A Rapid Hierarchical Radiosity Algorithm

Pat Hanrahan
Department of Computer Science
Princeton University
Princeton, NJ 08540

David Salzman
68 Francis Avenue
Cambridge, MA 02138

Larry Aupperle
Department of Computer Science
Princeton University
Princeton, NJ 08540

Abstract

This paper presents a rapid hierarchical radiosity algorithm for illuminating scenes containing large polygonal patches. The algorithm constructs a hierarchical representation of the form factor matrix by adaptively subdividing patches into subpatches according to a user-supplied error bound. The algorithm guarantees that all form factors are calculated to the same precision, removing many common image artifacts due to inaccurate form factors. More importantly, the algorithm decomposes the form factor matrix into at most $O(n)$ blocks (where n is the number of elements). Previous radiosity algorithms represented the element-to-element transport interactions with n^2 form factors. Visibility algorithms are given that work well with this approach. Standard techniques for shooting and gathering can be used with the hierarchical representation to solve for equilibrium radiosities, but we also discuss using a brightness-weighted error criteria, in conjunction with multigridding, to even more rapidly progressively refine the image.

CR Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

Key Words: radiosity, ray-tracing, global illumination, n-body problem.

1 Introduction

Developing a correct treatment of the physics of bidirectional reflectance and of light transport is an important focus of modern research in image synthesis. Although efficient solutions to the fully general case are not known, these physically-based models have produced some of the most realistic computer-generated images to date. The most successful approach has been *radiosity*, which, by making the simplifying assumption that all the surfaces are diffuse reflectors, allows for straightforward computation of the equilibrium distribution of light for complex scene geometries.

This paper presents efficient computational techniques for solving the transport equations that arise for radiosity in complex scenes. Our algorithm draws from recent insights into fast numerical algorithms for solving the N-body problem (Appel 1985; Barnes and Hut 1986; Greengard 1988). Computational efficiency is achieved by carefully analyzing the error in performing form factor integrals. Without careful error analysis, pictures may contain artifacts where the form factors have large error. More importantly, many form factor computations are done at much higher precision than is necessary. Careful error analysis, in combination with a multi-resolution representation, can be used to reduce significantly the number of interactions that are considered.

Previously we analyzed the form factor calculation between two unoccluded polygonal patches, discretized into n finer polygonal elements (Hanrahan and Salzman, 1990). We showed that the form factor matrix can always be approximated to within some preset numerical tolerance with at most $O(n)$ terms, and often many fewer. This paper extends our previous radiosity algorithm to handle scenes with many polygons, where occlusion plays an important role. Occlusion, although costly to detect, reduces the number of interactions even further. The form factor matrix is therefore sparser, allowing faster solution for equilibrium radiosities. The technique used for determining visibility is based on ray tracing, but two important optimizing heuristics are introduced. One takes advantage of visibility coherence between different levels of detail; the other is based on the observation that most interactions between patches are either totally visible or totally invisible with respect to each other. Finally, we show how to use multigridding in combination with a brightness-weighted error estimate. This leads to a faster progressive radiosity algorithm.

2 Review of Previous Work

2.1 Radiosity

Radiosity algorithms assume the environment has been discretized into small elements which have constant brightness. In this paper, we use the term "element" to describe the smallest piece of a surface subdivision, and the term "patch" for any larger pieces, including the original polygon, formed by combining elements or other patches, Enforcing an energy balance at every element yields a system of equations of the form:

$$B_i = E_i + \rho_i \sum_j^n F_{i,j} B_j$$

where B_i is the radiosity, E_i is the emissivity, ρ_i is the diffuse reflectance, $F_{i,j}$ is the form factor (the percentage of light leaving element i that arrives at element j), and n is the number of elements in the scene. Similar equations exist for all elements, yielding a linear system of equations.

$$\begin{pmatrix} 1 & -\rho_1 F_{1,2} & \dots & -\rho_1 F_{1,n} \\ -\rho_2 F_{2,1} & 1 & \dots & -\rho_2 F_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n,1} & -\rho_n F_{n,2} & \dots & 1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}$$

This system of equations can be efficiently solved using iterative algorithms such as the Gauss-Seidel method. Physically, the Gauss-Seidel method is equivalent to successively *gathering* incoming light. An alternative iteration scheme is to reverse this process by successively *shooting* light from patches in order of their brightness (Cohen et al. 1988). This has the advantage that the solution converges more quickly, and if the scene is drawn during the iteration, successive images

gradually improve as the computation proceeds (Bergman et al. 1986).

The most expensive part of the calculation is computing the form factors. Assuming two infinitesimal elements, the differential form factor between them is given by

$$F_{ij} = \frac{\cos\theta_i \cos\theta_j}{\pi r_{ij}^2} dA_j.$$

The angle θ_i (or θ_j) relates the normal vector of element i (or j) to the vector joining the two elements. The form factor from an infinitesimal area to a finite area is the integral

$$F_{ij} = \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r_{ij}^2} dA_j,$$

and the form factor between two finite areas is the double integral

$$F_{ij} = \frac{1}{A_i} \int_{A_i}\int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r_{ij}^2} dA_i dA_j.$$

These form factor formulae do not take into account occlusion. To do this requires that differential form factors be accumulated only if the two infinitesimal elements are mutually visible. The first practical approach to integrating visibility into form factor computations was the *hemi-cube* algorithm (Cohen and Greenberg 1985). The hemi-cube algorithm is simple and fast, and can be accelerated using current workstation graphics hardware. Algorithms based on ray tracing also have been proposed for form factor calculation (Malley 1988; Ward et al. 1988; Wallace et al. 1989; Sillion and Puech, 1989)

There are two major sources of error when computing form factor integrals. First, the integral is evaluated by sampling the patches in some way; since the results of uniform sampling process are subject to aliasing, early methods had noticeable aliasing errors. However, more recent methods (Wallace et al 1989) have overcome sampling errors by incorporating stochastic sampling into a ray tracer (Cook, 1986). Second, the form factor between two surface samples can be approximated by the differential form factor only if the distance separating the two samples is large compared to their size. This condition frequently occurs along edges and in corners where polygons meet. To avoid this problem, Baum et al. (1989) switch to an analytically calculated form factor in these situations. Another approach, used by Wallace et al. (1989), is to supersample adaptively the integral.

The form factor matrix is n by n, where n is the number of elements. This n^2 growth causes time and memory problems for complex scenes, The first method to reduce the computational costs was motivated by the method of *substructuring* used in finite element calculations. The polygons comprising the scene are discretized at two levels (Cohen et al. 1986). One level contains the patches into which input polygons are broken, and the other level contains the elements into which each patch is broken. Normally, the number of patches and elements are determined *a-priori*, but the number of elements can also be determined by recursive subdivision based on radiosity gradients Cohen et al.(1986). Other attempts to utilize adaptive subdivision are described in Campbell & Fussel (1990) and Heckbert (1990).

2.2 N-Body Problem

The hierarchical subdivision algorithm proposed in this paper is inspired by methods recently developed for solving the N-body problem. In the N-body problem, each of the n particles exerts a force on all the other $n-1$ particles, implying $n(n-1)/2$ pairwise interactions. The fast algorithms compute all the forces on a particle in less than quadratic time, building on two key ideas:

1) Numerical calculations are subject to error, and therefore, the force acting on a particle need only be calculated to within the given precision.

2) The force due to a cluster of particles at some distant point can be approximated, within the given precision, with a single term—cutting down on the total number of interactions.

Appel was the first to develop a hierarchical algorithm for solving the N-body problem, by approximating the forces between particles in two clusters with a single force, when the separation between the clusters significantly exceeded their sizes. A top-down traversal of a hierarchical k-d tree representing the clusters yielded an $O(n \log n)$ algorithm (Appel 1985). More recently, Esselink analyzed Appel's algorithm and showed that time needed to calculate the forces takes only $O(n)$ time (Esselink 1989), and that the observed $O(n \log n)$ running time is a consequence of the preprocessing time required to build the hierarchical data structures. Barnes & Hut developed a similar algorithm based on octrees (Barnes & Hut 1986). Greengard and Rokhlin devised the first $O(n)$ algorithm, using a p-term multipole expansion for the potential due to any cluster, along with algorithms for splitting, merging, and translating the resulting multipole expansions (Greengard 1988). The algorithm proposed in this paper is most closely related to Appel's and Barnes & Hut's algorithms; it should be mentioned that these two algorithms are very easy implement, and only take a few hundred lines of code.

The radiosity problem shares many similarities with the N-body problem which suggest that these ideas can be used to increase its efficiency. In both the N-body and the radiosity problem, there are $n(n-1)/2$ pairs of interactions. Moreover, just as gravitational or electromagnetic forces fall off as $1/r^2$, the magnitude of the form factor between two patches also falls off as $1/r^2$. Finally, according to Newton's Third Law, gravitational forces are equal and opposite, and, according to the reciprocity principle, form factors between two polygons are related.

One major difference between the two problems is the manner in which the hierarchical data structures are formed. The N-body algorithms begin with n particles and cluster them into larger and larger groups. Our radiosity algorithm, however, begins with a few large polygons and subdivides them into smaller and smaller patches. Subdividing based on the error of a potential interaction provides an automatic method for discretizing the scene within the given error bounds. The specifics of our subdivision algorithm is discussed in Section 3. The separate problem of building clusters out of individual patches is not dealt with in this paper.

Another difference is that the N-body algorithms take advantage of linear superposition; the principle of superposition states that the potential due to a cluster of particles is the sum of the potentials of the individual particles. This principle does not always apply to the radiosity problem, because of occlusion: intervening opaque surfaces can block the transport of light between two other surfaces, which makes the system non-linear. Occlusion thereby introduces an additional cost to the radiosity problem. This is discussed in Section 4.

Finally, the N-body problem is based on a differential equation, whereas the radiosity problem is based on an integral equation. The integral equation arising from the radiosity problem can, however, be solved efficiently using iterative matrix techniques. Fortunately, the hierarchy of interactions produced by our subdivision is equivalent to a block structured matrix, and the iteration can be efficiently computed. This is discussed in Section 5.

3 Form Factor Matrix Approximation

This section describes a recursive refinement procedure which simultaneously decomposes a polygon into a hierarchy of patches and elements, and builds a hierarchical representation of the form factor matrix by recording interactions at different levels of detail. We begin by describing the procedure and its results, and then proceed to analyze the error in the resulting form factors, and the number of interactions that need to be considered. This section is quite similar to Hanrahan and Salzman (1990).

Consider the procedure **Refine**:

```
Refine(Patch *p, Patch *q, float Feps, float Aeps)
{
    float Fpq, Fqp;

    Fpq = FormFactorEstimate( p, q );
    Fqp = FormFactorEstimate( q, p );

    if( Fpq < Feps && Fqp < Feps )
        Link( p, q );
    else {
        if( Fpq > Fqp ) {
            if( Subdiv( q, Aeps ) ) {
                Refine( p, q->ne, Feps, Aeps );
                Refine( p, q->nw, Feps, Aeps );
                Refine( p, q->se, Feps, Aeps );
                Refine( p, q->sw, Feps, Aeps );
            }
            else
                Link( p, q );
        }
        else {
            if( Subdiv( p, Aeps ) ) {
                Refine( q, p->ne, Feps, Aeps );
                Refine( q, p->nw, Feps, Aeps );
                Refine( q, p->se, Feps, Aeps );
                Refine( q, p->sw, Feps, Aeps );
            }
            else
                Link( p, q );
        }
    }
}
```

Refine first estimates the form factor between two patches, and then either subdivides the patches and refines further, or terminates the recursion and records an interaction between the two patches. If the form factor estimate is less than F_ϵ (**Feps** in the program), then the true form factor (not taking into consideration occlusion) can be approximated accurately by the estimate (see below), and the patches are allowed to interact at this level of detail. (The procedure **Link** records the interaction between the two patches.) However, if either of the form factor estimates is larger than F_ϵ, then the form factor estimate is not accurate, and so the patch with the larger form factor is subdivided, and **Refine** is called recursively with the smaller subpatches.

Subdiv subdivides a patch into subpatches. In our implementation, a patch is a planar quadrilateral, and it is subdivided equally into four new quadrilaterals by splitting it at its center. The subdivision hierarchy is stored in a quadtree; the pointers to the four children are stored in the fields **nw**, **ne**, **sw**, and **se**. (This data structure is similar to adaptive radiosity textures proposed in Heckbert (1990), although information is stored at all levels of the hierarchy, not just at the leaf nodes, and each level also stores a list of its interactions.) **Subdiv** returns *false* if the patch cannot be split; this condition occurs if the area of the patch is smaller than some absolute predetermined area A_ϵ, and is necessary to prevent infinite recursion in corners and along edges. If subdivision is not possible, we force the two patches to interact. Note that a patch may be refined against many patches, and so the actual subdivision of a patch may have

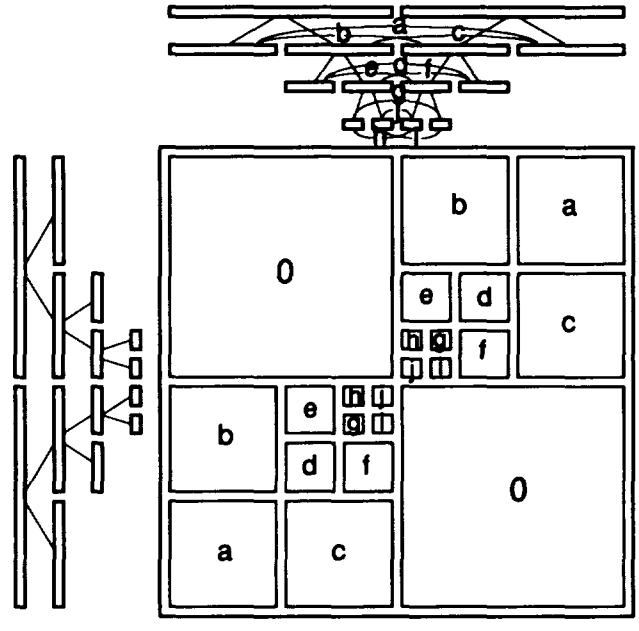

Figure 1: The block form factor matrix for a particular binary tree example. Each labelled block corresponds to a labelled arc connecting nodes in the hierarchical subdivision. Although the blocks are all square in this example, that is not the case in general.

been performed previously. When this occurs **Subdiv** need do no other work and simply returns *true*.

The procedure **FormFactorEstimate** returns an upper bound on the form factor from the first patch to the second patch, assuming the first patch has infinitesimal size and the second patch has finite size. The form factor can be estimated by either calculating the solid angle subtended by a disk with cross sectional area equal to the surface area of the patch (Wallace et al. 1989), or by circumscribing a sphere around the patch and estimating the solid angle subtended by the sphere.

An example of a tree that might be produced by **Refine** and its associated form factor matrix is shown in Figure 1. For simplicity, the figure illustrates the interactions between two hypothetical 1D patches; in this case the hierarchy can be represented with a binary rather than quaternary tree. The two binary trees representing the induced subdivision, and are drawn side by side along the edges of the form factor matrix. Since in this example each binary tree represents a polygon, no interactions are shown with itself. The leaves of the tree are the elements in the discretization. The combination of all the leaf nodes completely cover the input patch. Interactions between patches at different levels are represented by labelled blocks in the form factor matrix, and by labelled arcs between nodes in the trees. Notice that the size of the block in the form factor matrix depends on the level in the tree the patches interact at. The higher the level, the bigger the block.

The first point in the analysis is the relationship between the termination criteria and the accuracy of the computed form factors. Obviously, the termination criteria causes the form factor corresponding to each interaction to have approximately the same magnitude, because, if an estimated form factor were larger, the patches would be subdivided, otherwise, they are allowed to interact. More importantly, the termination criteria also places an upper bound on the error associated with the form factor integral between the two interacting patches. This can be verified by examining

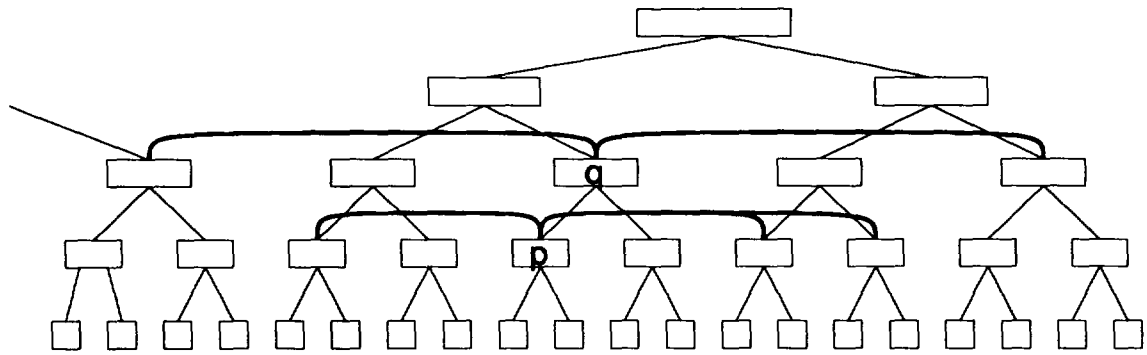

Figure 2: Interactions of the node p with neighboring nodes in a one-dimensional subdivision.

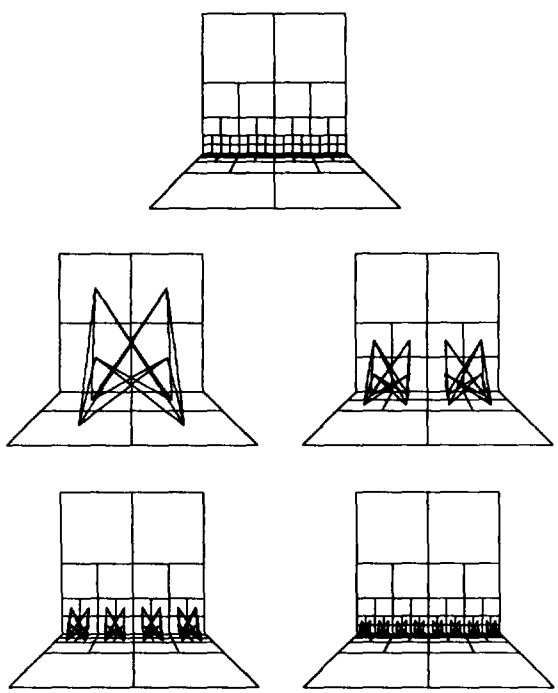

Figure 3: Interactions between a pair of perpendicular polygons.

the form factor from a point to a disk of radius r.

$$F_{\text{disk}} \approx \frac{r^2}{R^2 + r^2} = \left(\frac{r}{R}\right)^2 \left(1 - \left(\frac{r}{R}\right)^2 + \left(\frac{r}{R}\right)^4 + \cdots\right) \tag{11}$$

where R is the distance from the point to the center of the disk. Thus, the error due to the finiteness of the geometry is given by terms involving powers of (r/R). Because F goes as $(r/R)^2$, when F is small (implying that the size of the patch is small compared to the distance separating the patches), the differential form factor is also a good estimate of the true form factor. A more rigorous proof of this result can be obtained by forming the Taylor expansion of the form factor integral. In the N-body problem, this expansion is the multipole expansion. However, one need not ever calculate the expansion explicitly to use this algorithm.

The second point in the analysis is that the resulting form factor matrix has fewer than n^2 blocks. To a certain extent this is obvious, because every time an interaction occurs at some higher level of detail, the number of interactions is reduced, but we wish to count the interactions more precisely. For simplicity, again consider the 1D problem of n equally spaced patches along a line. Later we will consider what happens if the patches are 2D and non-uniformly distributed. Let us construct a binary tree above the patches by merging adjacent contiguous patches recursively. This is shown in Figure 2. The error criterion says that two patches can interact directly only if $(r/R)^2 < F_\epsilon$. In other words, two patches of size r can interact only if the distance R between them is greater than $r/\sqrt{F_\epsilon}$. For concreteness, let us fix F_ϵ so that this criterion is equivalent to saying that two patches at the same level in the binary tree can interact only if at least one other patch at that level is between them: Otherwise, they would subtend too large a solid angle and would subdivide, pushing the interaction down a level in the tree. Now consider the interactions of a patch p in the interior of the tree. At any level in the tree, the rule forbids the patch p from interacting with its immediate neighbors. These immediate neighbor interactions, therefore, must be handled by p's children. In the same way, p is only responsible for handling the interactions from its parent q's immediate neighbors. Therefore, p need only interact with the children of q's immediate neighbors. Figure 2 shows the node p and its parent q. The above considerations imply that p need only make three connections to nodes at its level. This argument applies to all levels of the tree (except the top and the bottom, but these levels result in fewer interactions), and therefore each node in the tree connects to a constant number of other nodes. Thus, the total number of interactions is proportional to the number of nodes in the tree, which is $O(n)$. A similar analysis has been derived independently by (Esselink 1989).

Figure 3 shows the quadtree subdivision and the interactions at each level in the hierarchy computed by `Refine` between a pair of perpendicular polygons. This figure shows that each interior patch has a constant number of interactions with other patches regardless of the level in the tree.

Figure 4 plots the actual number of interactions versus the number of potential interactions at a fixed uniform level of discretization. The number of interactions for perpendicular polygons goes, surprisingly, as $O(\sqrt{n})$. The subdivision induced between two perpendicular polygons is comparable to a binary tree turned on its side with its leaf nodes along the common edge, and the total number of nodes in such a sideways binary tree will be $O(\sqrt{n})$. The worst case for `Refine` is two parallel polygons whose size is much larger than the distance separating them. In this case, there will be $O(n)$ interactions. As the polygons move further apart, or are tilted relative to each other as in the case of perpendicular polygons, the number of interactions is reduced. Finally, as the two polygons move still farther apart, eventually only a

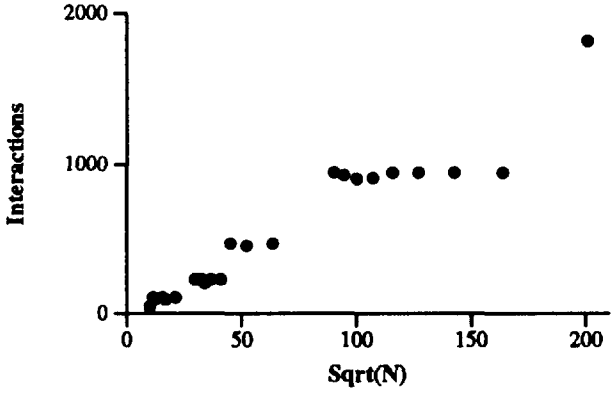

Figure 4: Number of interactions vs. number of elements for a pair of perpendicular polygons.

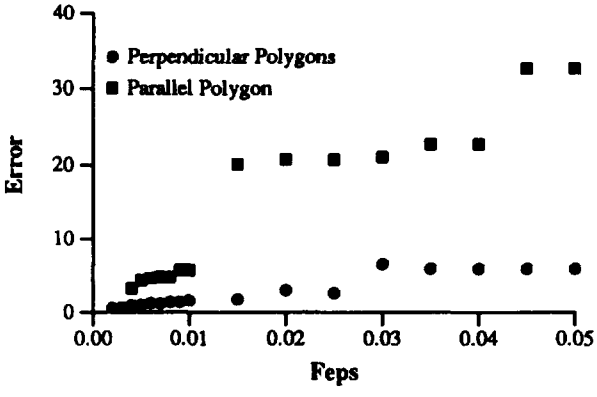

Figure 5: Measured relative percentage error vs. F_ϵ.

single interaction is required.

To verify the accuracy of the form factors generated by our method, we compared the computed form factors with the analytical form factors which are available for the parallel and perpendicular geometries (see, for example, (Siegel and Howell 1981)). To compute the form factor between two finite areas, **Refine** can be modified to return the sum of the form factors of a patch's children or, if the patch is a leaf, the product of the patch's area and the differential form factor to the other patch. Figure 5 shows the measured relative error between the computed and the analytical form factors as a function of F_ϵ. As expected, the actual error in the form factor is proportional to the F_ϵ given to **Refine**, as predicted by the theory. Note that the plateaus in these figures are due to the discrete nature of the subdivision.

In summary, our hierarchical refinement method estimates the form factor matrix between two unoccluded patches to within a fixed error tolerance automatically. In the process it reorganizes the form factor matrix into $O(n)$ or fewer blocks; the estimated form factor associated with each block has the same value and error as other blocks.

4 Visibility

The pairwise method for computing form factors described in the previous section is accurate as long as each patch is completely visible with respect to the other patch. Unfortunately, occlusion exists in all realistic environments, and so this idealization is not very useful in practice. In this section we modify the algorithm to take into consideration visibility.

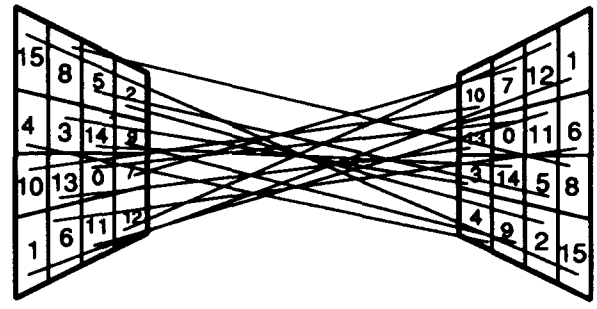

Figure 6: Jittered rays fired between two polygons to determine the percentage visibility.

Intervening occluding surfaces can only decrease light transport between two patches, thus, the true form factor in the presence of occlusion is never greater than the form factor estimate described above. The effect of occlusion can be modeled by multiplying the estimated form factor by a visibility correction factor which estimates the percentage each patch sees of the other.

$$F = V_e F_e$$

where F_e is the estimated form factor without considering occlusion, and V_e is the estimated visibility. If $V_e = 1$ then the two patches are totally visible; if $V_e = 0$ then they are completely occluded; and otherwise they are partially visible. Thus, assuming no visibility error, the level of detail for the interaction between two patches need never be finer than that computed by the procedure **Refine**.

Recall that all the form factor estimates computed by **Refine** have approximately the same error. This fact has two important consequences. First, since the form factor is not precise, the calculation of V_e need only be estimated to the same precision. Ideally, the visibility module should take into account the precision required; in reality, current visibility modules probably compute visibility much more accurately than is necessary. Second, since all the visibility estimates should have approximately the same error, it is reasonable to perform the same amount of work per estimate. This means that the total number of visibility tests required is proportional to the number of interactions. The total amount of work performed is:

$$T(n) = F(n)V(n)$$

where $F(n)$ is the number of computed form factors and $V(n)$ is the cost of performing the visibility test for a given number of elements. As has been shown, $F(n)$ varies at most linearly with n, so many fewer visibility tests need be done than with conventional radiosity algorithms.

In our current implementation, we perform two types of visibility tests. The first visibility test determines whether two polygons face each other, face away from each other, or if the support plane of one polygon splits the other. This test considers only the two polygons and not the environment, and therefore can be done in constant time. The second visibility test checks how much of each polygon is visible from the other polygon given the global environment. The test fires a fixed number of rays between the two patches, and computes the percentage of rays not blocked by intervening surfaces. The same number of rays are fired per interaction, because all the visibility estimates should have the same error. Each patch is subdivided into a 2D grid (typically 4x4), and the cells in the grid are assigned numbers from a magic square (Cook 1986). Each ray is really a line segment formed by joining jittered points within corresponding cells with the same number as shown in Figure 6. A naive ray intersection test takes $O(n)$ time. In order to accelerate the visibility test, we use a modified version of the BSP-tree algorithm,

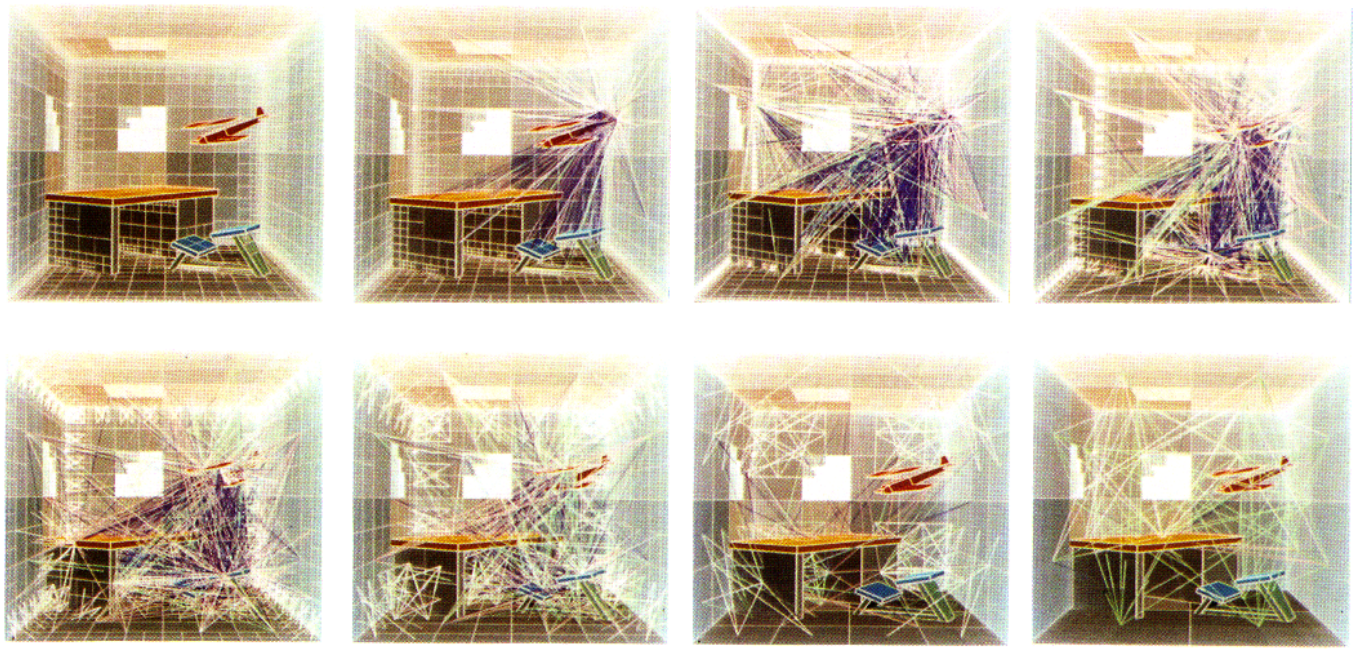

Figure 7: Hierarchical subdivision, interactions, and visibility. This sequence shows the hierarchical subdivision of each polygon, from the smallest elements in the upper left image, up the hierarchy to the largest subpatches shown in the image at lower right. Line segments link interacting patches, shown at the hierarchical level of the smaller patch. Segment color indicates visibility: white – completely visible, green – partially visible, pink – cut by supporting plane, and dark blue – relatively invisible. Note that there are many more visible and invisible links than partial.

described in Thibault & Naylor (1987), which in principle reduces the intersection cost to $O(\log n)$. This ray intersection module recalls shadow testing, because it returns only whether the ray is blocked or not, and not the closest intersection along the ray direction.

Figure 7 shows a simple scene. Induced subdivision is indicated for each polygon, and interactions between subpatches are illustrated as links. The series shows all links, ordered by increasing height, within the hierarchy, of the linked subpatches. Note that large patches interact with other large patches that are far away, whereas smaller patches interact with similarly sized patches at closer distances. This hierarchy is most clearly shown in the corners and along the edges of the room.

A tentative list of interactions was computed by refining patches relative to each other irrespective of visibility. Each interaction was then tested for visibility and the form factor adjusted. In the figure, links between interacting patches that are completely visible relative to each other are colored white. Green links interacting patches that are partially visible with respect to each other, pink links those for which one patch's supporting plane splits the other, and dark blue links those patches whose interactions are found to be completely occluded (invisible).

Table 1 shows some basic statistics for this scene. Note that the total number of interactions is only 15526, compared with over 10^9, assuming all elements interacted directly. The total time to compute this picture was approximately one minute (all times quoted in this paper are on a SiliconGraphics 210 GTX).

The interactions in scenes such as those shown in Figure 7 exhibit a great deal of *visibility coherence*. Fringes of partial and splitting links tend to surround larger areas of complete visibility or occlusion (see also Figure 8, below). Partial visibility corresponds to penumbral areas generated along the silhouettes of an object, which occur less often than the interior or exterior of an object, although pathological exceptions may be easily constructed. In the scene shown in Figure 7, (see Table 1), we find that 52.6% of the interactions are totally visible, 28.8% are totally invisible, and only 18.5% are partially visible.

The recursive refinement procedure can exploit visibility coherence in a natural fashion to prune out unnecessary refinement and visibility calculation. If, in the course of subdivision, two subpatches become totally invisible relative to each other, then the refinement between them can be immediately terminated, significantly reducing the number of calls to **Refine**. If two patches become totally visible, there is no need for further visibility tests between them, although further refinement may still need to occur. Finally, if two patches are partially visible with respect to each other, further refinement would require additional visibility computation. However, as patches are subdivided, their visibility will tend to fall into the visible or invisible category, only those on the fringes remaining partial. Employing these optimizations on the scenes shown in Figure 7 cuts in half the number of rays fired in visibility tests, reducing running time by fifty percent as well.

The nature of visibility coherence, and the refinement procedure's use of this coherence, should be reflected in the visibility test. Total visibility or invisibility are the most common visibility interactions, and result in immediate pruning of computation, thus the visibility test should quickly detect these cases. Jim Blinn calls this principle *triage* (Blinn 1990). A partial visibility result simply indicates that further visibility computation is necessary lower in the hierarchy. An precise estimate of percentage visibility is not required until refinement is terminated and the patches are to be linked. Thus, for the purposes of refinement, the visibility test need only detect partial visibility situations, not completely analyze them. Furthermore, it is acceptable to

Polygons	98	
Potential elements	44773	
Potential interactions	1002288378	
Without visibility coherence		
Patches	7286	
Elements	5489	
Interactions	15526	
Totally-invisible	4477	28.8%
Totally-visible	8171	52.6%
Partially-visible	2878	18.5%
Tests		
Refinement tests	19117	
Visibility tests	11123	
Ray tests	177968	
With visibility coherence		
Patches	7350	
Elements	5537	
Interactions	15598	
Totally-invisible interactions	4495	28.8%
Totally-visible interactions	8249	52.9%
Partially-visible interactions	2854	18.3%
Tests		
Refinement tests	19213	
Totally-invisible refines	3600	18.7%
Pre-Totally-invisible refines	0	0.0%
Totally-visible refines	10487	54.6%
Pre-Totally-visible refines	9700	50.5%
Partially-visible refines	5126	26.7%
Partial visibility tests	9513	
Ray tests	20527	
Visibility tests	4296	
Ray tests	68736	

Table 1. Statistics for Figure 7.

return a partial visibility result if the visibility situation is complex, as additional subdivision will tend to reduce the complexity of the visibility calculation. This is very similar to Warnock's visible surface algorithm (Warnock 1969).

The methods used to detect visibility are likely to be more accurate when patches are totally visible or totally invisible. When two patches are partially visible, we assume there is more likely to be an error in visibility and increase the error in the form factor estimate. This causes increased subdivision in regions of partial visibility; the cost of this is minor because they occur so infrequently, however, the benefits are great because these often arise at shadow boundaries where there are sharp intensity gradients.

5 Solution Techniques

Once the form factors have been determined, the next step is to solve for the radiosities. The most efficient way to do this is to invert the matrix iteratively. Each iteration involves multiplying a matrix times a vector, which normally takes $O(n^2)$ operations. However, because the form factor matrix is represented with $O(n)$ blocks, each matrix multiplication can be done in linear time. In this section we give program fragments that implement the technique of gathering and briefly explain how to implement shooting. These techniques are quite similar to the unoccluded case, and we refer the reader to Hanrahan and Salzman (1990) for more details.

5.1 Shooting and Gathering

The classic Jacobi iteration (which differs from the Gauss-Seidel in that the brightnesses are not updated in-place) can be implemented using the following simple recursive procedure.

```
Gather( Patch *p )
{
    Patch *q;
    float Fpq;

    if( p ) {
        p->Bg = 0.0;
        ForAllElements( q, p->interactions ) {
            Fpq = FormFactor( p, q );
            p->Bg += Fpq * p->Cd * q->B;
        }
        Gather( p->sw );
        Gather( p->se );
        Gather( p->nw );
        Gather( p->ne );
    }
}
```

The average brightness of each patch is stored in `B` and its diffuse color is stored in `Cd`. The brightness gathered is stored in `Bg`, and is computed by receiving energy from all the patches `q` stored on the list of interactions of `p` (`p->interactions`).

The total amount of energy received by an element is the sum of the energy received by it directly, plus the sum of all the energy received by its parent subpatches. To update the energies for the next iteration, all the energy gathered is *pushed* down to the leaf nodes, and then *pulled* upward towards the root polygon. During this upward pass, the radiosity of interior subpatches are set equal to the area weighted average of its children's radiosities. Both these operations can be done in a single depth-first traversal of the quadtree, which takes time proportional to the number of nodes in the hierarchy.

The radiosity equation can be solved by shooting instead of gathering. All patches in the hierarchy are sorted into a priority queue based on their brightness. A patch at a time is taken off the queue, and its energy shot to the patches that interact with it. This version of shooting, however, has a much smaller granularity then the classic method of shooting used in progressive refinement. This is because in our algorithm each patch shoots light to a constant number of other patches, whereas in the previous algorithms a patch shoots light to the entire scene.

5.2 Multigridding and BF Refinement

An interesting variation of shooting or gathering refines the hierarchy as the iteration proceeds. This is similar to the idea of multigridding, where a finite difference equation is solved first at a coarse resolution, and then at successively finer resolutions. The advantage of multigridding is that the coarse solution involves a low resolution iteration that can be performed cheaply. This coarse solution provides a better starting point for the costlier iterations at the finer resolutions, resulting in fewer expensive iterations before convergence. Multigridding allows for an even more progressive radiosity algorithm: Shooting is performed in the early stages at coarse resolutions to get a rough idea of the image, and then at successively finer and finer resolutions as the calculation proceeds.

Multigridding is easily incorporated into the algorithm by successively refining the mesh with smaller and smaller F_ϵ's. The procedure `Refine` is extended to delete the link indicating a previous interaction at a given level of detail, if subdivision is required. `Refine` is then called between iterations to increase the resolution of the grid.

A final improvement to the algorithm bases the refinement of two patches on BF; that is, on the total amount of energy potentially transported between the patches. The procedure `Refine` is extended to use this test for subdivision rather than F alone. This causes refinement of the mesh to be put off until energy is actually available to be transported,

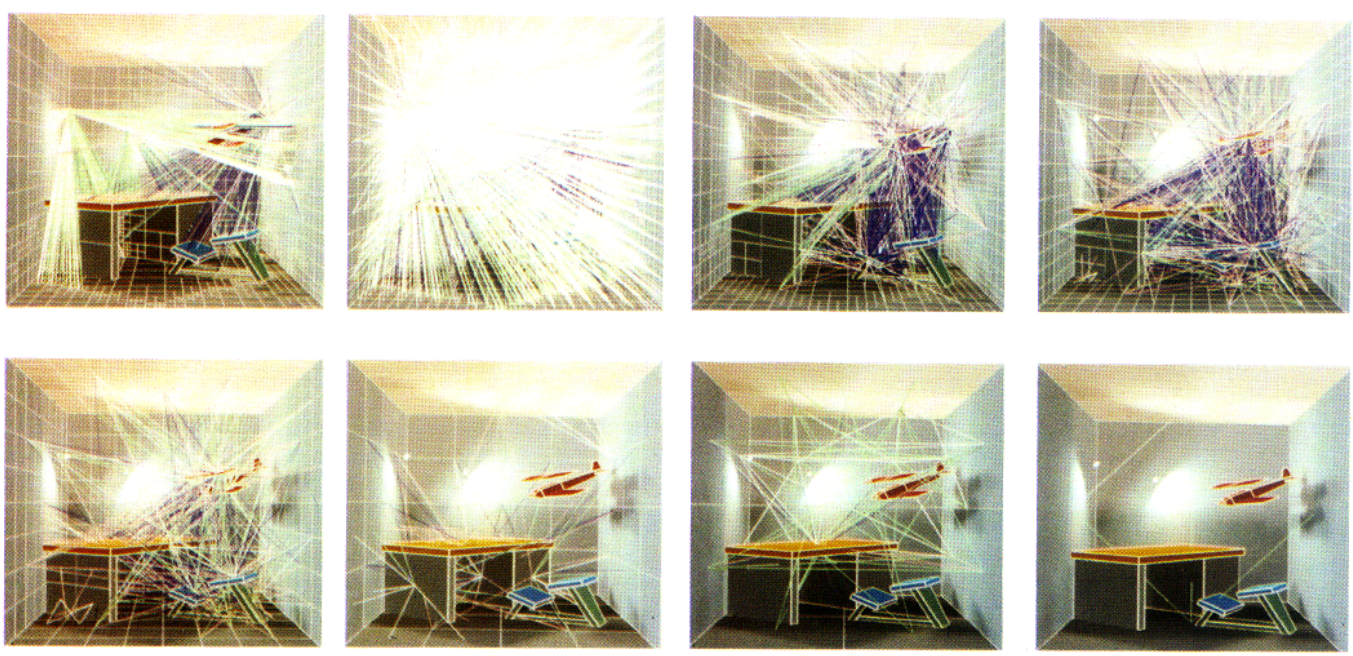

Figure 8: *BF* refinement. Refinement is based on total energy transport between patches. Thus there are many interactions with the light sources, and a reduced number of interactions in areas exhibiting less energy transport, such as corners. White – total visibility, green – partial visibility, pink – cut by supporting plane, and dark blue – total occlusion.

thus saving even more work on early iterations. This works particularly well in corners which normally contain a large number of interactions because of their proximity, but tend to be dark because light must reflect off several surfaces to reach the inner recesses.

With *BF* refinement and multigridding, shooting has no advantage over gathering. Since all interactions carry approximately the same amount of energy, there is no advantage to sorting them based on brightness.

Figure 8 shows the subdivision and interactions based on a *BF* error criteria for the same scene as shown in Figure 7. As in the previous figure, the series shows all links, ordered by increasing height in the hierarchy. To accentuate the effect, we have made the two lamps small and very bright. Note that this causes many more interactions with the lights than between other parts of the room; all the interactions at the finest level of detail in the corners are eliminated because they are inconsequential compared to the light interactions.

Another effect clearly shown in this figure is the increased subdivision along shadow boundaries. This is exhibited in the first image of the series in the subdivision of the right wall behind the biplane, and on the floor in the neighborhood of the desk. The second and third images in the series illustrate the corresponding links; note the spray of interactions ranging in from white (visible) to green (partial) to dark blue (occluded).

Table 2 gives basic statistics for the scene in Figure 8. The number of interactions is approximately 10,000, which is less than the number in Figure 7, even though the number of potential interactions in this scene is sixteen times greater.

Figure 9 shows a series of iterations using multigridding coupled with *BF* refinement. At the bottom are the induced subdivisions, and above the resulting images. The first image shows the initial mesh, the next two images show the mesh after iterations in which the error has been decreased, and the final iteration improves the final image, but does not involve decreasing the error in the mesh. Note the gradual refinement of the ceiling and right wall, as they are illuminated by the lamps and light reflected from the desktop. In the last image, enough light has been transported into the neighborhood of the near end of the desk to induce its subdivision. Table 3 gives the error bound for each iteration, as well as the number of patches, elements and interactions. Note that the last iteration in which error bound was not changed does not involve refining, and hence takes much less time than the other iterations.

Polygons	98	
Potential elements	175964	
Potential interactions	15481576666	
Patches	5674	
Elements	4280	
Interactions	11800	
Totally-invisible	4605	39.0%
Totally-visible	4519	38.3%
Partially-visible	2676	22.7%
Tests		
Refinement tests	14149	
Totally-invisible refines	3901	27.6%
Pre-Totally-invisible refines	0	0.0%
Totally-visible refines	5414	38.3%
Pre-Totally-visible refines	4128	29.2%
Partially-visible refines	4834	34.2%
Partial visibility tests	10021	
Ray tests	53187	
Visibility tests	3545	
Ray tests	56720	

Table 2. Statistics for Figure 8.

Figure 9: Multigridding and BF refinement.

Patches	Elements	Interactions	Error	Time(s)
5674	4280	11800	1.000	45
7646	5759	16216	0.707	34
10462	7871	24091	0.500	62
10714	8060	24886	0.500	10

Table 3. Statistics for Figure 9.

6 Results

Figure 10 shows an example image created by the algorithm. At the maximum level of detail, it contains potentially 52841 elements, of which 12635 patches are actually created by refinement. Using classical radiosity, this would require 1.4 billion interactions, whereas the algorithm requires only 20150. This image was produced in three minutes and fifty-seven seconds.

7 Summary and Discussion

The radiosity algorithm proposed in this paper drastically reduces the number of interactions that need to be considered while maintaining the precision of the form factors that are calculated. This reduction in the number of form factors allows much higher-quality imagery to be generated within a given amount of time or memory. Successively refining the environment using a brightness-weighted error criteria leads to a algorithm where the granularity of each step in the progression is much smaller than in the standard progressive refinement algorithm. This allows for more control and faster updates in interactive situations.

The algorithm proposed works best for environments with relatively few large polygons with high brightness gradients that require the polygon to be broken into many elements. This is very common in architectural environments, but there are situations where this assumption is not valid. The general principles outlined in this paper are still valid in these situations, but the methods for producing the hierarchy and estimating visibility would be quite different. Useful

Figure 10:

applications for such algorithms are for rendering volumes and participating media.

One of the emerging themes of realistic image synthesis is that the geometric aspects of the problem are becoming subservient to the optical aspects. The optical portion involves numerically solving an integral equation; the geometric portion involves primarily determining visibility between the finite elements used to discretize the equation. Unfortunately, most visibility algorithms developed in computer graphics were not developed with these numerical calculations in mind. What are needed are fast algorithms that compute visibility to within a given precision. Ideally, the less the precision, the faster the algorithm. Visibility algorithms also need to be developed that consider patch-to-patch interactions and not just point-to-patch interactions, as are almost exclusively the case. Finally, what are needed to take advantage of the coherence found in typical environments are fast algorithms for detecting whether patches are totally visible or totally invisible with respect to each other.

8 Acknowledgements

The authors wish to thank Andrew Appel, Dan Baum, David Laur, Toby Orloff, Jeffrey Posdamer, and James Winget for helpful comments. Brian Danella and S.V. Krishnan provided assistance with modeling and rendering.

9 References

Appel, A.A. (1985) An efficient program for many-body simulation. *SIAM J. Sci. Stat. Computing* 6(1), 85-103.

Barnes, J., Hut, P. (1986) A hierarchical $O(NlogN)$ force-calculation algorithm. *Nature* 324, 446-449.

Baum, D.R., Rushmeier, H.E., Winget, J.M. (1989) Improving radiosity solutions through the use of analytically determined form factors. *Computer Graphics* 23(3), 325-334.

Bergman, L., Fuchs, H., Grant, E., Spach, S. (1986) Image rendering by adaptive refinement. *Computer Graphics* 20(4), 29-38.

Blinn, J. (1990) Triage Tables. *IEEE Computer Graphics and Applications*, 10(1) 70-75.

Campbell, A.T., Fussel, D.S. (1990) Adaptive mesh generation for global diffuse illumination. *Computer Graphics* 24(4), 155-164.

Cohen, M.F., Greenberg, D.P. (1985) The hemi-cube: A radiosity approach for complex environments. *Computer Graphics* 19(3), 31-40.

Cohen, M.F., Greenberg, D.P., Immel, D.S., Brock, P.J. (1986) An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics and Applications* 6(2), 26-30.

Cohen, M.F., Chen, S.E., Wallace, J.R., Greenberg, D.P. (1988) A progressive refinement approach to fast radiosity image generation. *Computer Graphics* 22(4), 75-84.

Cook, R.L. (1986) Stochastic sampling in computer graphics. *ACM Transactions on Graphics* 5(1), 51-72.

Esselink, E. (1989) About the order of Appel's algorithm. Computing Science Note KE5-1, Department of Computer Science, University of Groningen.

Greengard, L. (1988) *The rapid evaluation of potential fields in particle systems.* MIT Press, Cambridge, MA.

Hanrahan, P., Salzman, D.B. (1990) A rapid hierarchical radiosity algorithm for unoccluded environments. Published in K. Bouatouch, Photosimulation, Realism and Physics in Computer Graphics. Springer-Verlag (1991), Reprinted as Princeton University CS-TR-281-90.

Heckbert, P.S. (1990) Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics* 24(4), 145-154.

Malley, T.J.V. (1988) A shading method for computer generated images. Master's Thesis, The University of Utah

Siegel, R., Howell, J.R. (1981) *Thermal radiation heat transfer.* Hemisphere Publishing Co., Washington, DC

Sillion, F., Puech, C. (1989) A general two-pass method for integrating specular and diffuse reflection. *Computer Graphics* 23(3), 335-344.

Thibault, W., Naylor, B. (1987) Set operations on polyhedra using binary space partitioning trees. *Computer Graphics* 21(4), 153-162.

Wallace, J.R., Elmquist, K.A., Haines, E.A. (1989) A ray tracing algorithm for progressive radiosity. *Computer Graphics* 23(3), 315-324.

Ward, G.J., Rubinstein, F.M., Clear, R.D. (1988) A ray tracing solution for diffuse environments. *Computer Graphics* 22(3), 85-92.

Warnock, J. (1969) A hidden-surface algorithm for computer-generated half-tone pictures. Technical Report TR 4-15, NTIS AD-753 671, Computer Science Department, University of Utah.