherical Harmonics

***Vlad Stamate***

## Introduction

In today's real time applications, whether it's games or other types, we often find scenes where light is emitted from a large variety of sources. Those light sources must be modeled accurately in order to render the scene realistically. For example, if the Phong lighting model is used (Equation 2.15.1) would have to be resolved for all light emitters at the pixel level in order to achieve convincing lighting:

$$L = Ambient + Attenuation * (Diffuse + Specular) \tag{2.15.1}$$

where each member is in turn a complex equation.

This is too time-consuming to be done in situations where there are many light sources and the amount of geometry in the scene is considerable. What we propose is a method to identify a small number of important lights and only calculate the full lighting equation for those. The remaining lights would only have their diffuse contribution added to the scene through simpler calculations.

Choosing which lights are important can be tricky and depends very much on the rendered scene. But generally high-intensity lights, lights with a biased color component (high chromacity), or lights that are generally very close to most of the geometry can be chosen. For those lights we apply the full lighting equation.

The remaining lights are projected onto an environment map (be it cubemap or sphere map) and the first nine coefficients of the second order Spherical Harmonics (SH) are then extracted out of the environment map[1]. The process of calculating the SH coefficients is done only once for all the lights and by integrating across the environment map. At the lighting calculation stage we will perform the full lighting calculation for the "important" lights and then apply diffuse lighting based on the Spherical Harmonics coefficients and the surface normals.

[1]As explained later in the article, we can sometimes get away without actually doing any projection by going directly from light source information to Spherical Harmonics coefficients.

## Spherical Harmonics Based Lighting

Spherical Harmonics (SH) have recently seen a resurgence in the field of computer graphics. The interest in them has been boosted by the latest generations of graphic hardware which can execute complex fragment and vertex programs on the GPU, as well as the capability of reading and writing to floating point texture formats. One common problem sometimes associated with the use of SH is the complexity of the math involved. The aim of this article is not to give a full mathematical demonstration of the theory but rather to provide a "hands-on" view of the technique. However, some background is useful.

Irradiance, which is the amount of light incident onto a point, can be expressed in terms of the lighting environment as the following integral:

$$E(n) = \int (n \bullet w) L(w) dw \tag{2.15.2}$$

In layman terms this integral means that the irradiance $E$ is a function of the normal (of the surface we try to light). The irradiance is computed by integrating (summing) the incoming light, scaled by the dot product of the normal of the surface $n$ and the incoming light direction $w$, across the whole environment map $L(w)$. We consider $w$ and $n$ to be normalized vectors as shown in Figure 2.15.1.

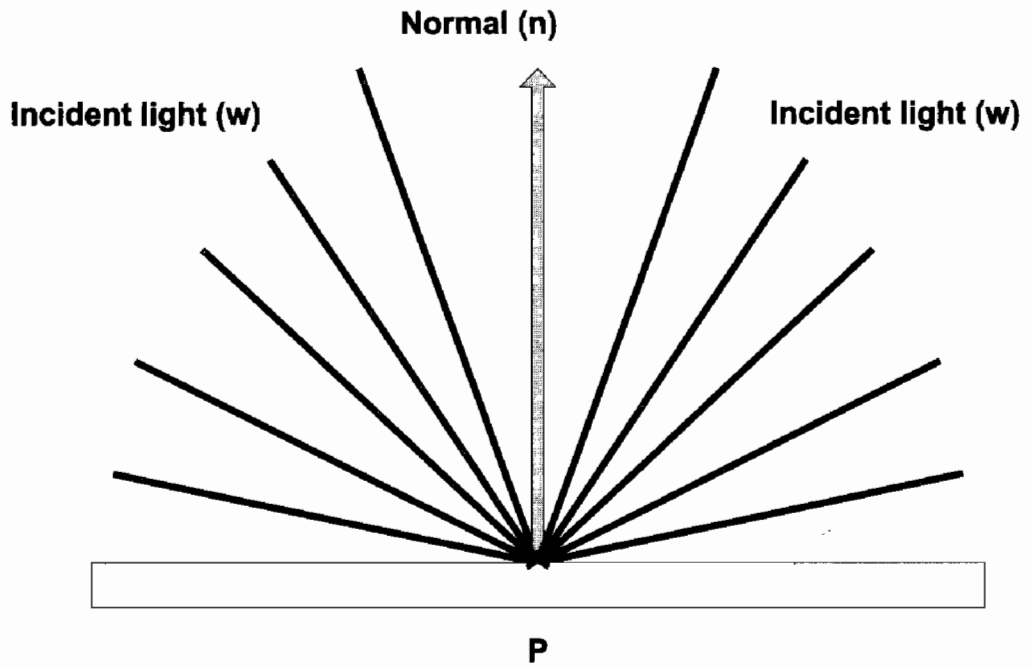


**FIGURE 2.15.1** *The total irradiance at a point is a function of all incident light vectors, and the normal at that point.*

Because this integral is expensive to compute in real time, what we can do is express $E$ and $L$ in terms of Spherical Harmonics. Spherical Harmonics are a set of orthonormal basis functions over the sphere.

Spherical Harmonics are a mathematical tool used to approximate an arbitrary spherical representation by decomposing it into a set of finite elements, similar to the way Taylor series operate. For more information about the full formula of Spherical Harmonics please refer to [Sloan03] and [Green03].

Coming back to Equation 2.15.2, we can express (project) the $E$ and $L$ in terms of Spherical Harmonics:

$$\vec{L} = \sum_{lm} L_{lm} Y_{lr} \tag{2.15.3}$$

$$\vec{E} = \sum_{lm} E_{lm} Y_{lm} \tag{2.15.4}$$

where l and m define the degree of the Spherical Harmonic.

The full representation of Spherical Harmonics (as detailed in Equations 2.15.6 and 2.15.8 later on in this article) includes an infinite number of coefficients $Y_{lm}$. The first few of them are exemplified in Equation 2.15.5 (in polynomial form):

$$\begin{aligned}
Y_{0,0} &= c_1 \\
Y_{1,-1} &= c_2 y \\
Y_{1,0} &= c_2 z \\
Y_{1,1} &= c_2 x \\
Y_{2,-2} &= c_3 xy \\
Y_{2,-1} &= c_3 yz \\
Y_{2,1} &= c_3 xz \\
Y_{2,0} &= c_4 \left(3z^2 - 1\right. \\
Y_{2,2} &= c_5 \left(x^2 - y^2\right.
\end{aligned} \tag{2.15.5}$$

where x, y, and z are the 3D Cartesian coordinates of the incident vector.

As can be seen, those coefficients are a function of direction. The constants c1, c2, etc. are derived from the convolution properties of Spherical Harmonics. The values of the constants are:

$$c_1 = \frac{1}{2\sqrt{\pi}} = 0.282095$$

$$c_2 = \frac{\sqrt{3}}{2\sqrt{\pi}} = 0.488603$$

$$c_3 = \frac{\sqrt{15}}{2\sqrt{\pi}} = 1.092548$$

$$c_4 = \frac{\sqrt{5}}{4\sqrt{\pi}} = 0.315392$$

$$c_5 = \frac{\sqrt{15}}{4\sqrt{\pi}} = 0.546274 \tag{2.15.6}$$

As [Ramamoorthi01] demonstrates we can now express $E$ as a function of $L$, but expressed in terms of Spherical Harmonics coefficients:

$$E(\theta,\phi) = \sum_{lm}^{\infty} AL_{lm} Y_{lm}(\theta,\phi) \tag{2.15.7}$$

where $\theta$ and $\phi$ are spherical coordinates, and $A$ is a term which is dependent on $n$ and $w$. The conversion from polar to Cartesian coordinates is obtained using the parameterization of the unit sphere:

$$(x,y,z) = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta) \tag{2.15.8}$$

where $x$, $y$, and $z$ are 3D Cartesian coordinates.

Based on the range of $l$ and $m$ we can control the error generated by the approximation. Obviously the more components in our sum, the better the approximation. In Equation 2.15.9, the term $\alpha$ represents the error term, and p and q represent the number of terms.

$$E(\theta,\phi) = \sum_{l,m}^{p,q} AL_{lm} Y_{lm}(\theta,\phi) + \alpha \tag{2.15.9}$$

To determine this light contribution to the lighting environment we only need to calculate a finite number of coefficients of the Spherical Harmonics function. [Ramamoorthi01] has shown that using only the first nine coefficients of the Spherical Harmonics is enough to produce high accuracy. [Ramamoorthi01] mentions that for any pixel the average error obtained is around 9% (as a fraction of the total intensity of the illumination). However, high-frequency lighting information is lost. Because of this mainly diffuse, low-frequency lighting can be modeled accurately by an SH representation containing a limited number of terms.

The coefficients are mathematically symbolized by the term $Y_{lm}$ where

$$l \geq 0$$

and

$$-l \leq m \leq l$$

$l$ defines the degree of the terms and to obtain the needed nine terms we need $l = 0$, 1, and 2 that gives us constant terms, linear terms and quadratic terms. Those coefficients are listed in Equation 2.15.5.

This is where the game code breaks in two. First we need to encode the lighting environment in Spherical Harmonics terms. In a game this usually represents all the light sources in a scene, plus any environment lighting (sky, distant Sun, etc.). For static lighting environments we can do all the calculations offline. Note that "static" does not mean the lighting environment cannot be rotated: one property of SH is that they are rotationally invariant; thus models can still be lit properly by aligning them with the original "placement" of the light environment using the inverse of the transformation matrix that was used to rotate it.

However, for dynamic light sources situations (e.g., sky going from bright to overcast) we need to do this encoding in real time. This actually means calculating the $L_{lm}$ terms.

The second part is calculating the actual irradiance, $E$. This depends on $\theta$ and $\phi$ (spherical coordinates), as well as the surface normal. Therefore this needs to be done at the geometry rendering stage when either the vertex or fragment normal is available.

## The Two Methods

There are two ways we can use Spherical Harmonics Irradiance-based lighting and which one we choose depends on what we intend to do (or the type of data produced by artists).

If our lighting data contains only analytical lights (spot light, point light, etc.) defined from a set number of properties then we can jump straight from that data to SH coefficients.

On the other hand, lighting data may contain arbitrary configurations of light emitters, which is a possibility in applications that know the details of the light sources (for example: how big the bulb, how long and thick the neon light). Also some applications may use pre-rendered images representing a given lighting environment (an overcast sky as mentioned earlier). In those cases, SH coefficients will have to be calculated by integrating across the environment maps involved.

This part of the article explains both methods.

### Analytical Lights

This is the simplest and most straightforward way of using Spherical Harmonics Irradiance lighting. We presume all lights' position to be at infinity. However, we are still looking at the lighting environment as a whole. Consider the example model shown in Figure 2.15.2 with two lights.

We build a sphere of radius 1 on which we project each light. This way the points on the sphere are actually the normalized light directions.

If we consider the sphere as our environment lighting then in the case of Figure 2.15.1 the only light contribution comes from the projection of the two lights on the sphere surface. Hence doing an integral is not actually needed. So instead of summing over the whole map we just add the lights positions that we know about. In this case the $x$, $y$, and $z$ terms in the Equation 2.15.5 are the normalized light direction.

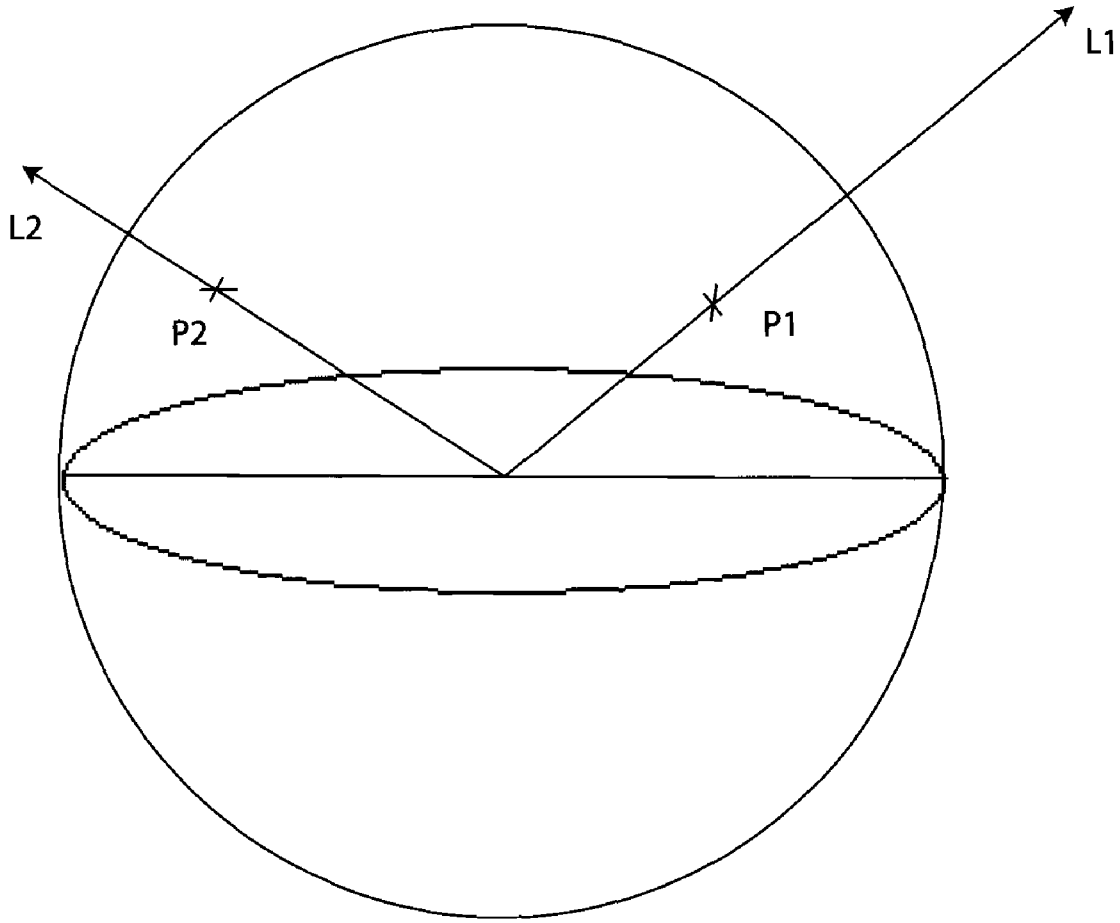


**FIGURE 2.15.2** *Two lights' projection on the unit sphere.*

Since Spherical Harmonics are basis functions over a sphere we can add several lights together by adding their Spherical Harmonics coefficients. This is possible since light is additive both in spatial domain and in frequency domain. This property is very useful when calculating coefficients for multiple lights: just project each light and then add the coefficients, thus making it easy for an application to pick at run time which light sources to project onto SH and which not.

### Surface Lights

Surface light sources are useful for games because they can be used to represent real light environments more accurately (e.g., sky, pre-rendered environment maps or even real-time rendered HDR or non-HDR maps). However, they are not frequently used in real-time applications due to the complexity of their implementation. This is where Spherical Harmonics can come in handy. At the beginning of this article we mentioned that Equation 2.15.2 represented the incident light at any point covered by a given lighting environment. After projecting Equation 2.15.2 onto SH we ended up with Equation 2.15.6. Because we are dealing with discrete sets of data we can effectively replace the integral in Equation 2.15.6 with a sum. What Equation 2.15.6

actually represents is a sum of all the points in the environment map ($L_{lm}$) weighted by the Spherical Harmonics coefficients ($Y_{lm}$).

Since we have decided to calculate up to nine coefficients, we need to calculate nine sums, one for every $Y_{lm}$ with:

$$l = (0, 1, 2) \text{ and } m = (-2, -1, 0, 1, 2)$$

The nine sums will represent (to a certain degree of approximation) the whole of the environment map.

What we have done is compress the whole of the environment map (which can be a large texture or a set of textures) to nine single values. "Compress" is the word because the technique has similarities to the way DCT compression based algorithms work (e.g., JPEG): by projecting the data into frequency space and then applying a filter on this data before projecting it back into the original domain. Here we are using Spherical Harmonics transforms to perform the projection to frequency space.

When rendering we will use these nine values together with the surface normals to compute the radiance (the amount of light leaving each point). To do this we plug in the normals into Spherical Harmonics Equation 2.15.6. This operation is necessary since we are decoding back the information from frequency domain to spatial (image) domain. This can be compared to a cube map look up into lighting data encoded as Spherical Harmonics.

#### Per Frame or Offline Calculation

When doing all those calculations to "compress" the environment map we have two choices: either we do it offline or at runtime, in a per-frame fashion. If we intend to use this technique as a light reduction technique then we can use it per frame to reduce the number of lights for which we compute full light calculations. This way we can do a runtime decision of which lights to keep (the "important" lights) and which lights to project onto Spherical Harmonics.

Offline calculations can be done in cases when we have a pre-rendered environment map (Sun and sky, ground, or more complex lighting environments which will be hard to express using analytical lighting techniques).

## Determining the Lights' Contribution to the Scene

This step is important because it determines both the look and the speed of our algorithm. What is explained here is not totally related to Spherical Harmonics and can be used anytime when we try to coalesce our lights. The main point of this section is to explain how to determine which lights will have the full lighting calculations applied to them and which lights will benefit from SH lighting calculations instead.

There are many methods to do this and the one to choose depends on each particular application. In the demo accompanying the article we use the following algorithm.

There are three important factors: the intensity of the light, the chromacity of the light, and the distance of the light to our geometry. We have to sort our lights by a

function of the three factors above. What function to use is usually a case of trial and error.

Chromacity represents the bias in color of a certain light. A light with high chromacity will be further away from grey. For example if we have three lights with an RGB of (100, 100, 100), (125, 125, 125) and (255, 0, 0) and we eliminate the last one our scene will turn from "redish" to grey which is wrong. The red light is the light with the highest chromacity among the lights above.

How do we quantify all this into an algorithm? We compute three values:

$$D_{RG} = |R - G|$$

$$D_{GB} = |G - B|$$

$$D_{RB} = |R - B| \tag{2.15.10}$$

The chromacity factor is:

$$C_f = \max(D_{RG}, D_{GB}, D_{RE} \tag{2.15.11}$$

The intensity factor is:

$$I_f = R + G + B \tag{2.15.12}$$

The light factor will be:

$$L_f = c_1 C_f + c_2 I_f \tag{2.15.13}$$


where $c_1$ and $c_2$ are arbitrary percentage/weights factors. In the demo application provided on the accompanying CD-ROM the following two values are used:

$$c_1 = 0.7$$

$$c_2 = 0.3$$

We now calculate an $L_f$ for every light and then sort all the lights according to $L_f$. For the top number of lights we compute the full Phong lighting equation—the rest are dealt with using SH.

## Putting It All Together—Code Explanation

Spherical Harmonics are used to assist us in lighting calculations and they require work to be done both on the host (CPU) and on the graphic device (GPU). Where the partition of the work lies depends mainly on the capabilities of the GPU. For each of the cases discussed, the next section describes the tasks that need to be carried out for both the CPU and GPU.

### Surface Lights

The algorithm is the same regardless of how the environment map is obtained. Environment maps usually represent 360° of data and there are different ways to store them. Let's take the example of environment maps stored in 2D textures. To compute the Spherical Harmonics' coefficients we need to sum all the pixels in the texture (which represents lighting information) and weigh them by the coefficients $Y_{lm}$. The algorithm for doing this is:

```
For every l and m (degree of the SH)
    For every x,y of pixels in the texture
        Calculate theta and phi, spherical coordinates out of 2D
          coordinates
        Calculate x,y,z, 3D coordinates out of spherical
          coordinates
        /* x, y, and z now represent a direction in the unit
           sphere which corresponds to the x and y coordinates of
           the pixel in the texture */
        Depending on l and m compute Ylm following the formula in
          Equation (5)
        integral += Sample(x, y, EnvMap) * Ylm;
```

Because there are nine combinations of l and m, we will end up with nine `integral` values. Those are our nine coefficients.

However, our lighting environment might not always come from a spherical environment projected onto a plane but also from a cube map. In this case the calculations required to obtain the Spherical Harmonics coefficients are a bit different. The two diagrams shown in Figure 2.15.3 show the steps necessary when having either plane or cube map representations.

### Analytical Lights

The code to turn analytical lights into SH is shown here:

```
    for(i=0; i<ui32NumberOfLights; i++)
    {
        sSHCoeff[0] += Light.color * fConst1;
        sSHCoeff[1] += Light.color * fConst2 * Light.x;
        sSHCoeff[2] += Light.color * fConst2 * Light.y;
        sSHCoeff[3] += Light.color * fConst2 * Light.z;
        sSHCoeff[4] += Light.color * fConst3 * (Light.x * Light.z);
        sSHCoeff[5] += Light.color * fConst3 * (Light.z * Light.y);
        sSHCoeff[6] += Light.color * fConst3 * (Light.y * Light.x);
        sSHCoeff[7] += Light.color * fConst4 * (3.0f * Light.z *
Light.z -  1.0f);
        sSHCoeff[8] += Light.color * fConst5 * (Light.x * Light.x -
Light.y * Light.y);
    }
```

Depending on which level we want to calculate the actual lighting (vertex or fragment) the same code can be used for both.

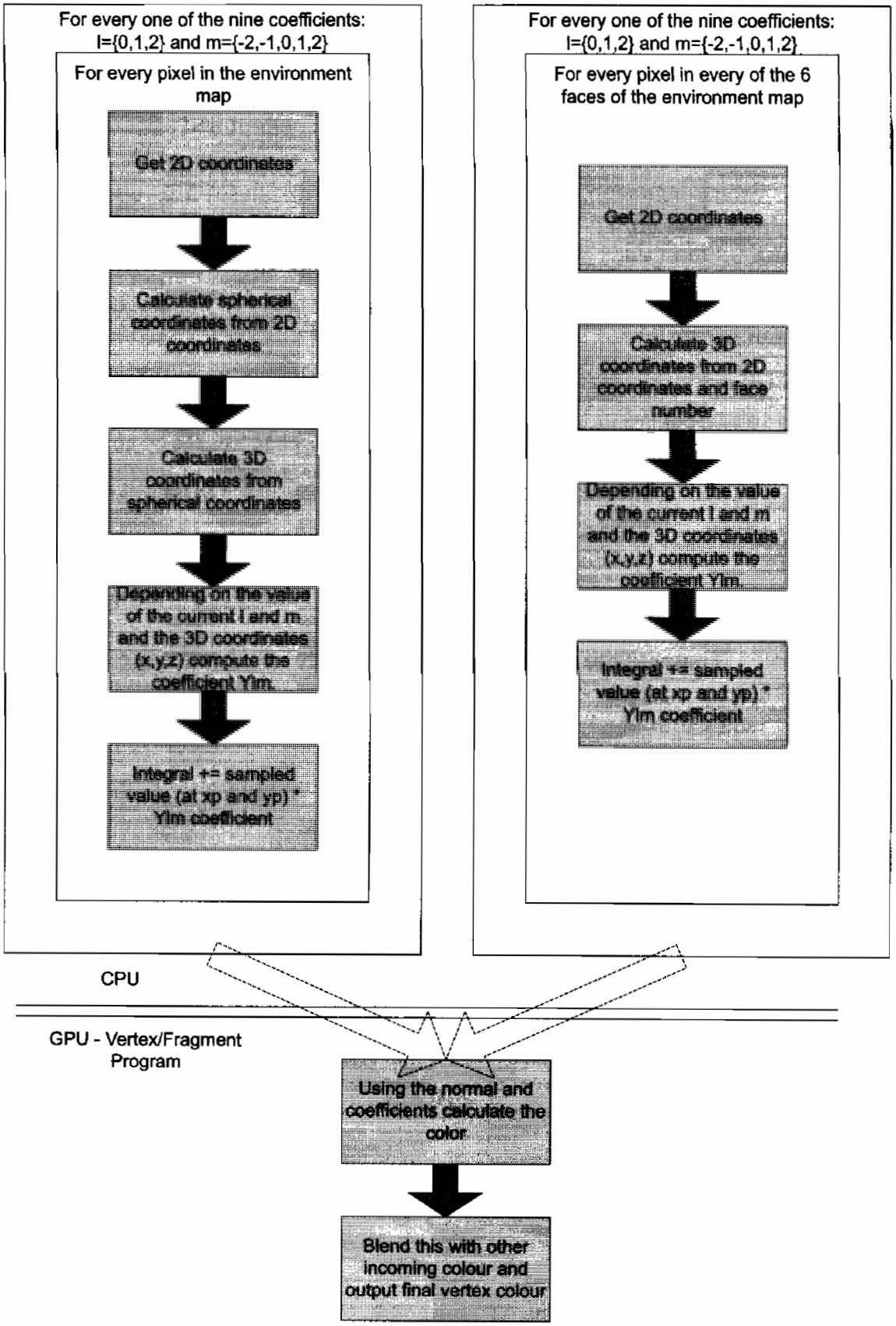


**FIGURE 2.15.3** *SH coefficient calculations from both plane and cube map environment maps.*

#### Vertex Program Work

- Advantages: faster than fragment program solution
- Disadvantages: image quality poor in low tessellated geometry

#### Fragment Program Work

- Advantages: looks good, giving us diffuse lighting per pixel
- Disadvantages: quite slow since there is a big program that needs to be executed for every fragment

## Pros and Cons of Spherical Harmonics

There is no such thing as a perfect technique. This part of the article sums up the advantages and disadvantages of this specific one.

#### Pros

- When using analytical light sources it does not matter how many light sources we add since the SH lighting can be done in one pass; therefore, rendering performance will not be affected.
- In the case of analytical light sources the algorithm is simple enough to allow calculations to be performed at runtime.
- Possibility of expressing a whole lighting environment using only nine values.

#### Cons

- Some lighting effects cannot be modeled using Spherical Harmonics (e.g., shadows, radiance transfer, Phong light effects, etc.). There are, however, techniques (not described in this article) which extend the basic Spherical Harmonics algorithm to achieve more complex effects. One of these techniques is called *Precomputed Radiance Transfer* (or PRT) [Green03], which requires more data besides lighting information to be available (visibility information for example).
- When using arbitrary light environments, the process of projecting onto Spherical Harmonics is quite costly (an integration across the whole environment map is needed). Therefore, it's not always doable at runtime.

## CD-ROM Demo


The demo included in the companion CD-ROM provides an example implementation of the Spherical Harmonics lighting calculations. The demo requires hardware support for vertex and fragment program, accessible through the OpenGL API. For the host part of the calculations the relevant files are `sh_env.cpp` and `sh_analytical.cpp`. The part of the Spherical Harmonics algorithm that needs to be done on the GPU has been implemented using both OpenGL Shading Language and the assembly shading language extensions. Have a look at `sh_lighting.vs` for an implementation using the former and `sh_lighting.vp` using the latter.

## Conclusion

This article describes the basics of Spherical Harmonics and how they can be used to represent a lighting environment either composed of analytical light sources or arbitrary lighting data. Techniques for compressing a given lighting environment to just nine RGB values have been shown. Two gamelike scenarios have been presented: one using analytical lights and one using surface (environment map) type lights. This article is by no means a comprehensive Spherical Harmonics theory but it introduces the reader to their world and opens the road to more advanced techniques, e.g., Precomputed Radiance Transfer or visibility/depth compressing methods.

## References

[Forsyth03] Forsyth, Tom, "Spherical Harmonics in actual games," available online at: *http://www.tomforsyth.pwp.blueyonder.co.uk/papers.html.*

[Green03] Green, Robin, "Spherical Harmonics Lighting: The Gritty Details," January 16, 2003.

[Ramamoorthi01] Ramamoorthi, Ravi et al., "An Efficient Representation for Irradiance Environment Maps," [SIGGRAPH 2001]. Code to show technique used in the paper can be found online at: *http://graphics.stanford.edu/papers/envmap/prefilter.c.*

[Sloan03] Pike-Sloan, Peter, "Efficient Evaluation of Irradiance Environment Maps," *Shader X²—Shader Programming Tips and Tricks,* Wordware Publishing, 2003.