# A RAPID HIERARCHICAL RADIOSITY ALGORITHM FOR UNOCCLUDED ENVIRONMENTS

Pat Hanrahan
David Salzman

CS-TR-281-90

August 1990

# A Rapid Hierarchical Radiosity Algorithm for Unoccluded Environments

*Pat Hanrahan*

Department of Computer Science
Princeton University
Princeton, NJ 08540
hanrahan@princeton.edu

*David Salzman*

John von Neumann National Supercomputer Center
Princeton, NJ 08540
salzman@princeton.edu

## Abstract

This paper presents a linear-time radiosity algorithm for scenes containing large mutually unoccluded polygonal patches. It subdivides pairs of patches adaptively to build a hierarchical data structure with $n$ elements at the leaves, and it encodes all the light transport between component polygonal elements. Given a required numerical precision, determined here by the specified bounds for maximum solid angle $F_\epsilon$ and minimum area $A_\epsilon$, our algorithm reduces the number of form factor calculations and interactions to $O(n)$ in the worst case and $O(\sqrt{n})$ in the best case. Standard techniques for shooting and gathering can then be used with the data structure. The best previous radiosity algorithms represented the element-to-element transport interactions with $n^2$ form factors.

## Introduction

Methods for producing realistic images from geometric descriptions of a scene have always been one of the central themes of research in computer graphics. The earliest "local illumination" algorithms considered only geometric and material properties of objects in isolation, and measured the amount of light reflected from a surface to the viewer without considering objects that might shadow or obscure the surface. More modern "global illumination" algorithms, however, consider the entire environment and produce pictures with shadows, reflections, refractions, diffuse inter-reflection & color bleeding, and light focusing or caustics. Although the physics of global illumination can be formulated as an integral "rendering" equation describing the transport of light within an environment (Kajiya 1986), fully general solutions are computationally intractable in practice.

Radiosity offers the best case today in which the general rendering equation can be solved (Greenberg 1989). Radiosity makes the simplifying assumption that all surfaces scatter light isotropically, so the intensity of outgoing energy has no directional dependence, and the directional distribution of light at any point on surface can be represented as a single number. There are two major steps in solving for radiosities. First, geometric form factors are computed between all pairs of surfaces. These form factors give the proportion of light leaving one surface that is incident on another. Second, a large system of simultaneous linear equations involving form factors and brightnesses must be solved to yield the light energy transported to each surface. Researchers have devised better methods in recent years both for computing form factors and for calculating brightnesses.

This paper describes a method for applying techniques recently developed for efficiently solving the N-body problem (Appel 1985; Barnes and Hut 1986; Greengard 1988) to the radiosity problem. We concentrate on the case of $k$ large unoccluded polygonal patches, discretized into $n$ finer polygonal elements. We show that the form factor matrix can always be approximated to within some preset numerical tolerance with $O(n)$ terms. Each term corresponds to coalescing a $n_1 \times n_2$ rectangular block in the form factor matrix, and representing the sum of the entries with a single number. Significantly fewer than $n^2$ independent form factors need to be computed, speeding up the first step in the radiosity algorithm. The successive matrix iterations needed to compute the brightnesses also run faster, since the multiplication of the $O(n)$ blocks in the form factor matrix by the column vector of brightnesses can be done in $O(n)$ time.

## Review of the Basic Radiosity Formulation

This section briefly describes the matrix formulation of the radiosity problem. More details are contained in standard textbooks on radiative heat transfer (Siegel and Howell 1981; Sparrow and Cess 1978).

Radiosity algorithms assume that all surfaces in the environment are perfect Lambertian reflectors, and hence scatter light equally in all directions. The energy distribution per unit area per unit solid angle is called *radiosity*, which we will also refer to as *brightness*. The radiosity of a diffuse material has no directional dependence, and can be denoted by a single number. We assume that a scene description has been discretized into $n$ elements, where each element is small enough that the brightness does not vary significantly across its surface. The energy leaving each element $i$ is given by

$$B_i A_i = E_i A_i + \rho_i \sum_{j}^{n} F_{ji} B_j A_j \tag{1}$$

where:

$B_i$ is the brightness of element $i$,

$E_i$ is the emissivity of element $i$,

$A_i$ is the area of element $i$,

$\rho_i$ is the diffuse reflectivity of element $i$,

$F_{ij}$ is the form factor from element $i$ to element $j$.

Equation (1) represents an energy equilibrium. It states that the radiosity (light energy) leaving element $i$ consists of the light it emits directly plus the sum of all the diffusely scattered light energy incident on it and retransmitted. The contribution of element $i$ to the brightness of element $j$ is equal to its own brightness times a form factor, giving the proportion of light leaving element $i$ which reaches $j$, times the diffuse scattering coefficient of $j$.

The radiosity algorithm involves two steps. First, one calculates the form factors. The form factor is purely geometric, and is proportional to the solid angle subtended by the emitter from the vantage point of the element receiving the light, assuming all points on the emitter are visible from the receiver. The differential form factor between two infinitesimal areas is given by

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

This is illustrated in Figure (1). The angle $\theta_i$ (or $\theta_j$) relates the normal vector of element $i$ (or $j$) to the vector joining the two elements. The cosine factors arise because the brightness is measured per unit area in the direction in which light is transported. The form factor from an infinitesimal area to a finite area is

$$F_{ij} = \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r_{ij}^2} dA_j, \tag{3}$$

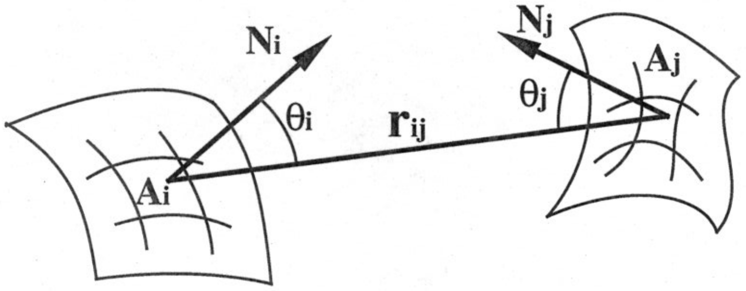

*Figure 1: Differential Form Factor Geometry*

and the form factor between two finite areas is

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

For an enclosed environment, where all energy leaving a surface eventually falls on another surface, the form factors sum to 1,

$$\sum_i^n F_{ij} = \sum_j^n F_{ij} = 1. \tag{4}$$

Moreover, the *reciprocity principle* states that the amount of light exchanged between emitter and receiver is unchanged if they are swapped, so

$$A_i F_{ij} = A_j F_{ji}. \tag{5}$$

Form factors must be approximated numerically, since the integrals in Equation (3) do not yield closed-form solutions for the geometries commonly encountered in computer graphics. For example, there appears to be no closed-form expression for the form factor between two polygons, although there is a simple closed-form solution for the form factor between a differential area and a polygon (Hottel and Sarofim 1967; Baum et al. 1989). The calculation of a form factor between two surfaces is further complicated by the need to determine the mutual visibility of all pairs of points on the two surfaces. A surface may hide a portion of itself, or be partially eclipsed from another element by intervening objects.

A number of researchers have invented projection methods for computing the differential form factor to any arbitrary shape. Nusselt projected a shape onto a sphere and measured its cross sectional area when viewed from the top of the sphere (Siegel and Howell, 1981; Maxwell et al. 1986). Cohen modified this method by projecting shapes onto a hemi-cube (Cohen and Greenberg 1985), since the hemi-cube projection is easier to compute with traditional computer graphics techniques. Each pixel on the hemi-cube stores information about the visibility of objects within the solid angle which projects onto that pixel's area. Associated with each pixel is a *delta form factor*. The total form factor for an element is equal to the sum of the delta form factors from each pixel which it covers on the hemi-cube. Sillion improved on this further by projecting onto a single plane positioned close to the differential area (Sillion and Peuch 1989), which entailed only a single projection where the hemi-cube used five.

Several sources of error arise when computing form factors using a hemi-cube. Most notable is the aliasing, or beating, that occurs between the sampled patch elements and the sampled solid angles corresponding to pixels on the hemi-cube. Another source of error arises because projection methods compute form factors from source to receiver, whereas the form factor from receiver to source is needed. The reverse form factors can be computed using the reciprocity principle, but this will be inaccurate when the source is larger than the receiver or when parts of the source are not visible from the receiver. The other major source of error arises when two areas are large relative to their separation. When this occurs, Equation (2) cannot be used to estimate the form factor between them (Baum et al. 1989).

Ray tracing offers an alternative to projection methods for computing form factors (Maxwell et al. 1986; Ward et al. 1988; Wallace et al. 1989). The simplest method is to break a patch uniformly into small elements, and trace rays between the elements to determine mutual visibility. If the elements are small, the form factor between two elements can be approximated by the equation for the differential form factor. A better method, however, is to approximate the element-element form factor with the simple (closed-form) differential form factor for a point to a disk; this allows the form factor to be estimated with fewer larger samples. The accuracy of the form factor can be improved further by jittering sample positions on the source to prevent aliasing of shadow boundaries, by supersampling until the form factor converges, and by adaptively subdividing the source based on the amount of energy received (Wallace et al. 1989). These methods address many of the sources of error described in the preceding paragraph.

The second step in a radiosity algorithm is the calculation of the brightnesses. This requires solving a linear system of equations.

Using the reciprocity principle, the basic radiosity Equation (1) can be divided through by $A_i$ and then rewritten in the following form

$$B_i = E_i + \rho_i \sum_{j}^{n} F_{ij} B_j. \tag{6}$$

A similar equation can be written for every element. All these equations hold simultaneously, so the solution to the radiosity transport equation is the solution of the following matrix equation.

$$\begin{pmatrix} 1 & -\rho_1 F_{1,2} & \cdots & -\rho_1 F_{1,n-1} & -\rho_1 F_{1,n} \\ -\rho_2 F_{2,1} & 1 & \cdots & -\rho_2 F_{2,n-1} & -\rho_2 F_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\rho_{n-1} F_{n-1,1} & -\rho_{n-1} F_{n-1,2} & \cdots & 1 & -\rho_{n-1} F_{n-1,n} \\ -\rho_n F_{n,1} & -\rho_n F_{n,2} & \cdots & -\rho_n F_{n,n} & 1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_{n-1} \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_{n-1} \\ E_n \end{pmatrix} \tag{7}$$

Several methods have been used to solve this system of equations. Direct matrix inversion (e.g. Gauss's method with partial pivoting) can solve the linear radiosity system in $O(n^3)$ steps. This is the method used in the original radiosity paper (Goral et al. 1984). The matrix can also be inverted using iterative methods such as the Gauss-Seidel algorithm (Cohen and Greenberg 1985). Each iteration requires multiplying a matrix times a vector, which requires only $O(n^2)$ operations. Moreover, because the form factor matrix is strictly diagonally dominant, iterative methods are guaranteed to converge to a solution (Faddeev and Faddeeva 1963), often after only a few steps. Iterative methods also have the advantage that form factors can be computed one row at a time, as needed. Thus, they need not be precomputed and stored, which becomes desirable when the number of elements is large. On the other hand, when many iterations will be required before the brightness values converge, it remains preferable to save the form factors rather than recompute them. Physically, each iteration of the Gauss-Seidel algorithm means that the new brightness of a patch is computed by *gathering* the incoming energy from all the other patches. This is illustrated in Figure (2A).

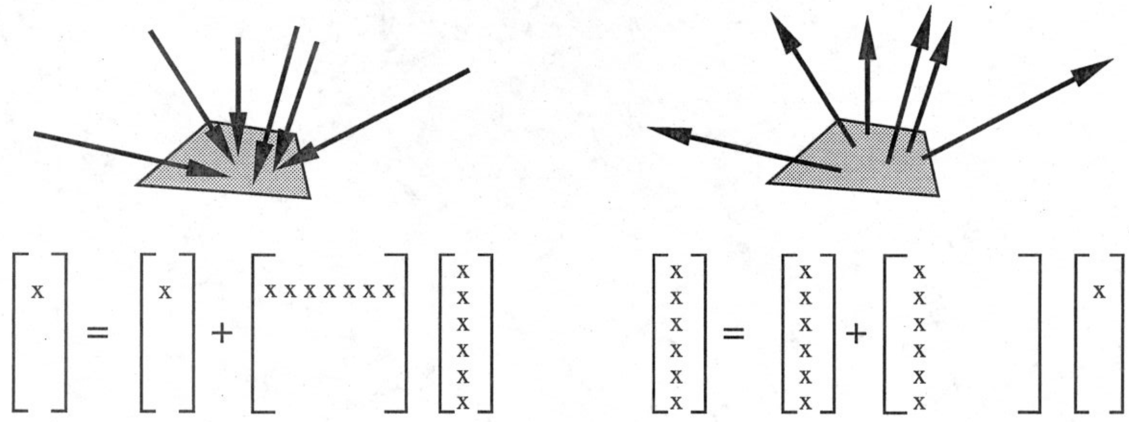

*Figure 2: Gathering vs. Shooting*

A competing way to organize the iteration is to *shoot* radiosity from one element at a time to all the other elements in the environment as shown in Figure (2B) (Cohen et al. 1988). This entails multiplying a column of the form factor matrix by one brightness value at a time:

$$\begin{pmatrix} B_1^{l+1} \\ \vdots \\ B_i^{l+1} \\ \vdots \\ B_n^{l+1} \end{pmatrix} += \begin{pmatrix} F_{1j}B_j^l \\ \vdots \\ F_{ij}B_j^l \\ \vdots \\ F_{nj}B_j^l \end{pmatrix}. \tag{9}$$

Most of the light energy is initially concentrated in a small number of patches: the light sources. To take advantage of this, the patches can be sorted by brightness and the brightest ones shot first. Then, only patches with unshot brightness need be used during the iteration and dark patches can be skipped, resulting in faster convergence. Another advantage of shooting over gathering is that each iteration transfers light from a bright patch to the many patches in the environment, so as the calculation proceeds the image is "progressively refined."

The matrix iteration can be written using changes in brightness rather than absolute brightness:

$$\Delta B_i^{l+1} = \rho_i \sum_{j \neq i}^{n} F_{ij} \Delta B_j^l. \tag{10}$$

where $\Delta B_i^l = B_i^l - B_i^{l-1}$ and $B_i^l = \sum_j^l \Delta B_i^j$. This way of organizing the iteration simplifies testing for convergence. It also is makes it convenient to sort patches based on unshot brightness.

## Review of N-Body Formulations

The radiosity 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 $n$ particles exerts a force on all $n-1$ other particles. The straightforward solution entails calculating all $n(n-1)/2$ pairwise interactions. Appel (Appel 1985), Barnes & Hut (Barnes and Hut 1986), and Greengard & Rokhlin (Greengard 1988) have each devised algorithms that compute all the forces on a particle to within a given precision in less than quadratic time. All three methods recognize the importance of hierarchical data structures for clustering particles.

Appel was the first to recognize that the forces acting on a particle need only be computed to within some numerical tolerance, and that the interaction between two clusters whose separation significantly exceeded their sizes could be reduced to a single resultant force within the allowable error. If two clusters with $m$ particles interact directly, then the force between the two clusters can be computed in constant time rather than in $m(m-1)/2$ time. He devised a simple top-down divide and conquer algorithm for computing these forces in $O(n \log n)$ time (Appel 1985). Barnes and Hut developed an adaptive N-body algorithm that formed clusters by building an octtree from the bottom up. The leaf nodes contained single particles and the interior nodes were formed by merging eight-child clusters (Barnes and Hut 1986). The algorithm makes two passes, computing total mass and center of mass while sweeping upwards, and distributing forces in a downwards traversal of the tree. Greengard and Rokhlin devised an $O(n)$ 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 achieves linear time by estimating the potential for each particle in some constant number of terms. There exist terms for each of the particle's nearest neighbors plus a $p$-term multipole expansion of the potential from the remaining particles in the system.

Both Appel and Barnes & Hut claimed, but did not prove, that their algorithms ran in $O(n \log n)$ time. More recently, Esselink has analyzed Appel's algorithm showing that time needed to calculate the forces takes $O(n)$ time (Esselink 1989). The observed $O(n \log n)$ running time is a consequence of the preprocessing time required to build the hierarchical data structures.

## Relationship Between the N-Body and Radiosity Problem

The radiosity problem shares many similarities with the N-body problem. In the N-body problem each body exerts a force on every other body resulting in $n(n-1)/2$ pairs of interactions. Similarly, in the radiosity problem each patch may scatter light to every other patch resulting in $n^2$ interactions. In the N-body problem, the forces obey Newton's third law and are therefore equal but opposite; in the radiosity problem, there is no requirement that the amount of light transferred between two surfaces is the same in both directions. However, the reciprocity principle does relate the form factors in both directions. 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 $1/r^2$. For this reason, the hierarchical clustering ideas like those used for the N-body problem apply to the radiosity problem.

There are differences, however. The N-body algorithms begin with $n$ particles and must cluster them into larger groups. The radiosity algorithm usually begins with a few large patches and must subdivide them into elements. In the classical approach, the minimum size of elements needed to subdivide a patch is normally given as an input parameter $A_\epsilon$, and all patches are divided uniformly into elements with the given area. As will be seen here, an adaptive algorithm can be used to subdivide large polygons into $n$ smaller elements. This is done by recursively dividing the large polygon. The resulting tree decomposition is very similar to the hierarchical clusters used in the N-body algorithms. Thus, the subdivision requirement provides a convenient way to build the hierarchy needed for the clustering algorithm to work. The separate problem of building clusters out of individual patches is more difficult, and not dealt with in this paper.

Another difference is whereas the N-body problem is based on a differential equation, the radiosity problem requires the solution of an integral equation. However, as has been described, the integral equation arising from the radiosity problem can be solved efficiently using iterative techniques. Each iteration is similar to one time step of the force calculation in the N-body problem, but multiple iterations need to be performed before the solution converges. Performing this iteration efficiently requires a slightly more complicated data structure than typically is used with the N-body problem. It should also be mentioned that while the N-body problem requires the calculations of forces over time because the bodies accelerate and move to new positions, radiosity calculations are normally done for static environments with constant form factors (although see (Chen 1990)).

Finally, the N-body algorithms mentioned above all take advantage of linear superposition, which states that the potential due to a cluster of particles is the sum of the potentials of the individual particles. The physics of radiosity differs. In radiosity problems, an intervening opaque surface can block the transport of light between two other surfaces, which makes the system non-linear. There is no such shielding of gravitational or electromagnetic interactions in the N-body problem precisely because superposition applies. Occlusion thereby introduces an additional cost to the radiosity problem. Nevertheless, since occlusion never increases the amount of light transported between two patches, the form factors between two patches can be still be bounded from above in spite of occlusion. Although in this paper we ignore the problems caused by occlusion, the basic ideas reported here can be extended to handle this case.

**Outline of the Hierarchical Radiosity Algorithm**

Our algorithm for each step in the radiosity problem is summarized below.

• Step 1: Form Factor Computation.

For all pairwise combinations of the $k$ input patches, simultaneously subdivide each patch recursively with respect to the other patch and build a hierarchical representation of each patch's subdivision. During the subdivision process record the interactions between patches and build the form factor matrix. The hierarchical data structure corresponding to each patch subdivision occupies $O(n)$ space. Also, as will be shown, the number of interactions between two patches is $O(n)$, so the total running time of this step is $O(n)$ and the total storage needed for the trees and the interactions lists is also $O(n)$.

• Step 2: Transport of Radiosities.

Solve the simultaneous linear equations for the patch brightnesses. This is done using iterative methods just like shooting or gathering. Since the matrix of form factors has $O(n)$ terms, each complete iteration takes only $O(n)$ time.

**Hierarchical Subdivision**

The first step in a radiosity algorithm builds the hierarchies and computes the form factor matrix. This is done recursively using `Refine` that has as inputs two patches and error tolerances.

```
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` says that whenever the estimated form factors between two patches turn out to be smaller than some predefined $F_\epsilon$ (`Feps` in the program), then allow the two patches to interact directly and terminate the subdivision. `Link` records the fact that two patches interact at a particular level in the tree. However, if either of the form factors is larger than $F_\epsilon$, then split whichever patch has the larger form factor, and recursively refine the new patches. It should be mentioned that if $F_\epsilon$ is set equal to 0, then each node will always be subdivided until its area is less than the area limit $A_\epsilon$ (`Aeps` in the program). The resulting set of interactions will be equivalent to the classical $O(n^2)$ subdivision method.

`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 differential form factor is defined to be the solid angle that a patch subtends with respect to a point. This can be estimated by calculating the solid angle subtended by a disk with cross sectional area equal to the surface area of the patch (Wallace et al. 1989). If the patches were not planar, then the solid angle could be estimated by circumscribing a bounding sphere around each patch and estimating the solid angle subtended by each sphere with respect to the others center as shown in Figure (3).

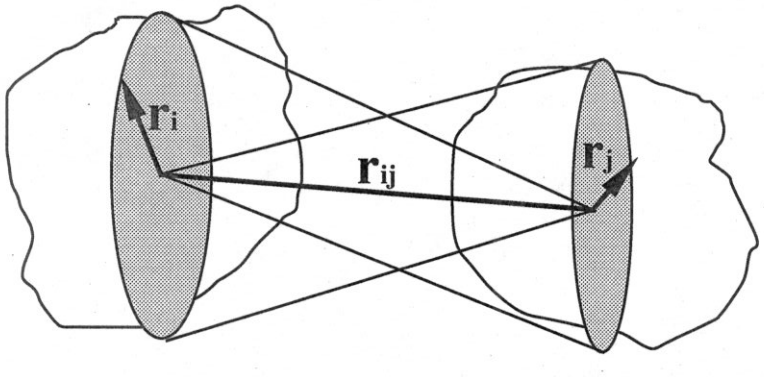

*Figure 3: Method used to Estimate Form Factor*

`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 parent fields `nw`, `ne`, `sw`, and `se`. `Subdiv` returns *false* if the patch cannot be split, which occurs if the area of the patch is smaller than some absolute predetermined area $A_\epsilon$. If subdivision is not possible, we force the two patches to interact and use `Link` to record this information. Note that the `Refine` procedure is invoked for all pairs of input patches, therefore each patch is refined multiple times — once for every other input patch. Thus, the actual subdivision of a patch may have been performed previously when it was refined versus another patch. If the actual subdivision has already occurred, `Subdiv` simply returns *true* and need do no other work.

This subdivision technique will work in principle for any surface that can subdivided into smaller pieces. Our decision to use planar quadrilaterals and subdivide them into fourths is not crucial to the algorithm; other patch geometries (e.g. triangles or bicubic patches) or subdivision techniques (e.g. halving with a $k$-D-tree) would also work.

The combination of all the interactions between nodes in the trees built by `Refine` represents the part of the form factor matrix corresponding to the interactions between the two original parent patches. The number of rows and columns in the form factor matrix is equal to the number of leaves in the trees. The proper union of all the leaf nodes from a given tree exactly covers the original patch represented by the root of the tree. Interactions between leaf nodes in the tree would correspond to single entries in the standard form factor matrix. Interactions between nodes at higher levels correspond to a rectangular block of entries in the form factor matrix. Thus, the interactions between nodes in the tree can be interpreted as a coalescing of the standard form factor matrix into a set of disjoint rectangular blocks.

An example of a tree that might be produced by `Refine` and its associated form factor matrix is shown in Figure (4). For simplicity, the example illustrates the interactions between two hypothetical 1D patches. Since the patches are 1D, the subdivision is represented by a binary tree instead of quadtree. These two binary trees are drawn on the edges of the form factor matrix. Labelled arcs that connect nodes in the two trees correspond to interactions represented by blocks in the form factor matrix with the same label. The figure is drawn so that the left side and top side of each block has the same size and position as the two patches in the binary tree connected by the arc corresponding to that block. The diagonal blocks are all zero because patches are assumed not to interact with themselves. Other blocks are non-zero and contain a single number which is the form factor between the two patches at that level in the tree. Notice that the size of the block depends on what level the patches reside in the tree.

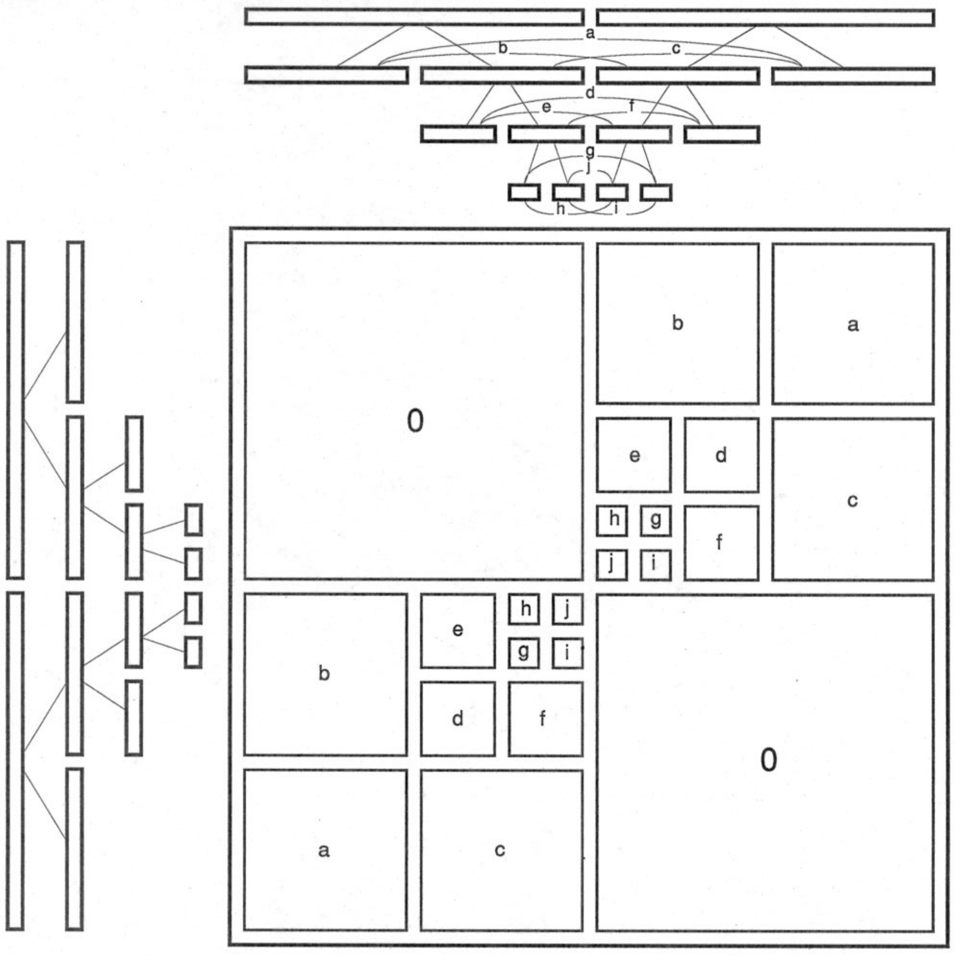

*Figure 4: The Block Form Factor Matrix for a Particular Binary Tree Example*

## Analysis of the Hierarchical Subdivision

The key result of this preprocessing is that each block in the form factor matrix has approximately the same value and error, and that there are at most a linear number of blocks.

To see that each interaction has the same error we show that the termination criteria also places an upper bound on the error associated with the form factor between the two interacting patches. In the N-body problem, the error in treating the potential of a cluster as a single particle varies as $(r/R)^2$, where $r$ is the radius of the cluster, and $R$ is the distance between the two clusters. This error analysis is based on the multipole expansion of the force due to a cluster (Appel 1985; Greengard 1988). In the radiosity problem, the error introduced by using a single differential form factor to approximate the interaction between the two patches also varies as $(r/R)^2$. This can be verified by comparing the differential form factor to the form factor from a point to a disk of radius $r$, where the distance from the point to the center is $R$. The form factor to a disk is equal to

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

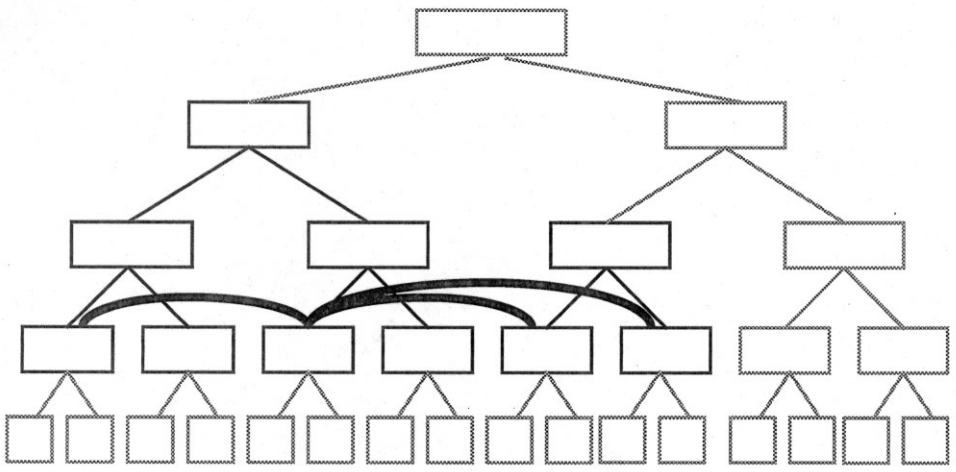

*Figure 5: Interactions Within a Binary Tree*

This differs from the differential form factor only in the higher order terms. Thus, the magnitude of the form factor, as well as the error due to the finiteness of the geometry, have both been bounded by $(r/R)^2$. When $F \approx (r/R)^2$ is small, that is, the sizes of the patches are small compared to the distance separating them, then the differential form factor is a good estimate of the true form factor to within the error bound. Note that the size here is relative. Two large patches may interact if they are separated by a large distance, whereas two small patches may interact if separated by small distances. $F_\epsilon$ acts as the error bound in `Refine`.

The second key result of this preprocessing is the construction of a form factor matrix with fewer than $n^2$ blocks. Recall that for any leaf element, all its interactions with other patches is represented by a row in the form factor matrix. This row slices through rect-angular blocks some of whose size may be greater than 1 and which represent interactions between interior nodes higher in the tree. Recall also that the form factor associated with each interaction is approximately $F_\epsilon$. If it had been larger, `Refine` would already have split that node. As stated by Equation (4), the sum of all the form factors from a leaf node must equal 1. Thus, the total number of interactions from a leaf node including all its parents is roughly equal to $1/F_\epsilon$, which is constant. It depends on the precision required, but not on the total number of elements in the system. In effect, $1/F_\epsilon$ sets the resolution of the hemisphere above each patch. It then follows that the total number of blocks in the form factor matrix varies as $O(n)$ and not $n^2$, since each patch interacts with at most a constant number of other patches. This savings is the essential contribution of our method.

The above analysis slightly overestimates the number of independent form factors be-cause interactions of interior nodes are counted more than once. A more precise argument for the number of form factors scaling linearly is based on simple counting. For simplicity, consider the 1D problem of $n$ equally spaced patches along a line. Let us construct a binary tree (not a quadtree) above the patches by merging adjacent contiguous patches recursively. This is shown in Figure (5). 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 $F_\epsilon$ 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. Now consider the interactions of a patch $p$ in the interior of the tree. At any level in the tree, the above rule forbids the patch $p$ from interacting with its immediate sibling or with the closest of its two first cousins, since they occupy positions adjacent to it. Potentially $p$ may need to interact with all the other patches at its level, but most of these interactions are handled by $p$'s ancestors. $p$'s parent is forbidden from interacting with its immediate neighbors by the same criterion, but it may interact with neighbors further away. Thus, $p$'s parent and ancestors handle all the interactions except those to $p$'s parent's immediate neighbors. So $p$ must connect to the children of its parent's immediate neighbors unless they are too close to it. Figure (5) shows a node $p$ and part of a binary tree containing $p$. $p$ makes three connections to nodes at its level. These are the 4 children of the two patches that its parent could not connect to, minus the cousin which is directly next to it. Interactions with any patches further away will have been handled by ancestors.

This argument applies to all levels of the tree, except the top, the bottom, and the sides. There are no direct interactions at the top two levels because every patch is too close to the other patches; at the lowest level there are interactions with nearest neighbors, since there are no children to handle the interactions otherwise. This yields 5 direct interactions on the bottom level of the tree in our example. There are also boundary effects at the sides of the tree, which decrease the total number of interactions and can be ignored for our purposes. The final result is that 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).

Similar arguments can be made with quadtrees or octtrees. For quadtrees, the number of interactions at the leaf nodes is $6^2 - 1 = 35$, and the number at an interior node is $6^2 - (3^2 - 1) = 28$. The quadtree case is similar to the interaction of two parallel quadrilaterals.

Figure (6) shows the quadtree subdivision of a pair of perpendicular polygons computed by `Refine`. Also shown in this figure are the actual interactions at each level in the subdivision. This figure shows that each interior patch undergoes a constant number of interactions with other patches regardless of their level in the tree. Large patches that are far apart interact directly, in the same way that small patches near each other in the corner interact directly.

In summary, our hierarchical subdivision method computes the form factor matrix to within a fixed error tolerance automatically. In the process it reorganizes the form factor matrix into blocks; the estimated form factor associated with each block has the same value and error as other blocks. Furthermore, the total number of blocks grows at most linearly in the number of elements.

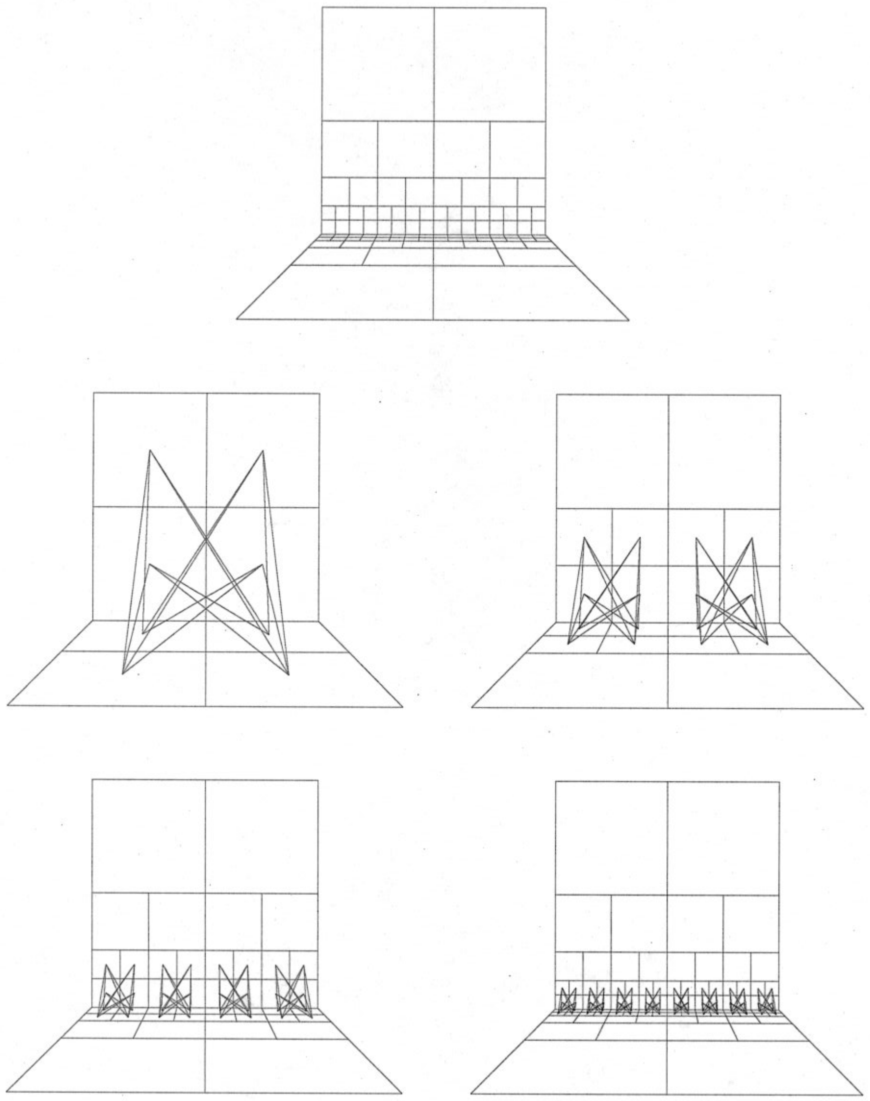

*Figure 6: Interactions Between Two Perpendicular Polygons*

## Iterative Solution

Once the form factors have been determined, the next step is to solve for the radiosities. As seen in the introduction 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 produced by our algorithm is represented with $O(n)$ blocks, each matrix multiplication can be done in linear time. In this section we will show how this is done, and give program fragments that implement the techniques of gathering and shooting.

The block matrix multiply can be made efficient because the blocks representing the matrix connect nodes in a tree representing the vector. The leaves of the tree represent elements of the vector, and the interior nodes in the tree correspond to recursively merging elements of the vector together.

The matrix multiplication proceeds in three steps. The first step computes a value for each interior node in the tree. This value is equal to the average value of all the elements of the vector comprising that node. This calculation can be done in a single depth-first traversal of the tree propagating averages upward. To compute the average for each node, the averages of its children are computed, and then the average of the node is set to the average of its children's values. The second step multiplies the value of each block times the appropriate average value of the source node and adds the result to the appropriate destination node in the tree. For each block this takes constant time. Finally, the third step distributes the results from the interior nodes to the elements of the vector. This is done by another depth first traversal of the tree which propagates values values down to the leaf nodes. Each interior node adds its accumulated value to its children's values, recursively. The running time of this algorithm is easy to analyze. The upwards and downwards traverals of the trees can be done in $n$ time. It also only takes $O(n)$ time to perform the block multiplies because there are $O(n)$ blocks. By using this algorithm, the result is that each iteration can be done in linear time.

There is one caveat to this method when used to perform an iterative matrix inversion. The broadcasting of the output must be deferred until the entire matrix multiplication has been completed, which requires allocating storage for two vectors. Also, iterations often converge faster when updated in place. Our method would permit this if additional passes were made through the tree to update the vector and recompute its averages, but we have not found this necessary.

It may be instructive to consider the physics of radiosity instead of the mathematics of general matrix iteration. The first step computes the brightness of the interior patches by averaging the brightnesses of their children weighted by their relative areas, and the second step multiplies individual blocks. Each block represents a form factor, and multiplying the form factor by the average patch brightness corresponds to shooting the energy of that patch to another patch. Finally, after the energy from some number of patches has been transported, each patch contains all the incoming energy which it has gathered. The new brightness of the leaf patches are then computed by summing the incoming energy of all their parents.

## Gathering

The classic Jacobi iteration can be implemented using the following simple recursive procedure. For convenience the iteration is organized around $\Delta B$'s.

```
Gather( Patch *p )
{
    Patch *q;
    float Fpq;
    if( p ) {
        p->dBg = 0.0;
        ForAllElements( q, p->arcs ) {
            Fpq = FormFactor( p, q );
            p->dBg += Fpq * p->rho * q->dBs;
        }
        Gather( p->sw );
        Gather( p->se );
        Gather( p->nw );
        Gather( p->ne );
    }
}
```

This procedure assumes the average brightness of each patch is stored in dBs, the delta brightness that needs to be shot. For leaf patches this is initially set to the amount of emitted light; for interior patches this is set to the average values, as discussed in the last section. Each patch also has diffuse color stored in rho. The delta brightness gathered is stored in dBg, and is computed by receiving energy from all the patches which interact with this patch. This process can then be applied recursively to the subpatches.

At this point, the form factor between the two patches needs to be computed. If the patches are completely visible to each other, then the form factor can be estimated within the allowable error using Equation (2).

## Shooting

The radiosity equation can also be solved by shooting instead of gathering. The first step initializes patch brightnesses. After this, however, all the patches are sorted into a priority queue based on their brightness. Then a patch at a time is taken off the queue, and its energy shot to the other patches with this procedure.

```
Shoot( Patch *p )
{
    Patch *q;
    float Fqp;
    ForAllElements( q, p->arcs ) {
        Fqp = FormFactor( q, p );
        q->dBg += Fqp * q->rho * p->dBs;
    }
    p->dBs = 0.;
    ReSort( p );
}
```

`Shoot` distributes the unshot radiosity from a given patch to all the other patches it interacts with directly. Once all the energy has been shot from the given node, that node is placed at the end of the queue with `ReSort`. This procedure is called repeatedly until the amount of unshot energy in the patch at the head of the queue is below some threshold.

## Results

Figure (7) plots the number of interactions calculated by `Refine` for two simple geometries - two parallel and two perpendicular polygons. The graphs show the actual number of interactions versus the number of potential interactions at a fixed uniform level of discretization. These plots verify the $O(n)$ behavior for parallel polygons. The number of interactions for perpendicular polygons surpisingly, however, goes as $O(\sqrt{n})$. The reason for this is that 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, as in Figure (4). The total number of nodes in such a sideways binary tree will be $O(\sqrt{n})$. In general, the worst case occurs when two equal-sized polygons are near each other and directly facing; fewer form factors will be needed in all other cases. Note that in Figure 7 there are several plateaus; these result because of the all-or-none decision to subdivide or not subdivide.

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 same geometries shown in Figure (7) (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 (8) shows the measured relative error between the computed and the analytical form factors as a function of $F_\epsilon$.

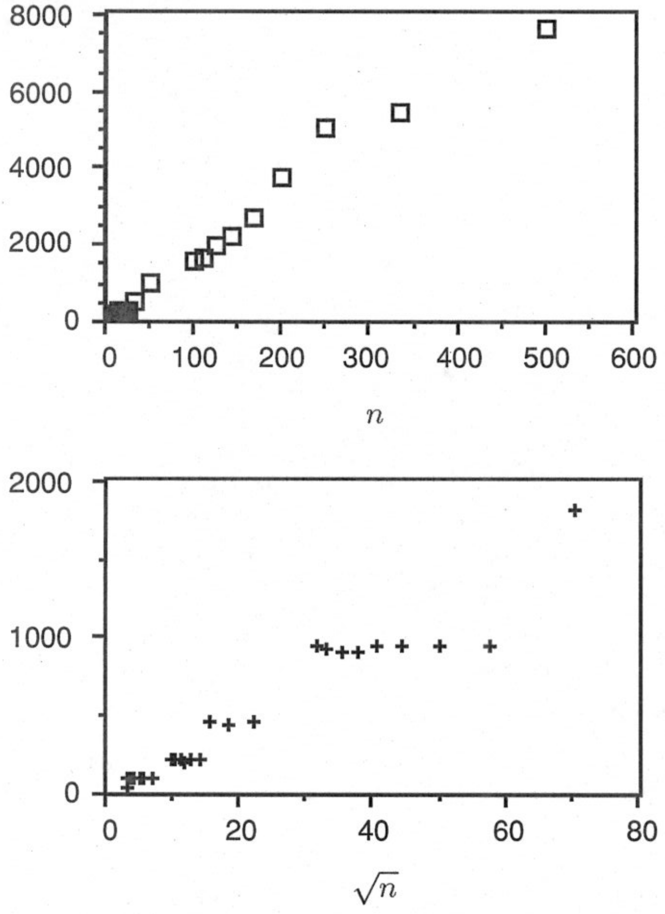

*Figure 7: Number of Interactions between Parallel (above) and Perpendicular (below) Polygons*

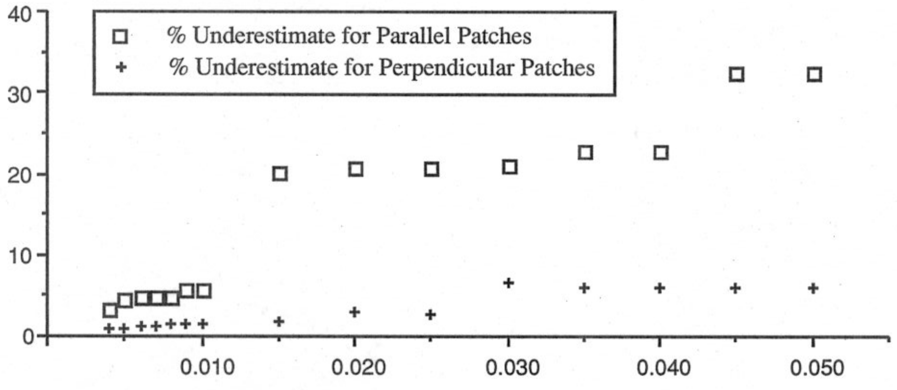

*Figure 8: Measured Relative Percentage Error vs. $F_\epsilon$*

As expected, the actual error in the form factor is proportional to the $F_\epsilon$ given to `Refine` as predicted by the theory.

As a final test of these ideas we modeled the interior of a barren room with 6 polygons of different diffuse colors, following (Goral et al. 1984). For simplicity, the scene was

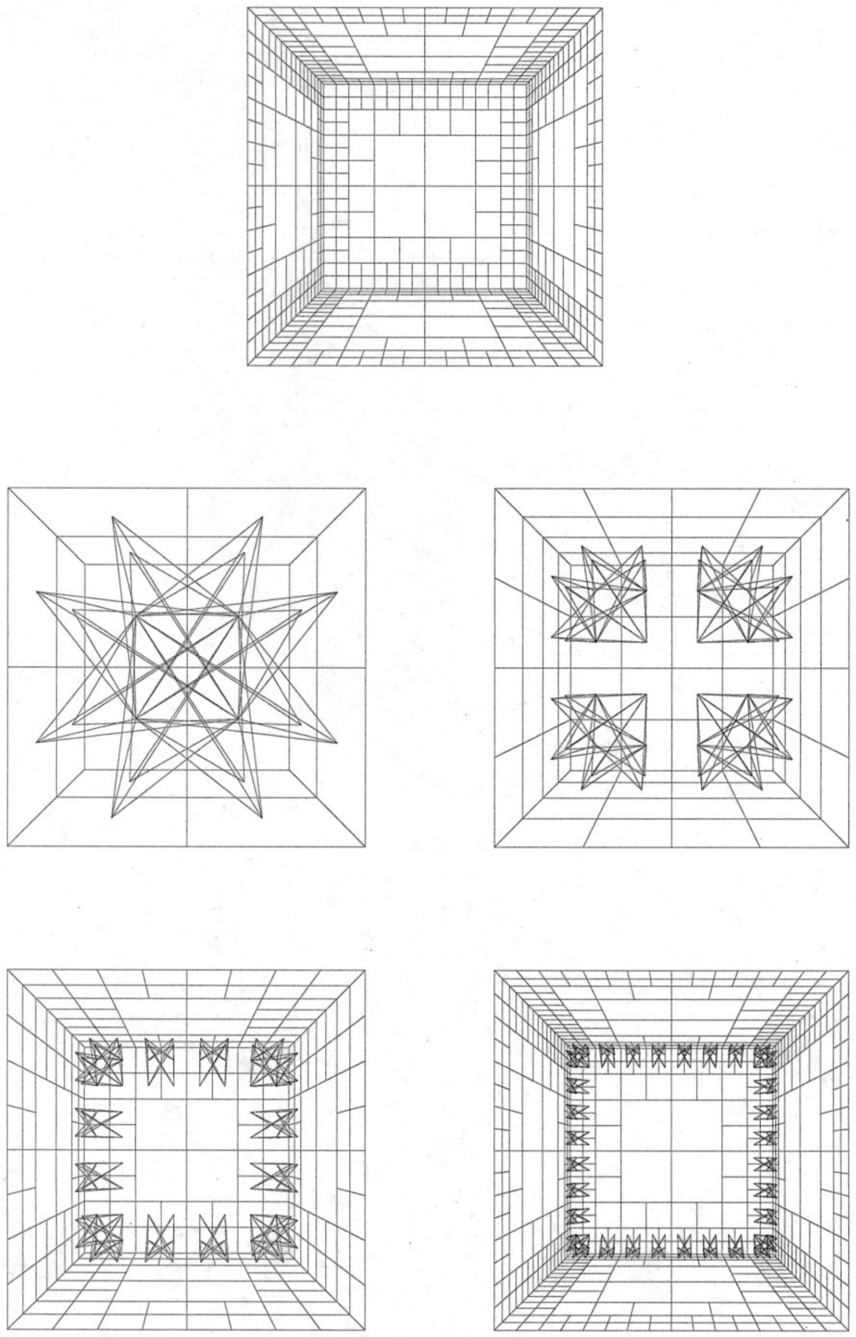

*Figure 9: Hierarchical Subdivision of a Room*

illuminated with a single point light source at the center of the room. Figure (9) shows the hierarchical subdivision produced by `Refine` with a $F_\epsilon = 0.05$ and a $A_\epsilon = (1/16)^2$. The top figure is the quadtree subdivision; each subsequent row shows all interactions of the rear wall at that level of the hierarchy with all walls except the front wall, which is

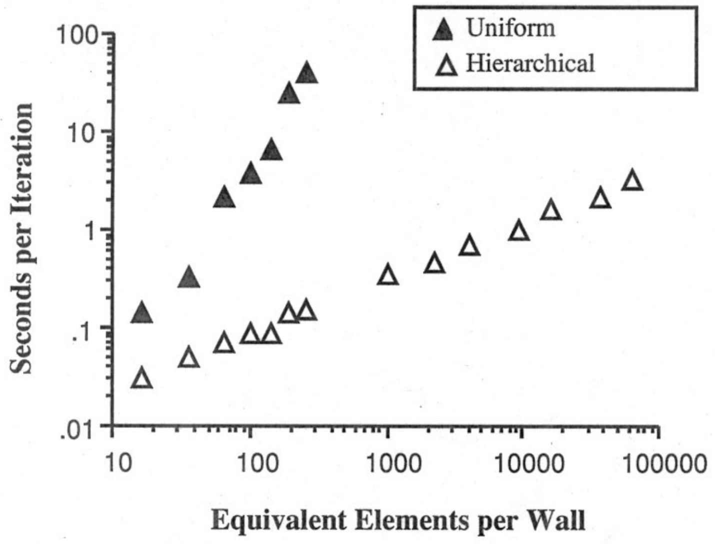

*Figure 10: Comparison of Hierarchical and Uniform Speeds*

not shown. $A_\epsilon$ was chosen so that each wall was divided into a grid of $16 \times 16$ elements. The additional refinement near the walls is a consequence of `Refine` improving the error bounds. Using the uniform subdivision, 983,040 form factors were computed (6 walls interacting pairwise, with $16^2 \times 16^2$ elements interacting for each of the 15 pairs). Using hierarchical subdivision, only 2940 were computed. Of the 15 interactions between walls of the room, 12 are between perpendicular polygons and 3 are between parallel polygons. Thus the number of interactions in this example is dominated by terms that vary as $O(\sqrt{n})$. Each iteration using our hierarchical method took approximately 0.15 seconds, versus 40 seconds for the uniform case. If we merely increased the subdivision to $64 \times 64$ subdivision, our method would run more than thirty thousand times faster than uniform subdivision. Figure (10) shows relative timings for the uniform and hierarchical methods as a function of the desired precision. As previously mentioned, the time spent per iteration is varying as $n^2$ using the uniform method and as $\sqrt{n}$ using the hierarchical method.

## Discussion

We have presented a new algorithm and data structure for radiosity calculations. It is based on an adaptive subdivision algorithm which automatically estimates form factors to a fixed error tolerance. More importantly, this algorithm allows patches in the subdivision to interact at different levels of detail. Large patches which are far away from each other are allowed to interact directly, whereas nearby patches are subdivided to the appropriate size based on their separating distance. This technique reduces the number of interactions from $n^2$ to $O(n)$ and in some cases $O(\sqrt{n})$. This work illustrates the importance of numerical analysis in the design of graphics algorithms. The algorithm is also easy to implement and we think will make global illumination calculations more practical for both realism and interactive applications.

Currently, the most common method for reducing the size of the form factor matrix involves breaking it into a two levels (Cohen et al. 1986). The first level contains the patches, and the second level contains the elements into which each patch is broken. The number of elements that a patch is broken is normally fixed *a priori*, but could be determined adaptively. In the patch-element method, radiosity is always shot from patches to elements, even if they are far away, and the new radiosity of a patch is computed by averaging the brightness of its elements. In our algorithm, radiosity from a patch is shot to patches at the appropriate resolution. Only if the receiving patch is very close will that patch be an element, Also, the patch-element method always treats sources as patches. This may lead to errors when the source is large, both because its brightness is proportional to its area, and because the form factor to a large source may have a large error when it is near to the receiver. In these cases, our algorithm automatically causes the source to be broken into smaller pieces.

The ideas developed in this paper can be used to speed up conventional hemi-cube algorithms. Remember that one way of explaining why our algorithm is linear is that each element subdivides the hemisphere above it into a fixed number of subdivisions independent of the number of other elements, but dependent on the precision required. Once the precision is fixed, the patch size of all other patches in the environment should be chosen so that their projection covers approximately one pixel on the hemi-cube. There is no need to subdivide them further, since all the subdivided pieces would all project onto the same pixel and no new information would be gained.

As described in this paper, we subdivide patches first and then solve the equations for the radiosities. A better implementation would be to perform subdivision on demand using lazy evaluation. One application of this capability would be to successively solve for radiosities by gradually decreasing $F_\epsilon$. Each time $F_\epsilon$ decreases, the patches would be subdivided further. This type of refinement algorithm would allow coarse solutions to be computed very quickly, with few elements, and gradually patches would be adaptively subdivided into finer pieces as the picture was refined. This idea is similar to the multi-grid method.

Another variation that would work effectively with lazy evaluation would be to subdivide based on brightness and angle instead of angle alone. During each iteration, the brightness of each patch increases, so its chances of being subdivided do too. Initially, when the brightness is low, a patch need not be subdivided much. As its brightness increases, so do its interactions with neighbors, so it needs to be computed more accurately and further subdivision should occur. Many of the interactions occur between pairs of patches in corners and along edges. However, these patches tend to be darker than large central areas near the light sources, so subdividing these based on brightness would avoid the needless calculations of their interactions until light was actually transported onto them.

We have begun to extend the algorithm described here to handle environments with occlusion. Occlusion exists in all systems of useful complexity, but violates the principle of linear superposition. The first fact to note is that the true form factor between two patches in the presence of occlusion can never be greater than the form factor estimate used in this paper. This is because intervening occluding surfaces can only decrease the percentage of light that is transported between two surfaces. Thus, the form factor estimate sets an upper bound on the form factor; the true form factor lies between 0 and this bound. This means that the methods used in this paper can still be used to estimate the appropriate resolution of two interacting patches. Once this is done, the exact form factor can be computed using ray tracing to test for visibility. Since the form factor for all interactions has approximately the same magnitude, it is also reasonable to perform the same number of visibility tests per interaction. This means that the total number of visibility probes varies as the number of interactions, which is linear. Each probe, however, could require $n$ intersection tests against other patches, which increases the total running time of the algorithm. These results will be reported in a companion paper.

**Acknowledgements**

The authors wish to thank Andrew Appel, Toby Orloff and Jeffrey Posdamer.

## 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

Chen, S.E. (1990) Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system. Computer Graphics 24

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

Esselink, E. (1989) Computing Science Note KE5-1, Department of Computer Science, University of Groningen

Faddeev, D.K., Faddeeva, V.N. (1963) Computational methods of linear algebra. W.H. Freeman and Co., San Francisco

Goral, C.M., Torrance, K.E., Greenberg, D.P., Battaile, B. (1984) Modeling the interaction of light between diffuse surfaces. Computer Graphics 18(3), 213-222
Greengard, L. (1988) The rapid evaluation of potential fields in particle systems. MIT Press, Cambridge, MA
Greenberg, D.P. (1989) Introduction to radiosity. SIGGRAPH '90 Course Notes
Hottel, H.C., Sarofim, A.F. (1967) Radiative transfer. McGraw-Hill, New York
Kajiya, J.T. (1986) The rendering equation. Computer Graphics 20(4), 143-150
Maxwell, G., Bailey, M.J., Goldschmidt, V.W. (1986) Calculations of radiation configuration factor using ray tracing. Computer Aided Design 18(7), 371-379
Siegel, R., Howell, J.R. (1981) Thermal radiation heat transfer. Hemisphere Publishing Co., Washington, DC
Sillion, F., Peuch, C. (1989) A general two-pass method for integrating specular and diffuse reflection. Computer Graphics 23(3), 335-344
Sparrow, E.M., Cess, R.D. (1978) Radiation heat transfer. Hemisphere Publishing Co., Washington, DC
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