Lighting using Spherical Harmonics

Mayuran Thurairatnam

Department of Computer Science at Hood College

Introduction

I will try to make one model for using spherical harmonics by combing three other methods and tailoring them to my needs for speed and ease of installment. Using these methods I can combine them to shadow a scene with soft shadows and cinematic like quality.

The three methods I am going to converge are as follows:

- Precomputed Radiance Transfer[2]
- Irradiance Volume[4]
- Shadow Blockers[5]

Also combining these methods I can get rid of any redundancy and make one easy to use model to shadow all types of objects.

Motivation

One of the main differences between cinematic rendering and real-time graphics used in games is the way they handle light and shadows.

Soft shadows are something that can be difficult to calculate in real-time, since one the best methods is ray casting.

Using spherical harmonics and the ever growing power of Graphic Processing Units (GPU) in video cards, I can try to bridge this gap.

Also, I would like to try to make the solution as easy as possible to integrate into a game engine.

Previous Work

Spherical harmonics

Spherical Harmonics (SH) are represented by a set of basis functions that I can use to approximate the amount of light around an object. If I assume a point is a sphere I can integrate over the sphere and make a SH vector that represents the lighting.

How to build

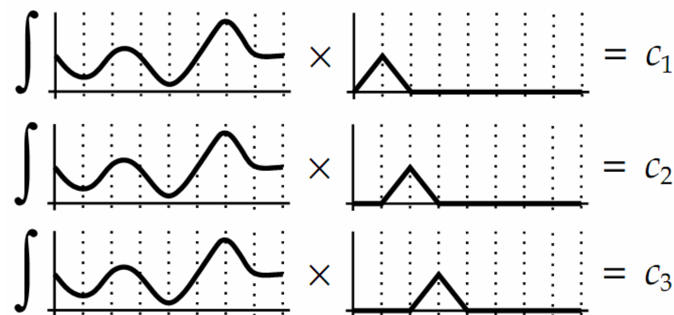

Figure 1 Spherical Harmonics basis function [image from [1]]

I can build these SH vectors using the Monte Carlo Integration [1]. This takes random samples over a sphere while taking the product with the SH basis functions and weighing them based on the number of samples.

Product

The product over two SH vectors is the intersection of the two [1]. This can be used to account for visibility; one vector can represent lighting from outside and one can be the shadowing underneath. The product will be another SH vector that can be used for future calculations.

$$\mathrm{M_{ij}} = \int_S L(s) y(s)_i y(s)_j \, ds$$

eq1 from [1]

One can make a Matrix $\mathrm{M_{ij}}$ by integrating over the original function L and the two other spherical harmonics. This

product matrix can then be multiplied by any SH vector to get a new vector that is the product of the two.

This is an expensive operation, and the triple product method will be the one I use for this solution.

Visibility by Integration

To calculate the amount of light reaching a point one can easily take the dot product of the two vectors and it will return a visibility value. This will be like integrating over the SH vector as follow [1]:

$$\int \tilde{\mathrm{L}}(\mathrm{s})\tilde{\mathrm{A}}(\mathrm{s}) = \sum_{i=0}^{n^2} L_i\, A_i$$

eq2 from [1]

L is the Light function and A is the blocker function. This is very quick, but it will give us one value that represents the visibility. Although, it will not be a SH vector, so it can only be used on the final visibility function.

To get another vector as a product one can use the triple product method [1]

$$\mathrm{M_{ijk}} = \int_S y(s)_i y(s)_j y(s)_k\, ds$$

eq3 from [1]

First, create a three dimensional matrix $\mathrm{M_{ijk}}$ that can be used along with the two SH vectors to be multiplied and will yield a product vector. This is fast enough that I can use it in the GPU, but only a few times. The product matrix takes up many of shader constants so low order SH vectors are recommended.

Rotating SH vectors can become very complicated. The previous way of SH vector rotation is to build a matrix for rotation in the Z and X90 directions. Using this method one can rotate via ZYZ rotations [1]. But this becomes slow. There is quicker method that can be implemented to rotate the vectors in the GPU [5].

This faster method involves storing the SH vectors for visibility spheres of increasing radius. This data takes the form of a 1D table that stores just a fraction of the total values, since one can approximate the vector using only the non-zero coefficients. Also, one must use a 2D table of the basis SH functions in the desired direction of rotation [5].

Precomputed Radiance Transfer

Precomputed Radiance Transfer (PRT) is a method of baking SH representations of light into a vertex or a texture. A PRT simulator would find the lighting calculations [2]. This is the robust, flexible method capable of taking into account high orders and inter-reflectance.

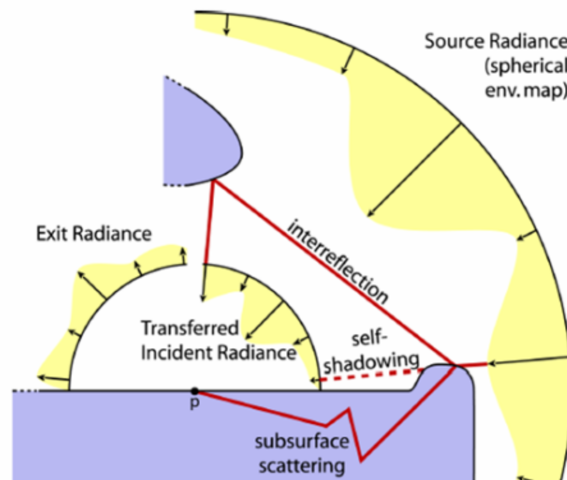

Figure 2 PRT Lighting for a point P [image from [DirectX SDK]]

PRT iterates over each vertex and uses the Monte Carlo integration to build the coefficients. These values are stored in each vertex for later use. The only con is these SH vectors only account for what objects the rays hit during simulation. Thus, the object must be static. Since the object is static, one can calculate inter-reflectance of light and subsurface light transfer.

Irradiance Volumes

When PRT is used, there is a need for shadowing objects that travels through these PRT static objects. These are known as Irradiance Volumes. These volumes would store at each point the lambertian cosine visibility [3]. These values would form a 5D function that one would index and then send the SH vector to the GPU.

The volume does not have to be at a high density, one uses central differencing or linear interpolation to find the SH value at any point.

FIGURE 3 IRRADIANCE VOLUME THE SPHERES REPRESENT THE AMOUNT OF LIGHT AT A POINT [IMAGE FROM GREGER98]

SPHERICAL BLOCKERS USING LOG SPACE

This technique is used for shadowing dynamic blockers [5]. There are a few steps in this method. First, one must approximate a dynamic blocker with spheres. Then, one accumulates the visibility for a point buy adding all the log blockers together.

SPHERE APPROXIMATION

Sphere approximation is critical for correctly representing a dynamic blocker. One would want to have the minimal number of spheres possible, while still representing the mesh accurately.

Normally, using methods such as an octree will yield many spheres. The method described in Wang et.al. first scatters spheres across a mesh's vertices and volume [6]. They developed a method of calculating the error that a sphere approximation yields. They use this method to adjust from the initial sphere positions to what position yields the lowest error.

Wang et.al. use a method of clustering to determine the radius and position of the spheres. However, one must only add a point to a sphere that yields the lowest error increase, instead of just adding it to the closest sphere.

Another method for calculating sphere position and radius is sphere teleportation. This technique is utilized when an error increase arises after a pass, in order to keep from staying in the local minimum. When using sphere teleportation, one must first locate the sphere that has the highest error. In addition, one must locate the sphere that has the highest overlapping volume with other spheres. Finally, one must delete that sphere and split the sphere with the highest error into two new spheres.

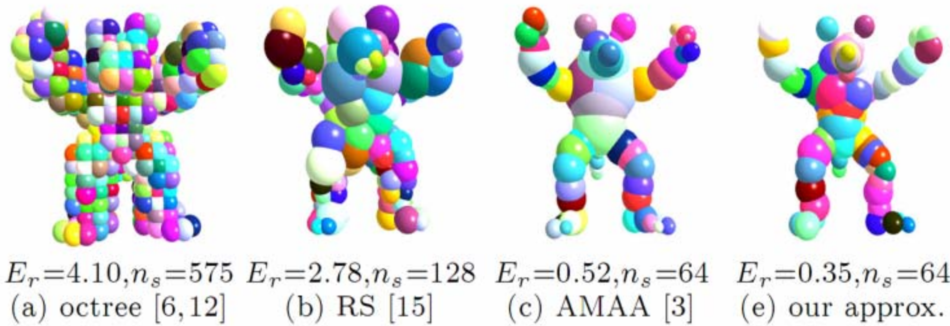

FIGURE 4 SPHERE APPROXIMATION FROM [6] SHOWS HOW MUCH BETTER IT IS THEN OCTREES OR MEDIAL AXIS METHODS. [IMAGE FROM [6]]

These spheres are then attached to the mesh's animation bones.

LOG SPHERES

The reason that spheres are used in these calculations is that they are rotationally invariant. The visibility spheres show 1 when blocked and 0 when not.

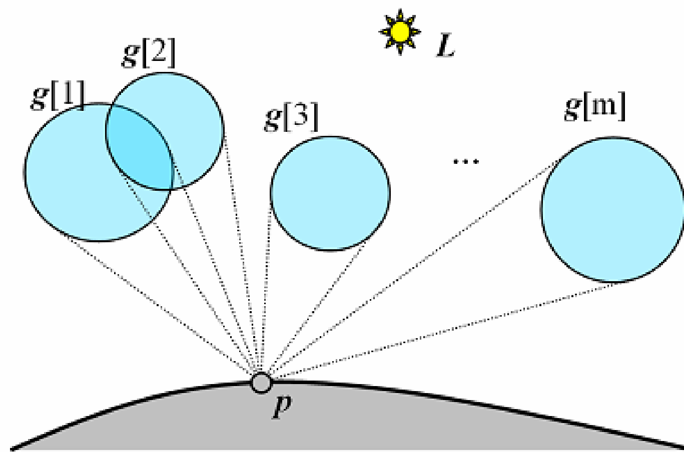

FIGURE 4 SHOWS HOW MANY SPHERE BLOCKERS CAN SHADOW A POINT P [IMAGE FROM [5]]

One can now iterate over all the spheres and calculate the visibility for a point P. The problem is that taking the product of two SH vectors is expensive, and taking the product of many SH vectors would not be feasible. Ren et.al. present a method that transforms the vectors into log space. Then one can just add the vectors together. This method speeds up blocker accumulation considerably. After the additions are complete, one merely has to take the exponential function of the vector, thus calculating the total blocker visibility vector.

In the following equation, the vectors are being built on the log function of the spheres, which yield artifacts.

$$\mathrm{f} = \log(g)$$

Eq4 from [5]

The solution proposed by Ren et.al. is to invert their SH Exp method by taking the log of the diagonalized version of the SH Product matrix like so:

$$\mathrm{g} = \log(g) = R_g^T q'(D_g) R_g (g - 1)$$

Eq5 from [5]

$$q'(x) = \frac{1}{q(\log(x))} = \log(x)/(x-1)$$

Eq6 from [6]

$$\tilde{D}_g = \max(D_g, \varepsilon), \tilde{M}_g = R_g^T \tilde{D}_g R_g$$

Eq7 from [7]

Using eigenanalysis on the SH product matrix ($\tilde{M}_g$) one calculates rotation matrices R_g^T and R_g and applies eq6 to (D_g). The max in eq7 is to avoid applying log to any numbers negative to close or 0 (ε).

Eigenanalysis cannot be calculated on the fly, so one should precalulate these values and store them in a texture along with any other functions that may be needed. One should store the precalculated log SH vector of a sphere of increasing radius pointing in the Z direction and then rotate in the shader.

Solution

Spherical Harmonic Use in Lighting

The table here shows what methods I'll use to shadow the wide variety of object common to games. For a static mesh it will use PRT for self-shadowing and Log blockers for other static meshes. This means that all meshes will have to be sphere approximated.

Casting/Receive a Shadow	Casting Static Mesh	Casting Static Moving Mesh	Casting Dynamic Mesh
Receiving Static Mesh (A room)	PRT	Log Blocker	Log Blocker
Receiving Static Moving Mesh (A car)	IR Volume	PRT from self Log from others	Log Blocker
Receiving Dynamic Mesh (A walking person)	IR Volume	Log Blocker	Log Blocker

TABLE1. HOW I WILL USE THE 3 DIFFERENT METHODS TO CAST SHADOWS ON ALL 3 DIFFERENT OBJECTS

Although by the looks of the table it seems I would need the log blockers for most shadows, they will only shadow a small part of the scene; most of the scene should be shadowed by PRT and irradiance volumes. The slowest solution should be shadowing a static mesh; it needs a shader that would use all 3 methods, while others meshes only need 2.

The Model

The problem is these methods require different types of information. Some information can be factored, while other information is unique to the method and cannot be used with the others.

Log Blockers are the lowest denominator; they need the lowest SH order, and only show shadowing information, no light transfer or inter-reflectance.

One could use high order SH for PRT and Irradiance Volumes, but this would provide a noticeable difference in shadows. The user would be able to tell when a dynamic object is shadowing, just like previous methods with coarse shadow buffers and shadow maps.

The solution is account for visibility only, as Log Blockers do, without the lambertian cosine accounted for.

PRT

PRT has the most power of the three methods, subsurface scattering, all three color channels, etc. Now the more of these I include, the more contrast there will be between static and dynamic objects. However, I do not want the user to be able to tell what is static and what is dynamic.

Instead of doing any light transfer functions for PRT I just calculate a visibility of a point and bake that into the vertex.

Irradiance Volume

Irradiance Volumes are a 5d function, they provide irradiance function for each direction. Since irradiance includes lambertian cosine, this makes the visibility function easy to computer in the shader. However, this solution has a few SH products already. Therefore, I can just contain the visibility of the point in the volume, without the irradiance. This saves a considerable amount of space, instead of a large 5d function I need only one SH vector. Now some may want the extra performance gain from missing the SH product; however, if I use this method the volume size can be much bigger, while the data remains small enough to fit into a volume texture. This texture can be used as a vertex lookup in the future to provide very high detail visibility volume.

Log Blockers

To customize log blockers for our solution I will be trying to make it easy to integrate into a game engine. Ren et.al. have an excellent implementation using clusters and ratio vectors to expand the range of the amount of blocker spheres that can be represented in a scene. They use the GPU to process all of the vertices in a scene in clusters and account for the visibility of what sphere sets to use. This may scare some developers away from using the solution since it would be vertex dependent. The scene would need to be highly tessellated and the texture would have to be adjusted as vertices in the scene increase and decrease.

I present a solution where one uses the pixel shader to calculate the blocker visibility directly. This puts all the work on

the pixel shader, and could be detrimental if there is already a large workload on the pixel shader for other effects. However, my solution retains a few advantages:

- Not vertex dependent
- Higher detail without high tessellation
- Easy to implement
- No sphere set hierarchies
- Steady speed

Even though I will have a slower speed, and can approximate less sphere blockers, if implemented correctly I can keep the speed consistent over a scene. This is accomplished by doing a Z-only pass, and then when the color write pass is done only the pixels that will be shaded are executed. This means that no matter how many vertices are displayed on the scene that require log blockers, the speed would remain the same since the pixel shader is performing the majority of the calculations. Currently I am limited to 16 sphere blockers per a pixel to perform at an acceptable frame rate (30 Hz). Therefore, a better sphere culling algorithm would provide better results.

Originally, I was going to use the vertex shader to cull spheres, and send the indices of the spheres to the pixel shader. However, the problem is that the vertex shader cannot send integers, only the interpolated floating point numbers from the three surrounding vertices. I hope that future API's (DirectX 10) will provide a solution for this.

Performance Optimization

There are a few performance optimizations that I can implement. First, I could use 4 coefficients to represents SH vectors for all methods. If one wants to provide more detail, you can go into higher orders for other methods, but the best solution for log blockers is 4.

I use the optimal linear approximation method for log blockers [5]. This technique serves to help reduce the amount of SH products I need.

I also use the faster rotation method [5]; however, instead of using polar coordinates and converting in the GPU, my

technique uses a cubemap and the input would be the desired direction of the SH vector.

The Z-only pass might not help significantly for our demo; however it will make a substantial improvement in a game where there are many objects.

RESULTS

The machine that was used was a p4 3.0 GHz with an NVidia Geforce 6600 GT. Although the hardware is not powerful, it would be a great ruler for measuring the loosest end of the hardware that would use this technique in a game. A complex scene is one where I have a dynamic object walking around on a static object. This will use all three methods. Also a HDR shader was introduced to show how adding multiple shaders would affect speed of the method. A maximum of 16 spheres could be shadowed on a point.

The results stayed steady at 20Hz, fluctuating little. Although this slow, future hardware such as NVidia G80 or ATI RX600 GPU's have much more powerful pixel shaders and a scalable architecture.

FIGURE 5 DYNAMIC FIGURE WALKING INFRONT OF A STATIC BUILDING

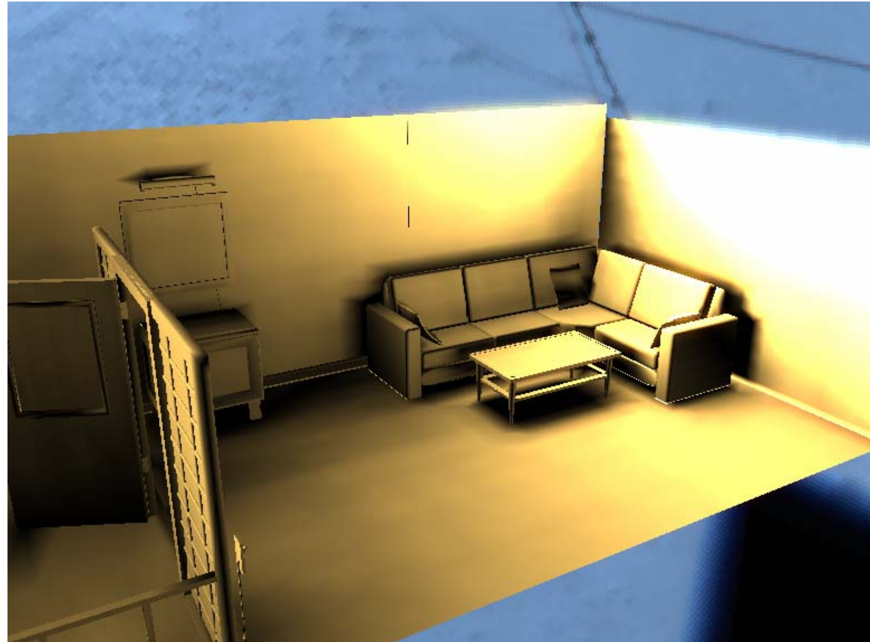

FIGURE 6 A WARM LOCAL LIGHT SOURCE LIGHTS UP THE ROOM FROM THE INSIDE

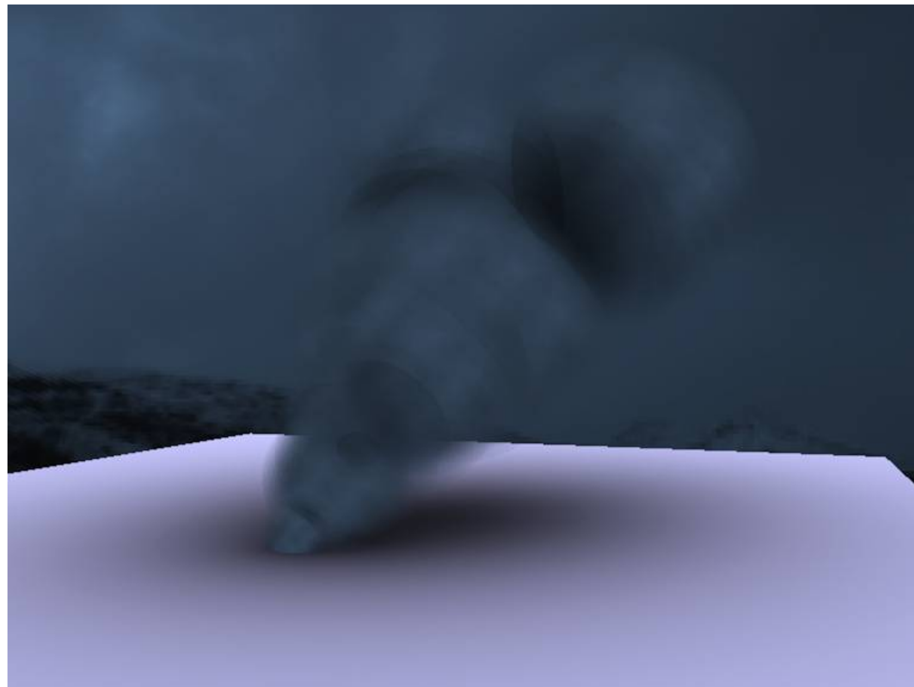

FIGURE 7 EVEN SMOKE CAN BE REPRESENTED WITH BLOCKERS

FUTURE WORK

One of the major things I wanted to do for future work was to cull spheres from the vertex shader, balancing the shading pipeline is key, and could provide an excellent result. This

concept is not possible right now on current API's and shaders.

In addition, trying to put an irradiance volume in a volume texture may provide better results for shadowing on objects. Using the vertex shader one could sample the volume texture; however, for large scenes this may prove impractical.

Higher order SH's would greatly improve quality, and provide nice, long, soft shadows. It would be interesting to see how far I can take these methods on future hardware.

Finally, I would like to find a way to dynamically cast translucent shadows. Although uses are few for this, it would provide great realism for things like a dragon's wings or other organic material.

References

1. *The Irradiance Volume.* **Greger, Gene, et al.**

2. **Green, Robin.** *Spherical Harmonic Lighting.* s.l. : Sony Computer Entertainment America, 2003.

3. *Precomputed radiance transfer: theory and practice.* **Kautz, Jan, Sloan, Peter-Pike and Lehtinen, Jaakko.** s.l. : ACM Press , 2005.

4. *Irradiance Volumes for Games.* **Tatarchuk, Natalya.** s.l. : ATI, 2005.

5. *Real-Time Soft Shadows in Dynamic Scenes using Spherehical Harmonic Exponentiation.* **Ren, Zhong, et al.** s.l. : ARC SIGGRAPH, 2005.

6. *Variational Sphere Se tApproximation for Solid Objects.* **Wang, Rui, et al.** s.l. : ACM SIGGRAPH.