Speeding up the radiosity rendering algorithm using the kernel-independent FMM
Ross Adelman

Introduction

For my project, I decided to apply the fast multiple method (FMM) to the radiosity rendering algorithm. The radiosity rendering algorithm is a technique for creating photorealistic images. Introduced in [1], it works by treating light as radiated power, and looking for an equilibrium in power transfer among diffuse surfaces. A more efficient implementation was introduced in [2], and the algorithm was extended to non-diffuse surfaces in [3]. For my project, I only considered diffuse surfaces.

Speeding up the radiosity rendering algorithm using a hierarchical approach like the FMM is nothing new. This concept was explored in [4], where the input triangle mesh was adaptively subdivided and organized into a quad-tree based on level of detail. The results in that paper were quite nice; however, it required tampering with the mesh. Applying the FMM, which uses a box-based data structure, to the radiosity rendering algorithm was explored in [5].

This paper explores using the kernel-independent FMM, described in [6], to speed up the radiosity rendering algorithm. Instead of using complicated series representations for the multipole and local expansions, equivalent density representations are used. For my project, I implemented many of the strategies described in that paper, and also modified them a little to work better with the radiosity kernel.

Background

My implementation takes as input a mesh of triangles, each defined by three vertices. Each triangle has an implicit center, area, and outward-facing normal, but these are calculated beforehand to speed up later calculations. In addition to these geometric properties, each triangle has a color, \( \rho \), and an emission value, \( E \). The emission value is how much power per unit area the triangle radiates by itself. Both of these values are given as input parameters. In addition, each triangle has a radiosity value, \( R \). The radiosity value is how much total power per unit area the triangle radiates, including what it radiates on its own plus any power that it reflects. The purpose of the radiosity rendering algorithm is to calculate each triangle's radiosity value. Once these values are known, the scene can then be rendered using OpenGL, where the each triangle’s radiosity value is used as its color. To solve for the radiosity values, the following is done.

\[
R_n = E_n + \rho_n \sum_{m=1}^{N} F_{nm} R_m
\]  

(1)

where
Useful for later on is the following expression, which calculates the radiance transfer from a source triangle to a receiver triangle.

\[ F_{nm} = \frac{1}{A_n} \int \int \frac{\cos(\theta_n) \cos(\theta_m) dA_m dA_n}{\pi r^2} \]  

(2)

Rearranging Eq. (1),

\[ \begin{bmatrix} 1 - \rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1 - \rho_2 F_{22} & \cdots & -\rho_2 F_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \cdots & 1 - \rho_N F_{NN} \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_N \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{bmatrix} \]  

(4)

\[ R = E \]  

(5)

This can be inverted using any number of techniques. However, since \( A \) is diagonally dominant, a simple iterative solver, like Jacobi iteration, can be used. The following iterative scheme is used to solve for \( R \).

\[ R_k = E + B R_{k-1} \]  

(6)

where

\[ B = \begin{bmatrix} \rho_1 F_{11} & \rho_1 F_{12} & \cdots & \rho_1 F_{1N} \\ \rho_2 F_{21} & \rho_2 F_{22} & \cdots & \rho_2 F_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_N F_{N1} & \rho_N F_{N2} & \cdots & \rho_N F_{NN} \end{bmatrix} \]  

(7)

and

\[ R_0 = E \]  

(8)

Thus, every iteration, Eq. (6) is used to calculate a new guess for the solution to (5). This is a \( O(N^2) \) operation, but can be speeded up using the FMM.

**Test Scenes**

Two test scenes were created. Scene A contains four colored spheres illuminated by a fifth (see Figure 1). Scene B is a variation of the Cornell box (see Figure 2).
Figure 1. Scene A.

Figure 2. Scene B.
Data Structures

Figure 3. Two sample renderings showing the box structure used. In these examples, $K = 7$, and only non-empty boxes are shown.

The pre-FMM and SLFMM start out the same way. A bounding cube is placed around all of the triangles in the mesh, and is divided into $K^3$ boxes, where $K$ is an input parameter. The triangles are sorted into these boxes, and a list of non-empty boxes is created. Figure 3 shows this in action for scenes A and B.

Calculating the S- and R-Expansions

Instead of using a series representation for the S- and R-expansions (e.g., using spherical harmonics), proxy surfaces are used. The idea behind proxy surfaces is that a smaller number of carefully-placed triangles can closely approximate the effect of a larger number of triangles. Each proxy surface is made up of proxy triangles. For an S-expansion, the proxy surface approximates the effect of the triangles in a box. For an R-expansion, the proxy surface approximates the effect of the triangles outside the neighborhood of the box.

To calculate the radiosity values of the proxy triangles, a check surface is used. Each check surface is made up of check triangles. The effect of the source triangles on the check triangles is calculated using Eq. (3). Then, the radiosity values of the proxy triangles are chosen so that their effect on the check surface matches (or comes as close as possible to) that of the source triangles. To do this, an over-determined set of linear equations is constructed and solved.

$$Ax = b$$

(9)

A is an $M$-by-$P$ matrix, $x$ is an $P$-by-3 vector, and $b$ is an $M$-by-3 vector, where $M$ is the number of check triangles and $P$ is the number of proxy triangles. Each column of $x$ and $b$
correspond to a color: red, green, and blue. The values of A represent the radiance transfer from the proxy triangles to the check triangles, and are calculated using Eq. (3). The values of b are the radiosity values of the check triangles due to the source triangles that the proxy triangles will approximate. The values of x, which is being solved for, are the radiosity values of the proxy triangles that yield the same (or as close as possible to the same) radiosity values on the check surface as the source triangles.

**S-Expansion Proxy and Check Surfaces**

![S-expansion proxy surface](image)

Figure 4. S-expansion proxy surface.

![S-expansion check surface](image)

Figure 5. S-expansion check surface.
There are two approaches for creating an S-expansion proxy surface for a box. The first approach is to construct an icosahedron that fits tightly around the triangles in the box. Another platonic solid, such as an octahedron, can be used. Also, for greater accuracy, the triangles can be subdivided. These triangles are used as the proxy triangles.

The second approach is to choose a subset of the source triangles to be the proxy surface (see Figure 4). The number of triangles to choose for the proxy surface is provided as an input parameter, but is usually around 80. The triangles are chosen so that there is a triangle pointing in every direction. Currently, my implementation uses this second approach.

There are two approaches for creating the corresponding S-expansion check surface. The first approach is to construct an icosahedron centered at the center of the box with an inner radius equal to three times the outer radius of the box. The resulting triangles are subdivided until there are roughly four times as many triangles as there are proxy triangles, usually around 320. These triangles are used as the check triangles (see Figure 5(a)).

The second approach begins by constructing an icosahedron centered at the center of the box with an inner radius equal to three times the outer radius of the box. The resulting triangles are subdivided until there are roughly four times as many triangles as there are proxy triangles, usually around 320. Then, each triangle is replaced with three triangles, which point in the X, Y, and Z directions. These triangles are used as the check triangles (see Figure 5(b)). The idea behind this approach is that the receiver triangles outside the neighborhood of the box could be facing in a number of directions, so having check triangles that face in three different directions helps lead to better radiosity values for the proxy triangles. My implementation uses this approach.

**R-Expansion Proxy and Check Surfaces**

To create an R-expansion proxy surface for a box, an icosahedron centered at the center of the box with an inner radius equal to three times the outer radius of the box is constructed. Depending on the accuracy required, the triangles can be subdivided. The total number of triangles is usually around 80. These triangles are used as the proxy triangles (see Figure 6).
There are two approaches for creating the corresponding R-expansion check surface. The first approach is to construct an icosahedron that fits tightly around the triangles in the box. The resulting triangles are subdivided until there are roughly four times as many check triangles as there are proxy triangles, usually around 320. The resulting collection of triangles is used as the check surface.

The second approach is to choose a subset of triangles in the box to be the check surface (see Figure 7). The number of triangles to use is provided as an input parameter, but generally, the number is around four times the number of proxy triangles, usually around 320. Currently, my implementation uses this approach.
Interesting Artifacts

There are a number of interesting artifacts that can occur depending on which approach for constructing the S- and R-expansion proxy and check surfaces. For example, using the first approach for constructing the R-expansion check surface can lead to the artifacts seen in Figure 8(a). This happens because the check triangles near the edge of the box point in a completely different direction than the actual triangles in the box at the same location. As another example, using the first approach for constructing the S-expansion proxy surface can lead to the artifacts seen in Figure 8(b). It is difficult to see, but there is a visible line around the edge of the rectangular plane. This is because some of the proxy triangles of the boxes containing the triangles that make up the rectangular plane point towards other triangles on the rectangular plane. In effect, the rectangular plane is illuminating itself. This is physically incorrect, and should not happen. By choosing the proxy surface to be a subset of the actual triangles, this artifact disappears.

Results

Figures 9 and 10 show the timing results and error analysis for scenes A and B. Both scenes produced similar timing results. The direct method has the usual $O(N^2)$ trajectory. The pre-FMM method using either S- or R-expansions is a little better, appearing to have an $O(N^{3/2})$ trajectory, which is expected. The SLFMM method is even better, coming close to an $O(N^{4/3})$ trajectory, which is also expected. Three iterations were performed for every $N$.

The absolute maximum error for each scene is usually low for most values of $N$. The pre-FMM using S-expansions has the lowest error of the three. The SLFMM method has the highest error of the three. Since each color channel has eight bits of dynamic range, only error greater than 1.0 / 255.0, which is roughly $3 \times 10^{-3}$, is visible. For scene A, the pre-
FMM using S-expansions stays below this threshold, but for scene B, all three methods have trouble satisfying this error bound.

An alternative way to look at the error is by plotting the mean absolute error. Figure 11 shows this for scene B. This plot indicates that for most of the triangles in the mesh, the calculated radiosity values are very good. For a small fraction of the triangles, though, the radiosity values are wrong. This likely stems from artifacts similar to those described earlier.

Figure 9. Timing results and error results for the scene A.
Future Work

Still left to add is the MLFMM method. To do this, many of the data structures need to be modified to allow for levels. In addition, I did not account for visibility. The test scenes I used did not require this, so this did not cause any problems. However, most scenes will look odd unless visibility is considered. Some possible ways to do this would be to have bounding spheres for each box, and then instead of testing whether a ray hits every triangle in a box, just test whether the ray hits the bounding sphere. Adding this to the pre-FMM
and SLFMM methods would be fairly straight forward, but adding it to the MLFMM method might be difficult due to the multi-level structure.

Figure 11. Mean absolute error for scene B.

Conclusion

For my project, I applied the FMM to the radiosity rendering algorithm. I implemented the pre-FMM using both S- and R-expansions as well as the SLFMM method. I achieved decent speed-ups for all three methods. Instead of using a series representation for the S- and R-expansions, I employed the kernel-independent FMM. This approach worked well, and the resulting errors were decently low.

References