Iterative Methods For Fast Radiosity Solutions

Gladimir V. Guimaraes Baranoski[*] Randall Bramley[†] Peter Shirley[‡]

April 1995

Abstract

In applications involving the radiosity method such as computer animation, time is a crucial factor. This report shows that iterative methods which converge in a smaller number of iterations do not necessarily solve the radiosity system of linear equations in a faster way. For high average reflectance environments is introduced a method that converges faster than any standard method presented in the computer graphics literature so far.

CR Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: General; I.3.6 [**Computer Graphics**]: Methodology and Techniques; I.3.7 [**Computer Graphics**]: Three Dimensional Graphics and Realism.

Key Words and Phrases: iterative methods, linear systems, radiosity.

1 Introduction

In radiosity algorithms, the average radiance of n Lambertian patches is approximated by solving a linear system with n unknowns. When n is small (i.e. fewer than thousands of patches), general matrix methods like Gauss-Siedel can be used where the explicit $n \times n$ matrix can be pre-computed and stored [8]. When n is large, radiosity-specific methods such as progressive refinement and overshooting are used where the matrix rows or elements are recomputed whenever they are needed [7]. When n is very large (i.e. hundreds of thousands of patches), then stochastic techniques can avoid computing the n^2 elements of the matrix [20]. In this paper we will examine the merits of various general matrix methods.

In applications where n is small enough to store the entire matrix in main memory, general matrix techniques will be faster than radiosity-specific methods [1]. For "massing studies" [14] the lighting can be examined on simple geometric approximations of the environment being designed, and n can be very small. When the color scheme and lighting is being designed, the computationally expensive part (form factors) of the matrix in the linear system can be reused as the material properties are changed. For these applications the fastest possible general matrix solution is desirable.

[*]Supported by the Brazilian National Council of Research - CNPq. Email: gbaranos@cs.indiana.edu

[†]Email: bramley@cs.indiana.edu

[‡]Email: shirley@graphics.cornell.edu

[1]Radiosity-specific methods will initially converge faster because they can begin iterations before computing the entire matrix. If general matrix techniques are modified to gradually construct the matrix during initial iterations, then this advantage of progressive method goes away.

In this paper we attempt to find the fastest possible general matrix solution, and provide some insight into the important characteristics of the linear systems that arise in radiosity applications. We are careful to search for solution methods that converge in small amounts of time, as opposed to a small number of iterations. For this discussion we assume a conventional RISC architecture, where coherent memory access is vital.

Initially we describe the radiosity system of linear equations. In section 2 we introduce some mathematical concepts related to the application of the techniques described in section 3, namely the Conjugate Gradient and the Chebyshev methods. In section 4 we provide the standards used here to compare the iterative methods for solving the radiosity problem. Finally we present our results and discuss the effects of reflectivity and occlusion on the performance of the iterative methods.

1.1 Radiosity System of Linear Equations.

Assuming an environment divided into n patches, the total spectral radiant power leaving a patch depends on the spectral radiant power emitted by the patch plus the spectral radiant power that is reflected. The spectral radiant power depends in turn on the total spectral radiant power leaving the other patches in the environment. The following system of equations represents the process of spectral radiant power transfer:

$$\Phi_j = \Phi_j^E + \rho_j \sum_{i=1}^{n} F_{ij} \Phi_i \qquad \text{for} \quad \text{each} \quad \text{j} = 1, 2 \ldots \text{n} \tag{1}$$

where:

Φ_j = total spectral radiant power leaving patch j (watts/nm).
Φ_j^E = spectral radiant power emitted by patch j (watts/nm).
ρ_j = reflectivity of patch j (dimensionless).
F_{ij} = form factor between patch i and patch j (dimensionless).
Φ_i = total spectral radiant power leaving patch i (watts/nm).

The reflectivity ρ_j represents the fraction of incident radiant power which is reflected back to the environment and depends on the material characteristics. Form factor F_{ij} indicates how patch i “sees” patch j, in other words it specifies the fraction of radiant power that leaves patch i and arrives at patch j. Form factors depend on the shape and relative orientation of the patches as well the presence of obstacles between the patches.

Equation (1) holds for each patch in the environment. Therefore to find the total spectral radiant power of each patch we need to solve the linear system:

$$H\Phi = \Phi^E \tag{2}$$

given by:

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{21} & \ldots & -\rho_1 F_{n1} \\ -\rho_2 F_{12} & 1-\rho_2 F_{22} & \ldots & -\rho_2 F_{n2} \\ . & . & \ldots & . \\ . & . & \ldots & . \\ . & . & \ldots & . \\ -\rho_n F_{1n} & -\rho_n F_{2n} & \ldots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ . \\ . \\ . \\ \Phi_n \end{bmatrix} = \begin{bmatrix} \Phi_1^E \\ \Phi_2^E \\ . \\ . \\ . \\ \Phi_n^E \end{bmatrix}$$

Another possibility is to rewrite equation (1) in terms of spectral radiant exitance. From [24], we have:

$$\Phi_j = \pi M_j A_j \tag{3}$$

where:

M_j	=	spectral radiant exitance of patch j (watts/(nm $\times m^2$))
A_j	=	area of patch j (m^2)

Also from [24], we have:

$$\Phi_j^E = \pi E_j A_j \tag{4}$$

where:

E_j	=	spectral irradiance emitted from patch j (watts/($nm \times m^2$)).

Substituting (3) and (4) into equation (1) gives:

$$\pi M_j A_j = \pi E_j A_j + \rho_j \sum_{i=1}^{n} \pi F_{ij} M_i A_i \qquad \text{for} \quad \text{each} \quad \text{j} = 1, 2 \ldots \text{n} \tag{5}$$

Before going further in our derivation we review some identities derived from the mathematical formulation of form factors:

- Reciprocity relationship:

$$A_i F_{ij} = A_j F_{ji}$$

- Summation relationship:

$$\sum_{j=1}^{n} F_{ij} \leq 1 \qquad \text{for} \quad \text{each} \quad \text{i} = 1, 2 \ldots \text{n}$$

If we assume a closed environment the above sums are by definition equal to 1.0. However, in practice, even for closed environments the computed value can be greater than 1.0 depending on the accuracy of the method used to compute the form factors.

- A *planar* or a *convex* patch can not see itself which means that the form factor regarding itself is:

  $$F_{ii} = 0$$

  However there may be situations in which it may be appropriate to consider the form factor F_{ii} of a plane or convex patch different from zero. For example, suppose patches i and j are placed in front of each other and patch j is a perfect reflector (mirror). In this case patch i can "sees" itself and F_{ii} is different of zero.

  In addition a *concave* patch can see itself, then by definition:

  $$F_{ii} \neq 0$$

Applying the form factor reciprocity relationship in the equation (5) and dividing it by A_j we get:

$$\pi M_j = \pi E_j + \rho_j \sum_{i=1}^{n} \pi F_{ji} M_i \qquad \text{for} \quad \text{each} \quad \text{j} = 1, 2 \ldots \text{n} \tag{6}$$

Dividing the above expression by π we finally get the classical expression in terms of spectral radiant exitance which holds for each patch in the environment:

$$M_j = E_j + \rho_j \sum_{i=1}^{n} F_{ji} M_i \qquad \text{for} \quad \text{each} \quad \text{j} = 1, 2 \ldots \text{n} \tag{7}$$

Radiosity, B, is the term used for radiant exitance in the computer graphics literature. Then if we want to determine the radiant exitance, henceforth called radiosity, of each patch we need to solve the linear system:

$$GB = E \tag{8}$$

given by:

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \ldots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \ldots & -\rho_2 F_{2n} \\ . & . & \ldots & . \\ . & . & \ldots & . \\ . & . & \ldots & . \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \ldots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ . \\ . \\ . \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ . \\ . \\ . \\ E_n \end{bmatrix}$$

In general G is a square matrix (n $\times$ n). However it may be rectangular if one applies an hierarchical method [9]. The matrix G is also well conditioned, in other words it is not very sensitive to small perturbations in the input.

The density of G, defined as the number of nonzero entries divided by n^2, depends on the environment complexity. For example an environment divided into a large number of patches (50000) in such way that several patches can not "see" each other may have a matrix 10% dense [7]. On the other hand inside of a sphere all the patches can "see" each other and the corresponding matrix is full even if the sphere is divided into a very large number of curved patches. Notice that the density depends more on the placement of patches in the environment than on the number of patches. A matrix entry is equal to zero when:

- $F_{ij} = 0$ for $i \neq j$ (due to occlusion or patches placed in the same plane);
- $\rho = 0$ (black patches, which does not occur in real environments , although sometimes researchers assign $\rho = 0$ to light sources.).

Considering that $0 \leq \rho < 1$ and assuming that there are only convex or planar patches in the environment ($F_{ii} = 0$) we can say that the matrix G is strictly diagonally dominant since the property:

$$|G_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |G_{ij}| \qquad \text{holds} \quad \text{for} \quad \text{each} \quad \mathrm{j} = 1, 2, \ldots, \mathrm{n}$$

On the other hand if there are concave patches ($F_{ii} \neq 0$) in the environment we can not ensure matrix G dominance of rows. In this case, weak diagonal dominance holds $\mid g_{ii} \mid \geq \sum_{i \neq j} \mid g_{ij} \mid$, with strict inequality holding for at least one i. That should be the case for any radiosity problem unless all $\rho_i = 1$.

1.2 Radiosity Solutions

Direct methods for solving linear systems ($GB = E$) are not suited for large radiosity systems of equations because of the relative high density of the coefficient matrix, and because rapid solutions are needed. The special properties of the coefficient matrix allow the use of iterative methods. These methods are generally of the following form:

```
1    Initialize B and E
2    while (not converged) do
3        choose ΔB
4        update $x \leftarrow B + \Delta B$
5        update $r \leftarrow r + \Delta B$
```

where r corresponds to the residual given by:

$$r = E - GB$$

The iterative methods used to solve the radiosity system of linear equations can be divided into general matrix methods, which update all components of the solution vector on each iteration, and radiosity-specific methods such as progressive refinement and overshooting methods, which update a single component on each iteration.

An example of a general matrix method is the Gauss-Siedel iterative solver:

```
1    for (all $i$)
2        $B_i$ = starting guess
3    while (not converged)
4        for $i = 1, 2, ..., n$
5            $B_i = E_i - \rho_i \sum_{j \neq i}^{n} B_j F_{ij}$
6     display image using $B_i$ as the intensity of the patch $i$.
```

Usually the vector of emitted radiosities is used as the starting guess. This algorithm estimates the radiosity of a patch i by adding its emitted radiosity, and all the radiosity that it reflects (line 5). It uses the most recent estimates of the radiosities of all patches to estimate the radiosity of patch i. The physical interpretation consists of *gathering* the radiosity from other patches to estimate the incoming radiosity. It converges quickly due to the diagonal dominance of the coefficient matrix. After a few passes through the matrix we get the final image. Gauss-Siedel can have $O(n^2)$ memory requirement if the matrix G is dense and its elements are stored.

The progressive refinement approach [7], which was later shown to be equivalent to Southwell's relaxation method [12], allows us to obtain intermediate images and to reduce the storage requirement to $O(n)$ by computing the form factors on the fly:

```
1    for (all $i$)
2        $\nabla B_i = 0$
3        $\Delta B_i = E_i$
4    while (not converged)
5        pick $i$, such that $\Delta B_i * A_i$ is largest
6        $\nabla B_i = \nabla B_i + \Delta B_i$
7        for every other patch $j$
8            $\Delta rad = \rho_j F_{ji} * \Delta B_i$
9            $\Delta B_j = \Delta B_j + \Delta rad$
10       $\Delta B_i = 0$
11       display image using $\Delta B_i + \nabla B_i$ as the intensity of the patch $i$.
```

In the above algorithm ΔB_i represents the amount of "unshot" radiosity and ∇B_i represents the amount of "shot" radiosity ($\nabla B_i = B_i - \Delta B_i$). More formally ΔB_i represents the residual **r** and ∇B_i represents the vector of unknowns that we are solving for.

During one iteration, the patch with the largest "energy" to shoot, i.e. largest $\Delta B_i * A_i$, is chosen and its radiosity is shot through the environment. So in this case the physical interpretation consists of *shooting* instead of *gathering*. As a result the other patches may receive some new radiosity Δrad. The intermediate images obtained in the early steps tend to be dim, so an ambient term is added [7]. As the iteration process goes on this ambient term is reduced until the final image is obtained. The ambient term is a weighted sum of the unshot radiosities of all patches in the environment:

$$Ambient = \frac{R}{\sum_{i=1}^{n} A_i} \sum_{i=1}^{n} \Delta B_i Ai \tag{9}$$

where A_i corresponds to the area of patch i and R corresponds to the overall interreflection factor R , which is given by:

$$R = \frac{1}{1 - \rho_{avg}} \tag{10}$$

The average reflectivity ρ_{avg} of the environment is a weighted average of the patches reflectivities:

$$\rho_{avg} = \frac{\sum_{i=1}^{n} \rho_i A_i}{\sum_{i=1}^{i=n} A_i} \tag{11}$$

In the original progressive refinement approach a patch may need to shoot radiosity multiple times. Overshooting methods based on overrelaxation have been introduced to solve this problem [10] [12] [22] [27]. The main idea consists of shooting the current radiosity of the patch plus an estimate that accounts for radiosity that will return due to the reflection by other patches. This approach can reduce the overall number of iterations required. One such algorithm was proposed by Feda et al. [10]:

```
1     for (all $i$)
2         $\nabla B_i = 0$
3         $\Delta B_i = E_i$
4     determine initial $Ambient$
5     while (not converged)
6         for (each patch $i$)
7             $\Delta B_i^{'} = min(\Delta B_i + R_i * Ambient, \sum \Delta B_j * A_j / A_i)$
8         select patch $i$, such that $\mid \Delta B_i^{'} * A_i \mid$ is largest
9         $\nabla B_i = \nabla B_i + \Delta B_i^{'}$
10        for (every other patch $j$)
11            $\Delta rad = \Delta B_i^{'} * \rho_j F_{ji}$
12            $\Delta B_j = \Delta B_j + \Delta rad$
13        $\Delta B_i = -\rho_i * Ambient$
14        determine new $Ambient$
15        display image using $\nabla B_i + \Delta B_i$ as the intensity of the patch $i$.
```

Feda uses the ambient term not only for display purposes but also as the overshooting amount. As was mentioned earlier several methods have been proposed to implement overshooting. For a comparison of four overshooting methods (Feda's, Gortler's, Shao's and Xu's) the reader should refer to [27].

1.3 Proposed Approach

Radiosity-specific methods allow us to have a rough preview of the overall illumination after few steps of iteration. However since few patches contribute to the overall illumination in the early stages, detailed illumination effects such as color bleeding are lost as mentioned by Michael Cohen et. al. [7]. Radiosity-specific methods have to iterate until the desired convergence is reached to avoid that loss. This means that even though they do not need to store all the form factors, they still have to compute all of them, possibly many times.

Suppose we want to generate several similar images to produce a computer animated sequence. Since there is some significant coherence from frame to frame we do not need to have a preview of all frames. In fact the first images can be used as a preview of the whole sequence. In this case the preview advantage of the radiosity-specific methods does not hold anymore. In this case we need to generate several similar images as fast as possible, i.e., to solve the linear systems in the shortest possible time. We are going to show that methods that converge in a smaller number of iterations, namely the radiosity-specific methods, are not necessarily faster than some general matrix methods.

Usually the form factor are computed on the fly when one uses a radiosity-specific method. The same approach could be applied if one uses a general matrix method. However in several applications we can approximate a scene with a reasonable number of patches and using techniques such as hierarchical radiosity [17] reduce the storage of the form factors to $O(m)$ in which m is the number of elements of the environment. By storing the form factors we have the additional advantage of avoiding the cost of recomputing them repeatedly. If the geometry of the scene changes from one frame to another we may assume that only a small fraction of the form factors need to be recomputed due to the animation coherence. In this case we can use acceleration techniques such as those presented in [4] [6] [11] to minimize the overhead of recomputing some of the form factors. As a matter of fact there are useful animation applications that do not change the geometry of the environment at all. For example, suppose we want to simulate the incidence of sun light and the corresponding shadows in a building during several parts of the day for architectural purposes. In this case only the vector of emitted radiosities will change.

In addition, for high albedo scenes, due to the characteristics of the coefficient matrix, the iteration process takes a longer time to converge. For example when the overall reflectivities of the environment approach to 1.0 the spectral radius of the iteration matrix (for a stationary method like Gauss-Siedel) also approaches to 1.0 and the linear solvers converge slowly. Here we test the performance of the Conjugate Gradient Method (previously mentioned in the graphics literature by Heckbert and Winget [18]) in those situations. Finally we are going to introduce a related method, namely the Chebyshev method, that can converge in those situations faster than any standard method already mentioned in the graphics literature so far.

2. Analysis in Matrix Terms

Before going over the methods we need to present some important linear algebra topics: eigenvalues estimates and matrix symmetrization. Both are directly related to the methods that we use here.

2.1. Eigenvalues Estimates

The characteristic polynomial of a square $(n \times n)$ matrix G is given by:

$$p(\lambda) = det(G - \lambda I) \tag{12}$$

where I represents the identity matrix.

The zeros of $p(\lambda)$ are called *eigenvalues* or characteristic values of the matrix G. If $\mathbf{x} \neq 0$ is such that $(G - \lambda I)\mathbf{x} = 0$ holds, then $\mathbf{x}$ is called an *eigenvector* or characteristic vector of G corresponding to the eigenvalue λ.

Usually to calculate the eigenvalues of a matrix G requires more computation than that required to solve the corresponding linear system. However we can obtain relatively inexpensive estimates of the eigenvalues using the Gerschgorin Circle Theorem [5].

Gerschgorin Circle Theorem says that the eigenvalues of G are contained within the union of n circles S_i, which have center g_{ii} and radius $\sum_{j\neq i}^{n} |g_{ij}|$ and are defined by:

$$S_i = \{z \in C \mid |z - g_{ii}| \leq \sum_{j\neq i}^{n} |g_{ij}\} \tag{13}$$

where $\mathbf{C}$ is the complex plane. The union of any k of these circles that do not intersect the remaining (n-k) must contain precisely k (counting multiplicities) of the eigenvalues.

In the case of the radiosity matrix, if we assume an environment formed by planar or convex surfaces ($F_{ii} = 0$) then all the main diagonal entries will be equal to 1.0. So the centers of the circles are also equal to 1.0. In addition if we assume a closed environment ($\sum_{j=1}^{n} F_{ij} = 1$), then the radius is given by ρ_i. In practice if the numerical method used to compute the form factors does not provide exact results (Appendix A), then the radius is in fact given by $\rho_i(1 \pm \delta)$, where δ corresponds to the numerical error associated with the summation of form factors of patch i.

2.2 Symmetrization

The transpose of an $n \times n$ matrix $G = g(ij)$ is the matrix $G^t = g(ji)$. A square $n \times n$ matrix G is said to be symmetric if $G = G^t$.

The radiosity matrix G can be made symmetric by scaling its rows:

$$G^s = diag(\vec{v}) * G, \tag{14}$$

where diag($\vec{v}$) is a diagonal matrix in which the diagonal entry v_i is the quotient of the area and the reflectivity of patch i.

The matrix G^s is also positive definite, i.e. all of its eigenvalues are positive [5]. This makes it suitable for the application of the Conjugate Gradient Method.

3. Statement of Standard Algorithms

3.1 The Successive Overrelaxation Method

The Successive Overrelaxation Method (SOR) consists of applying extrapolation to the Gauss-Siedel Method [16]. This extrapolation takes the form of a weighted average between the previous iterate and the computed Gauss-Siedel iterate, for each component. The physical interpretation is that instead of gathering the correct amount of radiosity, an extrapolation factor w is used to take into account steps that will occur later on the process.

The structure of the SOR algorithm is similar to that of the Gauss-Siedel algorithm. If $w = 1.0$ then the SOR Method becomes the Gauss-Siedel Method. Its algorithm can be obtained by replacing line 5 of the algorithm presented in section 1.2 by:

$$B_i = (1 - w)B_i + w(E_i - \rho_i \sum_{j=1, j \neq i}^{n} B_j F_{ij})$$

The major difficulty of this method is to choose the best value of w which gives the fastest convergence. This factor depends on the problem characteristics. However the idea of overrelaxation is exploited in the context of progressive refinement methods and it can be thought of as overshooting.

3.2 Conjugate Gradient Method

SOR is a linear stationary method, which implicitly updates solutions by $x_{k+1} = Tx_k + \tilde{t}$, where T is an iteration matrix. Nonstationary methods have an implicit iteration matrix T_k which changes on each iteration. When combined with a preconditioner [3] the nonstationary method of Conjugate Gradients (CG) [19] is probably the most powerful and effective iterative method for symmetric positive definite matrices occurring in engineering computations. The CG algorithm implicitly builds up information about the spectrum of the coefficient matrix G as its iterations proceed, providing a computationally superlinear convergence rate. CG is a residual polynomial based method, which means that $r_k = P_k(G)r_0$, where r_k is the k-th residual vector and P_k is a k-th degree polynomial satisfying $P_k = 1.$. Furthermore, CG is optimal in the sense that P_k minimizes the error vector $e_k = x_k - x^*$ in the G-norm given by $\|e_k\|_G^2 = e_k^T G e_k$. The minimization is over a space of dimension k, so after n steps at most the solution is found. CG requires some number of iterations (depending on the eigenvalue distribution of G) before the accumulated information allows a rapid decrease in the residual norm. Although this is highly effective for systems requiring relatively accurate solutions, it is not *a priori* clear that CG is optimal for radiosity systems, which require low accuracy solutions with only a few sweeps through the matrix G.

Before presenting the algorithm, it is useful to understand where some of the formulas of the algorithm come from. CG can be viewed as a minimization method in which solving $Gx = e$ is equivalent to minimizing $\phi(x) = \frac{1}{2}x^T Gx - x^T e$. On each step it generates a search direction d_k, then a step size α_k so that the updated solution vector $x_{k+1} = x_k + \alpha_k d_k$ will have minimal residual over all α:

$$r_k = r_{k-1} - \alpha_k G d_k \tag{15}$$

or

$$r_k = e - G(x_{k-1} + \alpha_k d_k) \tag{16}$$

This implies:

$$\alpha_k = \frac{d_k^T r_{k-1}}{d_k^T G d_k} \tag{17}$$

To prevent the "zig–zag" effect that causes steepest descent methods to have slow convergence, search directions are choosen to be G-conjugate, i.e. $d_i^T G d_j = 0, \;\; for \;\; i \neq j$. It can also be shown that it suffices to make d_{k+1} G–conjugate to d_k, in which case d_{k+1} is G–conjugate to all previous search directions. It can be shown that choosing α to minimize $\phi(x_{k-1} + \alpha d_k)$ will also make x_k minimize ϕ over all x $\in$ span$[d_1, d_2, \ldots, d_k]$ [13]. The CG algorithm can be implemented using three–term recursions and hence using only three vectors of additional storage.

The radiosity coefficient matrix is not symmetric. So we must perform its symmetrization before applying the Conjugate Gradient Method, i.e. $G^s = \vec{v}G$. The algorithm used to implement CG in a radiosity context is the following:

```
0     compute $G^s = V * G$
1     for (all i)
2           $B_i^{(0)}$ = starting guess
3     compute $\Delta B^{(0)} = E - G^s * B^{(0)}$
4     for (each i)
5           $D_i = \Delta B_i$
6           $\varphi = \varphi + \Delta B_i * \Delta B_i$
7     while (not converged)
8           for (each i)
9              for (each j)
10                $W_i = W_i + G^s_{ij} * D_j$
11           for (each i)
12              $temp = temp + W_i * D_i$
13           $\alpha = \varphi / temp$
```

```
14          for (each i)
15              $B_i = B_i + \alpha * D_i$
16              $\Delta B_i = \Delta B_i - \alpha * W_i$
17          for (each i)
18              $temp = temp + \Delta B_i * \Delta B_i$
19          $\beta = temp/\varphi$
20          $\varphi = temp$
21          for (each i)
22              $D_i = \Delta B_i + \beta * D_i$
```

Although CG requires the coefficient matrix to be symmetric, because it enters in only through matrix-vector products we need not explicitly form the matrix G^s but can instead compute its action on a vector by multiplying by G and then the diagonal matrix V. In the above algorithm ΔB represents the residual, V represents the vector of quotients of the reflectivity and the area of all patches and D represents the vector of search directions. Usually we use the vector of emissivities as our starting guess for the vector of unknowns (radiosities B). However in some cases the use of a starting guess which takes into account the ambient term gives better results. The ambient is computed replacing ΔB_i by E_i in equation (9). The computation of the ambient term does not add a significant cost to the algorithm since we may assume without loss of generality that are there few patches in a given environment that emit light, i.e., few patches i with $E_i \neq 0$. To use this starting guess we replace line 2 of the above algorithm by:

$$B_i^{(0)} = E_i + \rho_i Ambient$$

The speed of convergence of the CG method depends on the overall distribution of eigenvalues of the matrix G. Upper bounds [1] for the number of iterations are often based on the condition number of G. Since G is symmetric and positive definite, its condition number is $\kappa = \frac{\lambda_{max}(G)}{\lambda_{min}(G)}$. The number k of iterations needed to reduce the residual by a factor of ϵ in the G-norm is then bounded by $k \approx 0.5 * \sqrt{\kappa} * \log(2/\epsilon) + 1$. Using the Gerschgorin estimates obtained earlier, this means for a reduction by three orders of magnitude at most $k \approx 0.5 * \sqrt{\kappa} * \log(2/\epsilon) + 1$ iterations are needed. However, radiosity matrices also tend to have eigenvalues tightly clustered around a few values, which greatly reduces the number of iterations needed.

3.3 The Chebyshev Method

The Chebyshev Method is another nonstationary method based on residual polynomials. It is directly applicable to nonsymmetric matrices like the radiosity coefficient matrix. However it requires estimates of the smallest and largest eigenvalues of the corresponding coefficient matrix [26] [3].

The iterative process is characterized by:

$$x_{j+1} = x_j + \Delta x_j \qquad \Delta x_j = (\frac{1}{q_j})(r_j + p_j \Delta x_{j-1}) \tag{18}$$

where the scalars r_j are the residuals and q_j and p_j are the coefficients of the residual polynomials.

To obtain fastest reduction in the residual norm a residual polynomial method needs to select polynomials that quickly go to zero on the spectrum of the coefficient matrix G. For radiosity problems the eigenvalues are all real and positive, so given a knowledge of an interval $[\mu, \nu]$ containing the spectrum of G we want to select polynomials P_k that have their maximum value on $[\mu, \nu]$ minimal over all monic polynomials of degree k. In addition to this "minimax" property, Chebyshev polynomials can be computed using a three term recursion. This last property implies that the Chebyshev iteration can be implemented using only three additional vectors of storage.

The classical algorithm for the Chebyshev method [26] [21] is:

```
1    $x^{(0)} arbitrary$
2    $r^{(0)} = e - Gx^{(0)}$
3    $x^{(1)} = x_{(0)} + \gamma^{-1} r^{(0)}$
4    $r^{(1)} = e - Gx^{(1)}$
5    $\omega^{(0)} = 2/\gamma$
6    $j = 1$
7    **while** (not converged)
8        $\Delta x^{(j)} = \omega^{(j)} r^{(j)} + (\gamma\omega^{(j)} - 1)\Delta x^{(j-1)}$
9        $x^{(j+1)} = x^{(j)} + \Delta x^{(j)}$
10       $r^{(j+1)} = e - Gx^{(j+1)}$
11       $j = j + 1$
```

In the above algorithm:

$$\gamma = \frac{\beta}{\alpha} \tag{19}$$

$$\alpha = \frac{2}{\lambda_{\max} - \lambda_{\min}} \tag{20}$$

$$\beta = \frac{\lambda_{\max} + \lambda_{\min}}{\lambda_{\max} - \lambda_{\min}} \tag{21}$$

$$\omega^{(j)} = (\gamma - \frac{1}{4\alpha^2}\omega^{(j-1)})^{-1} \tag{22}$$

Note that the updates in lines 8-10 are vector updates. In practice, $\lambda_{\max}$ and $\lambda_{\min}$ may be replaced by ν and μ, where $0 < \mu \leq \lambda_{\min} < \lambda_{\max} \leq \nu$. However, the rate of convergence of Chebyshev is maximal when $\mu = \lambda_{\min}$ and $\nu = \lambda_{\max}$, and the method can diverge if, for example, λ_{max} is underestimated by ν. Therefore to implement the Chebyshev Method successfully we need estimates of the extremal eigenvalues of the matrix G.

In the case of the radiosity coefficient matrix the Gerschgorin Disk Theorem (see section 2.1) provides good estimates of the eigenvalues. In this case the extremal eigenvalues may be approximated by $1.0 \pm \rho_{max}$ in which ρ_{max} is the highest reflectivity of any patch in the environment. However our experiments show that increasing the highest reflectivity does not signficantly change the convergence. On the other hand if we increase the overall reflectivity of the environment, e.g. high albedo scenes, then slow convergence results. Our experiments show that using ρ_{avg} to estimate the eigenvalues gives better results for all the cases tested. Therefore our Chebyshev parameters are given by:

$$\nu = 1.0 + \rho_{\text{avg}} \tag{23}$$
$$\mu = 1.0 - \rho_{\text{avg}} \tag{24}$$

The Chebyshev algorithm used in the radiosity context is the following:

```
1    for (each i)
2        $B_i^{(0)}$ = starting guess
3    compute $\Delta B^{(0)} = E - G * B^{(0)}$
4    $\alpha = 2/(\nu - \mu)$
4    $\beta = (\nu + \mu)/(\nu - \mu)$
6    $\gamma = \beta/\alpha$
7    for (each i)
8        $B_i^{(1)} = B_i^{(1)} + \gamma \Delta B_i$
9    compute $\Delta B^{(1)} = E - G * B^{(1)}$
10    $\omega = 4/(\nu + \mu)$.
11    $j = 1$
12    while (not converged)
13        $\omega = (\gamma - \frac{1}{4\alpha^2}\omega)^{-1}$
14        for (each i)
15            $D_i = \omega * \Delta B_i^{(j)} + (\gamma\omega - 1) D_i$
16            $B_i^{(j+1)} = B_i^{(j)} + D_i$
17        compute $\Delta B^{(j+1)} = E - G * B^{(j)}$
18        $j = j + 1$
```

In the above algorithm ΔB represents the residual and D represents the term Δx_i of equation (18). In all the cases tested the use of the starting guess which takes into account the ambient component gave better results than the starting guess which uses only the vector of emissivities. To use this starting guess we replace line 2 of the above algorithm by $B_i^{(0)} = E_i + \rho_i Ambient$, where the ambient term is also computed replacing ΔB_i by E_i in equation (9).

Heuristically, we can compare the CG and Chebyshev methods in terms of their underlying residual polynomials. For CG the polynomials tend to rapidly converge to zero at the extremal eigenvalues, then progressively move towards the interior of the region containing the spectrum, zeroing out eigenvalues as iterations proceed. Chebyshev polynomials tend to uniformly drive all the eigenvalues to zero at the same rate. For solutions requiring high accuracy, the CG algorithm is usually the faster of the two since it implicitly creates the "optimal" polynomial on each step. For the low accuracy solutions needed in radiosity, however, it is more important to quickly reduce the residual corresponding to all the eigenvalues. So we expect the Chebyshev iteration to be faster, which is borne out by the testing results.

4 Standards Applied to the Problem

We compared the following algorithms, using explicitly stored form factors:

- Gauss-Siedel(GS)
- Standard Progressive Refinement(PR)
- Overshooting(FEDA)
- Conjugate Gradient(CG)
- Chebyshev(CHEBY)

The starting guesses were chosen in order to obtain the best possible rate of convergence for each tested algorithm. Consequently the initial error norm is not the same for curves on a single graph.

The starting guess used for Gauss-Siedel and Conjugate Gradient algorithms was the vector of emissivities and for the radiosity-specific methods we used a vector of zeros. For the Chebyshev algorithm we used the starting guess which takes into account the *Ambient* term, i.e. $B_i = E_i + \rho_i Ambient$.

4.1 Performance Measurement

The general matrix methods check the convergence after a complete sweep of the coefficient matrix, i.e. one iteration. The progressive refinement methods perform this check after one relaxation step, i.e. one step of iteration. To make our measurement uniform we count steps of iteration. Then in this context an iteration of a general matrix method corresponds to n steps of iteration, where n is the order of the coefficient matrix.

In order to measure the time we start the clock at the beginning of a cycle of k steps of iteration and stop it after k steps of iteration. For the general matrix methods tested we use k equal to n. We check for k at each step in order to make the timing overhead the same for all methods. In addition all the error norms are computed outside of the timing cycles. The time measurements are given by elapsed CPU time on a SGI Challenge (20-R4400). All the algorithms were implemented using the same software guidelines to avoid differences that could affect the timing.

4.2 Convergence Checking

We use as our stopping criteria the largest unshot energy, i.e. the L_∞ norm of the vector with components r_iA_i, in which r_i represents the residual and A_i the area of patch i, given by:

$$\tau_\infty = \max_{1 \leq i \leq n} |r_i A_i| \tag{25}$$

If τ_∞ is smaller than a given tolerance we stop the iterations. The value assigned to the tolerance depends on how visually close to true solution one wants the final image be. In general it is not necessary to use very low tolerance as in most numerical applications. We used a tolerance equal to 10^{-3}, but present the full convergence histories so that the methods can be compared for larger tolerances.

Besides the L_∞ norm used as the stopping criteria, we also compute three other error norms: the L_2 norm of the residual, RMS_{error} and the percentage of "reflected" radiosity (*error*), defined by Goertler et. al. [12].

The L_2 norm of the residual is given by:

$$\tau_2 = (\sum_{i=1}^{n} r_1^2)^{\frac{1}{2}} \tag{26}$$

The RMS (root mean square) error is also based on an the L_2 norm and consists in calculating the square root of the area weighted mean of the square of individual errors:

$$RMS_{error} = \sqrt{\frac{\sum_{i=1}^{n}(B_i^* - B_i)^2 A_i}{\sum_{i=1}^{n} A_i}} \tag{27}$$

where:
B_i^* = converged radiosity of patch i

The *error* defined by Goertler et. al. is given by:

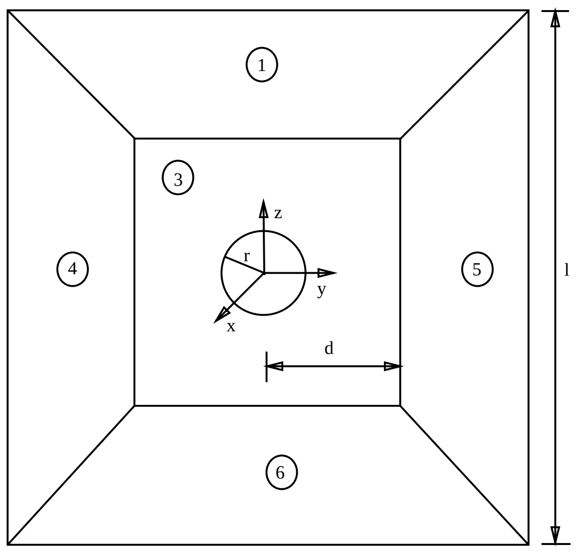
$$error = \frac{\sqrt{\sum_{i=1}^{n}(B_i^* - B_i)^2}}{\sqrt{\sum_{i=1}^{n}(B_i^* - E_i)^2}} \tag{28}$$

The difficulty with the last two error norms is that the converged true solution is not available in practice. For testing purposes we approximate the exact solution with a solution vector obtained by the Gauss-Siedel method, iterated until the residual vector has norm less than 10^{-10}.

4.3 Test Cases

The test model used in our experiments is shown in Figure 1 and consists of a sphere in the middle of a room. The sphere was divided into 128 patches and the faces of the surrounding cube were divided into 144 patches forming a total of 992 patches. The light source corresponds to 16 patches on the center of the "ceiling" of the cube.

Assigning different values for the reflectivities we can have different ρ_{avg} and changing the sphere radius, r=1.0 and r=2.0, we can have have different densities (δ) for the coefficient matrix, 70 % and 53% respectively. Representative cases shown in Table 1. The values used for l and d were 6.0 and 3.0 respectively.



Face 2 is frontal.

r	=	sphere radius.
l	=	length of the faces of the cube.
d	=	distance between the center of the sphere and the faces.

Figure 1: Sketch of the test model.

Table 1: Test cases.

case	$\delta(\%)$	ρ_{avg}
A	53	0.24
B	53	0.46
C	53	0.77
D	53	0.88
E	70	0.78
F	70	0.89

5. Testing Results

Testing was performed to compare the performance of the five algorithms described. In particular, we examine the effect of matrix density (which depends on the amount of occlusion in the environment) and matrix spectrum (which depends on the reflectivities in the scene). In Figures 4-11 note that the error measure for CG can increase initially. Although CG is optimal in reducing the error norm $e_k^T G e_k$ on every step, in practice the vector $e_k = x_k - x^*$ is not available. So increases in the residual norm (which we can measure) do not contradict the theoretical properties of the CG algorithm.

Numerical testing is necessary because most theory about the convergence rates of these methods deals with *asymptotic* convergence. For the low accuracy solutions needed in graphics radiosity problems, oftentimes an adequate solution is available before the asymptotic convergence region is approached.

The Gauss-Seidel and CG methods are *parameter-free*, that is, there are no algorithmic parameters which must be set by the user. Chebyshev requires estimates of the smallest and largest eigenvalues, but for this application those can be set automatically as was done here, by using $1 \pm \rho_{avg}$. The Feda method can be fine-tuned by different choices of the overshooting parameter, in the same way that Gauss-Seidel is generalizable to the SOR method. However, like the SOR method the optimal choice of parameters is unknown except for a few special cases. The version implemented here automatically selects the overshooting parameter.

5.1 Steps of Iterations *vs* Time

In practice we need linear solvers that converge as rapidly as possible. Usually progressive refinement or overshooting converge in fewer steps of iteration than the other three methods. However, counting steps of iteration does not account for the differing amounts and types of work performed on each step. So a method that converges in fewer steps of iteration may in fact require more overall time.

The experiments show that this distinction does occur in radiosity applications. Figure 4 shows that for test case A with $\rho_{avg} = 0.24$ progressive refinement methods converge in fewer steps of iteration for the given tolerance of $tol = 10^{-3}$. However, Figure 5, showing the same convergence history as in Figure 4 but plotted against elapsed CPU time, shows that Gauss-Seidel and Chebyshev methods converge in less CPU time. When the overall reflectivity of the environment is increased, the difference becomes even more noticable, as shown in Figures 6 and 7 for test case B with $\rho_{avg} = 0.46$.

Why does this counter-intuitive result occur? The main reason is the differing amounts of pipelining and data locality the algorithms allow. Note that in progressive refinement vmethods, we must search for the patch with largest amount of unshot radiosity, which involves traversing a potentially large amount of data without performing any operations on it that decrease the residual. The general matrix methods, by contrast, simply process each row of the matrix in order. Although this may mean processing rows whose corresponding patch has no unshot radiosity remaining, in practice performance is enhanced. By avoiding the search phase, the computations can be better pipelined by compilers, and all data which is brought into the processor is actually used in improving the solution rather than searching for the next row to handle.

Furthermore, the innermost loop of the progressive refinement and overshooting methods consists of a *saxpy* or vector update operation, which entails $4n$ memory references (n each for reading ρ_j, $\triangle B_j$, F_{ij}, and an additional n for writing $\triangle B_j$) and $3n$ floating point operations (flops). By contrast the Gauss-Seidel, Chebyshev and CG methods have an inner product as the innermost loop. For the last two methods this entails $2n$ memory references and $2n$ floating point operations, because quantities not indexed by the innermost loop are kept in registers and so do not require a memory reference. In particular, the carry-around scalar that the inner product is summed into, and the reflectivity ρ_i, are kept in registers. Hence the ratio of memory references to flops is $4/3$ for the radiosity specific solvers, while the ratio is 1 for the general matrix methods. This means the general matrix methods better utilize data locality, getting more flops out of data in the cache or registers before having to read or write new cache lines. Note that the better data re-use of the general matrix methods is not *a priori* evident from examining the algorithms. It is possible that in progressive refinement, only a few patches are selected to shoot out radiosity over and over again. In that case, the data associated with those patches would likely remain in cache, potentially giving better data locality properties. Our experiments show that this is not the case in practice, however. Usually over 90% the paches are selected the same number of times, ± 1. Futhermore, the patches selected most often are only selected few more times than the average.

5.2 Effects of Reflectivity

Figures 7-9 show the performance of the various methods as matrix density (occlusion in the environment) is kept fixed at $\delta = 53\%$ and overall reflectivity is increased. Figures 9 and 10 do the same for $\delta = 70\%$. In general convergence slows as the environment's overall reflectivity increases, because the eigenvalues of the matrix become more spread out. Figure 2 shows the effects of the environment's overall reflectivity increase on the eigenvalues distribution. The increase of reflectivity is especially deleterious for PR. Because PR selects which patch to process on each step, it is a nonstationary method which actually changes its innermost loop depending on the specific data of the problem. This means it is not as amenable to analysis as the Chebyshev or CG methods, which are expected to take more steps of iteration as the eigenvalues get spread out.

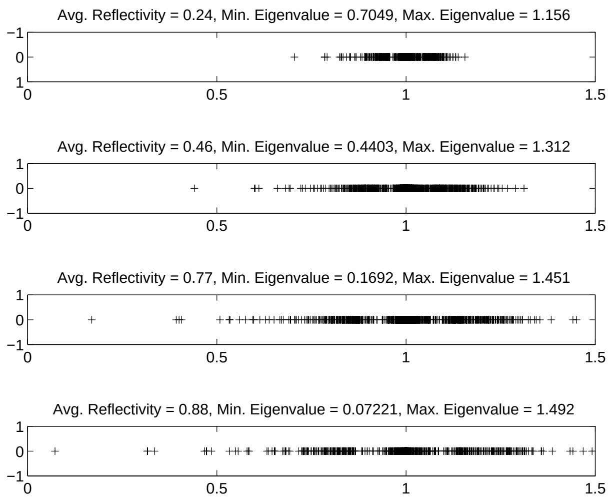


Figure 2: Eigenvalue distribution as reflectivity increases.

As reflectivity increases the spectral radius of the Gauss-Seidel iteration matrix approaches 1.0, and its relative performance decreases. The Gauss-Seidel method has a linear convergence rate that directly depends on the spectral radius of the iteration matrix. The Chebyshev method also has worsening performance, but relative to Gauss-Seidel it is not as sensitive to the increase in reflectivity. The adaptivity of the CG method makes its relative performance better as reflectivity increases; however, as Figure 11 shows only for the highest reflectivity and density levels tested did it become competitive with the Chebyshev method.

Table 2 summarizes the results for the test problems, and for each test problem the algorithms are listed in decreasing order of performance. Note that the Feda method implemented failed on test problems E and F, which have high density and reflectivity. In general, the Gauss-Seidel and Chebyshev methods are the fastest overall. These conclusions do not change when using different stopping tests or the other error norms described earlier.

Table 2: Algorithms performance (total time in seconds).

A	B	C	D	E	F
GS(1.20)	GS(2.01)	CHEBY(4.81)	CHEBY(6.45)	CHEBY(4.05)	CHEBY(5.64)
CHEBY(1.61)	CHEBY(2.81)	GS(6.43)	CG(11.02)	CG(5.05)	CG(5.80)
PR(3.21)	FEDA(5.32)	CG(8.63)	GS(12.83)	GS(5.23)	FEDA(9.92)*
FEDA(3.92)	CG(6.60)	FEDA(11.73)	FEDA(18.47)	FEDA(7.45)**	GS(11.26)
CG(4.68)	PR(6.85)	PR(23.03)	PR(50.56)	PR(22.53)	PR(50.43)

* failed to converge after 2478 steps of iteration

** failed to converge after 3304 steps of iteration

5.3 Effects of Density

Comparing Figures 8 and 10, and Figures 9 and 11, shows the effect of increasing matrix density. Although the absolute times decrease for all the methods, CG presents the higher rate of performance improvement. Figure 3 shows the eigenvalues distribution as the density increases for $\rho = 0.78$. As we can see, the eigenvalues become less spread out, increasing the convergence.

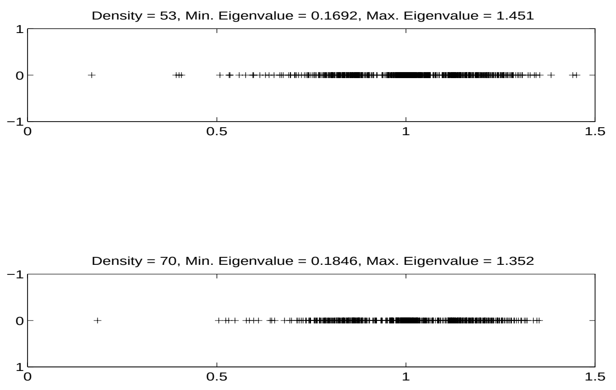


Figure 3: Eigenvalue distribution as density increases.

As mentioned earlier, the relative performance of CG improves as both density and average reflectivity increase. The system becomes much more difficult to solve, and more iterations are required by all the methods. Because CG is adaptive and takes some number of iterations to implicitly accumulate the eigenvalue information it needs, the higher density and reflectivity problems give it more of a chance to accumulate that information.

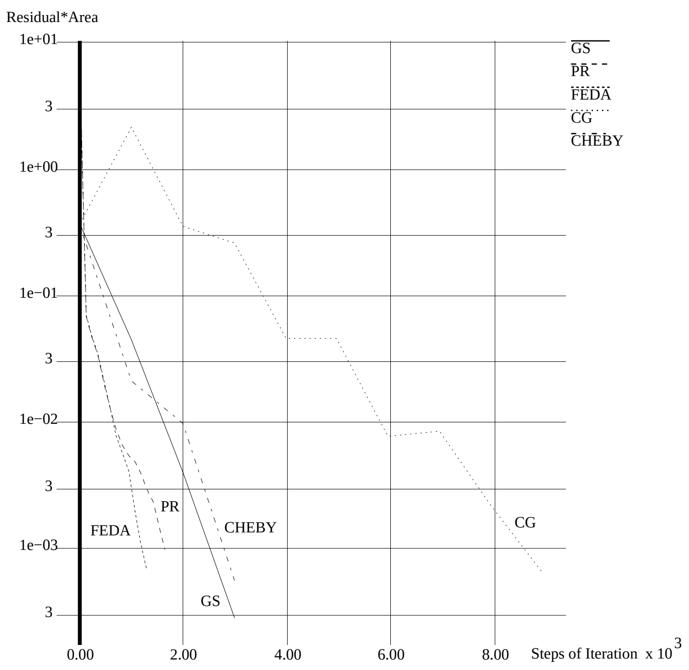


Figure 4: Test case A (steps of iteration): $\rho_{avg} = 0.24 \quad \delta = 53\%$.

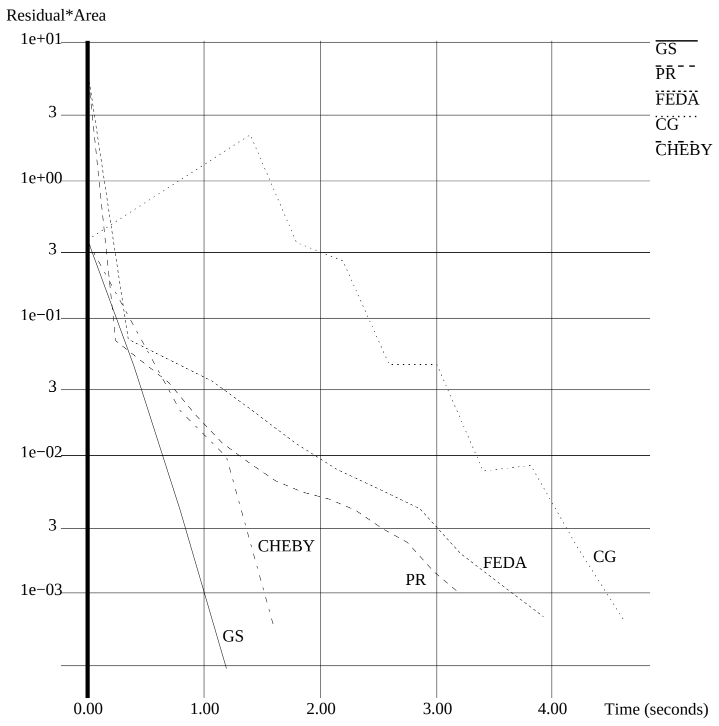


Figure 5: Test case A (time): $\rho_{avg} = 0.24 \quad \delta = 53\%$.

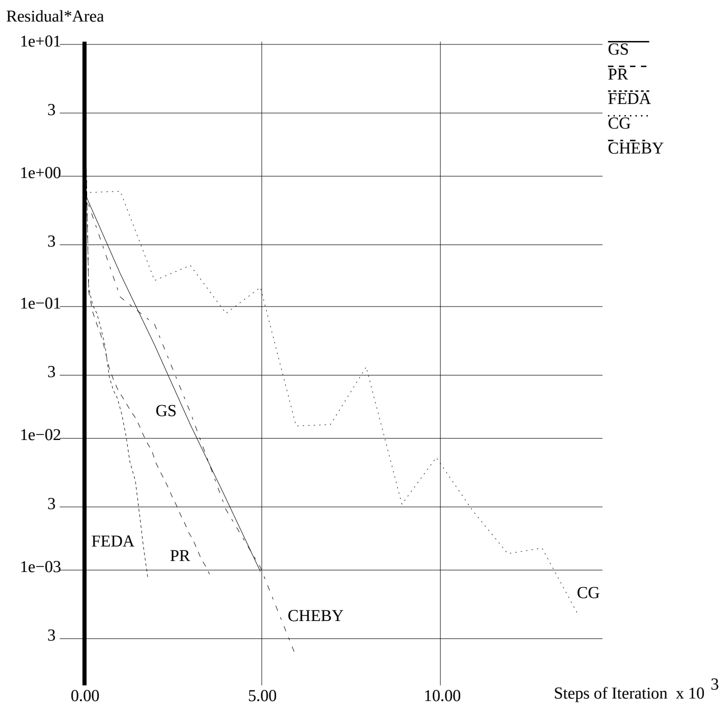


Figure 6: Test case B (steps of iteration): $\rho_{avg} = 0.46 \quad \delta = 53\%$.

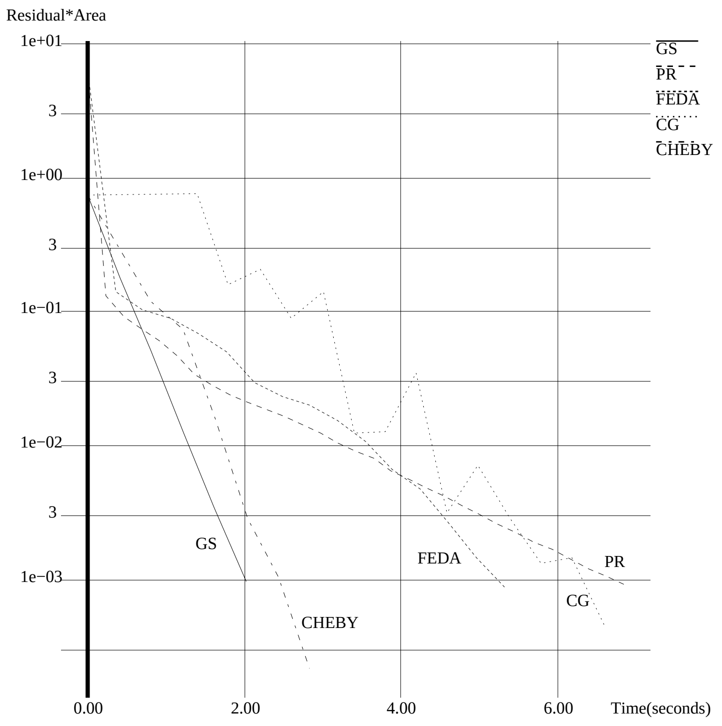


Figure 7: Test case B (time): $\rho_{avg} = 0.46 \quad \delta = 53\%$.

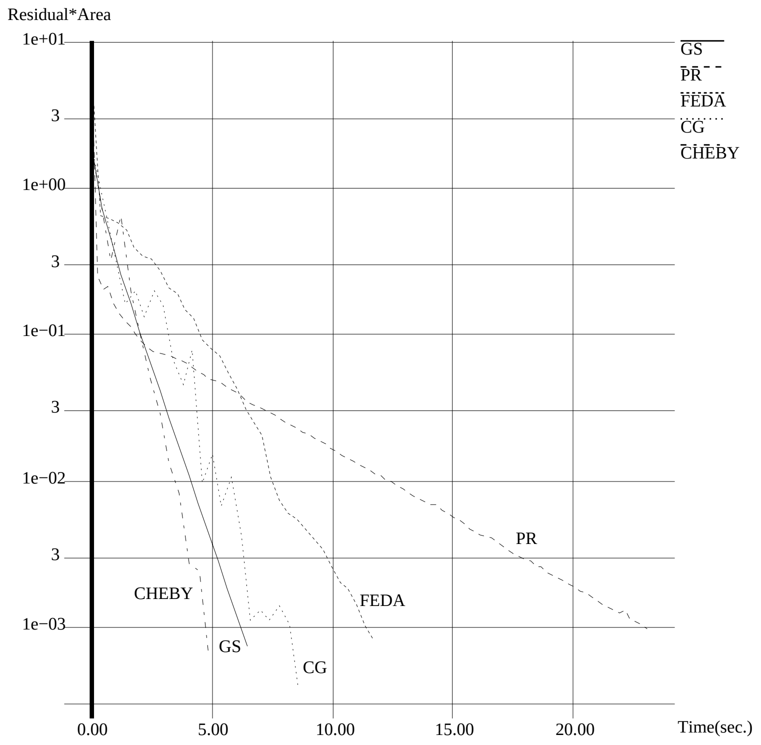


Figure 8: Test case C (time): $\rho_{avg} = 0.77 \quad \delta = 53\%$.

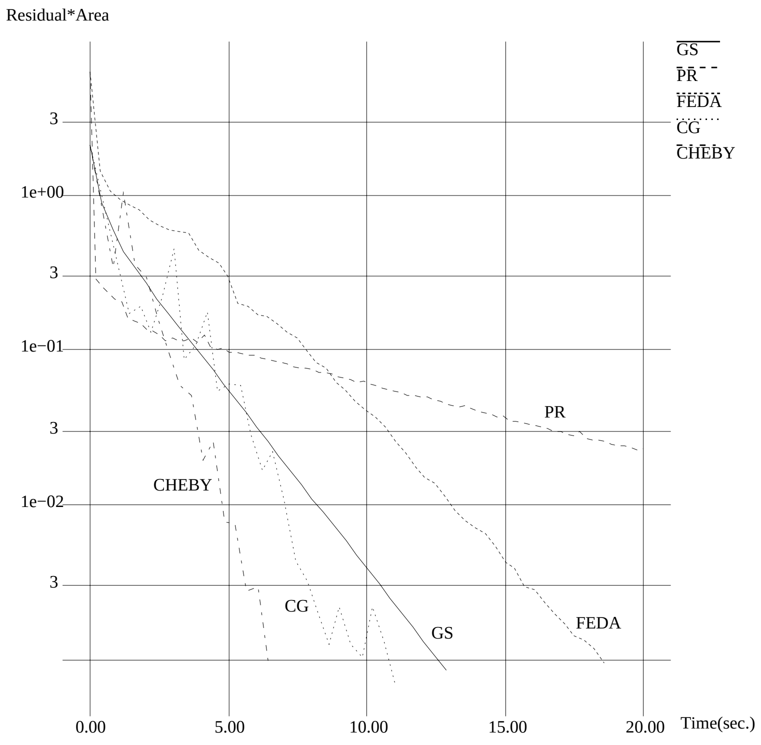


Figure 9: Test case D (time): $\rho_{avg} = 0.88 \quad \delta = 53\%$.

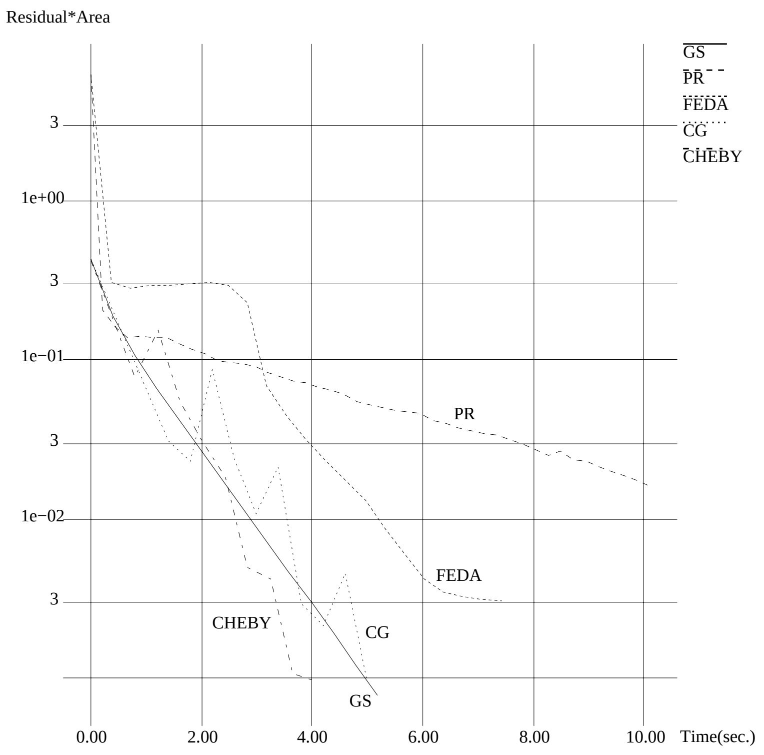


Figure 10: Test case E (time): $\rho_{avg} = 0.78 \quad \delta = 70\%$.

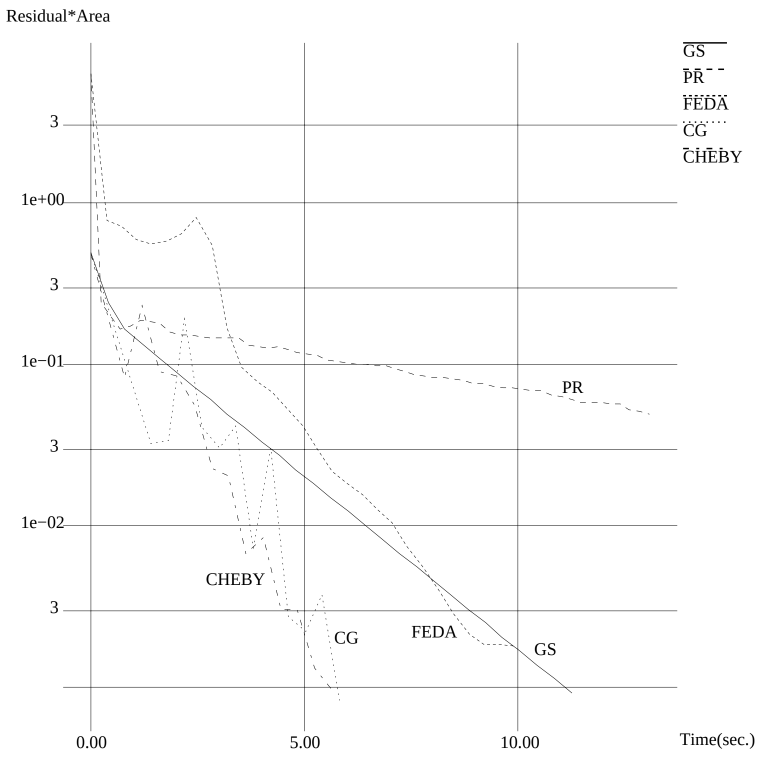


Figure 11: Test case F (time): $\rho_{avg} = 0.89 \quad \delta = 70\%$.

6 Conclusion and Future Research

Our experiments using explicitly stored form factors have shown that although radiosity-specific methods make rapid initial improvement, faster than any other method for limited tolerances (10^{-1}), they are slower than the general matrix methods for higher tolerances. Radiosity-specific methods require searching for a patch to shoot on each step, which can require traversing a large data structure. The disadvantage of general matrix methods of storing the form factors is compensated by the regularity of the computations which allows good pipelining and data locality.

The performance advantages of the general matrix methods are not as attractive when the form factors are computed on the fly, e.g. when n is large. In that case, the innermost loop consists of computing the form factors, which generally requires more flops than the matrix solving algorithms themselves. Avoiding even a few extra form factor computations by searching through rows for the most unshot radiosity may then give the edge to radiosity-specific methods.

For high average reflectance environments, which may occur in several applications, the rate of convergence is slower for all of the iterative methods used. Our experiments have also shown that the CG Method and the Chebyshev method, with the estimates of the maximal eigenvalues described previously, represent the fastest approaches to handle those cases.

The experiments also show that selecting the “best” method is delicate, and no single method is superior in all cases. The relative performance depends on architectural performance features such as pipelining and data locality as well as problem characteristics. Developing pratical solution strategies will likely require implementing a variety of linear solvers, with the one actually used chosen at runtime dependent on problem parameters such as reflectivity and occlusion, and figuring out the best parallel implementation for shared memory multiprocessor workstations. It will also be necessary to bring more numerical linear algebra tools to bear on the problem.

Finally we believe that the understanding of the physical meaning of the eigenvalues and eigenvectors in the radiosity context may help us to obtain even faster approximations for the radiosities vector. Our future efforts will be focused on that question.

Appendix A - Form Factors Accuracy

The major computational bottleneck of any application of the radiosity method is the calculation of form factors. The form factor F_{ij} between two patches **i** (emitter) and **j** (receiver) represents the fraction of power leaving $patch_i$ that arrives directly to $patch_j$. The form factor depends only on the geometry and the orientation of the two patches (Fig. 10). The analytical expression of F_{ij} is given by:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} g_{ij} \frac{\cos\alpha_i \ \cos\alpha_j}{\pi L^2} \, dA_j \, dA_i \tag{29}$$

where:

A_i	=	area of $patch_i$.
A_j	=	area of $patch_j$.
L	=	length of the ray between the differential areas dA_i and dA_j.
α_i	=	angle that the ray makes with $patch_i$'s normal.
α_j	=	angle that the ray makes with $patch_j$'s normal.
g_{ij}	=	visibility term.

The visibility term depends whether there is or not an obstructing surface between dA_i and dA_j. If there is no obstacle g_{ij} is equal to one otherwise g_{ij} is zero.

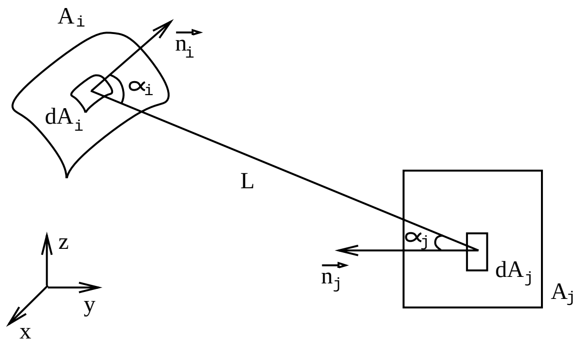


Figure 12: Geometry of the form factor between two patches.

There are several methods for the calculation of form factors between finite areas. These methods can be divided into deterministic and nondeterministic.

The parametric differential method [2] is an example of a deterministic method in which the integrand of equation 28 is evaluated using a numerical technique called Gaussian quadrature. Sample points are placed in both patches following a Gaussian distribution (Fig. 11). The value of g_{ij} is determined testing the intersection of rays, connecting pairs of sample points, with other objects in the environment. For example when we use the 5-points Gaussian quadrature we employ 25 sample points per patch and a total of 625 rays.

In a nondeterministic approach a set of sample points is randomly distributed in the source patch (Fig. 12). The rays are shot in a random direction having a cosine density [25]. The number of times each patch in the environment is hit by a ray is recorded. The form factor between the source $patch_i$ and a certain $patch_j$ is given by the number of rays that hit $patch_j$ divided by the total number of rays shot from $patch_i$. A variation of this approach consists of using a Gaussian distribution of sample points instead of a random distribution (Fig. 13).

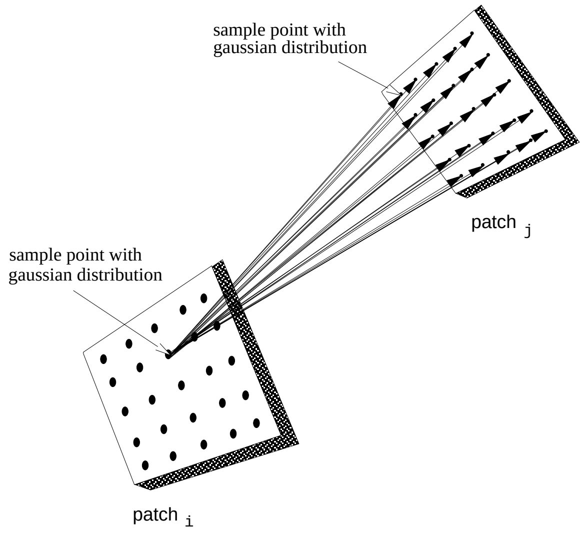


Figure 13: Deterministic method with a Gaussian distribution of sample points.

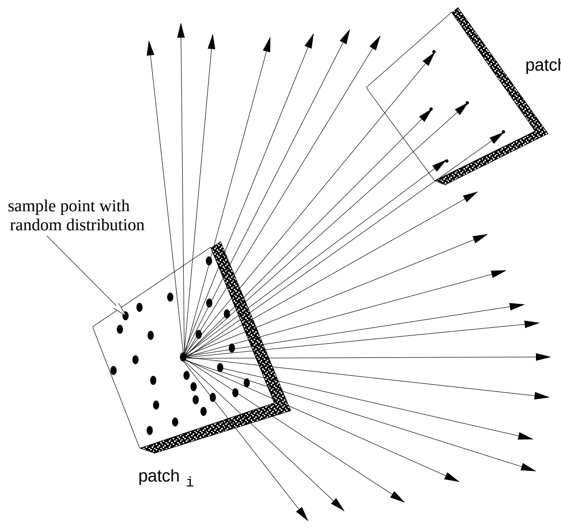


Figure 14: Nondeterministic method with a random distribution of sample points.

We performed some tests to compare the two approaches and their variations. The results are presented in Tables 3,4,5 and 6 in which we use the following terminology:

- PDM - Parametric Differential Method;
- SR - Shooting with random distribution of sample points;

- SD5 - Shooting with 5-points Gaussian distribution of sample points;
- SD10- Shooting with 10-points Gaussian distribution of sample points;

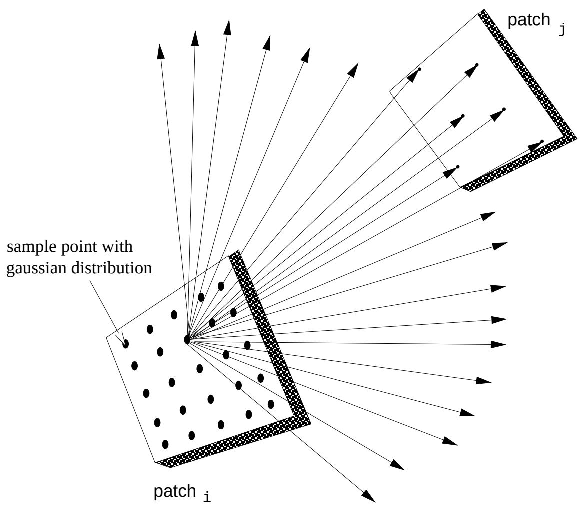
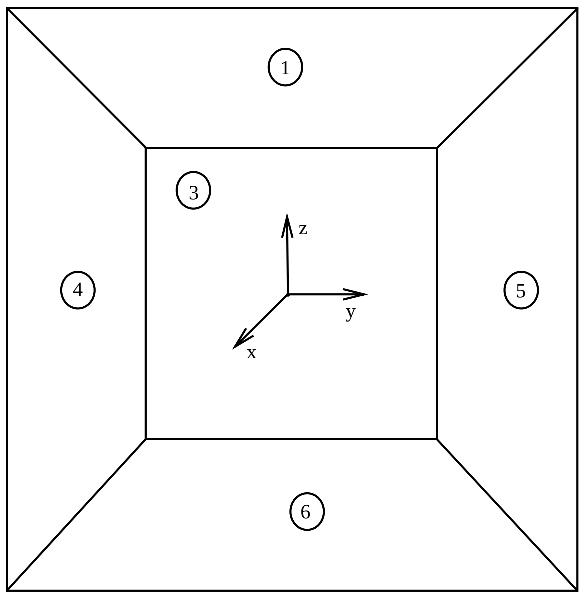


Figure 15: Nondeterministic method with a Gaussian distribution of sample points.

We considered an environment formed by a cube (Fig. 14). Recall that the summation of the form factors of one surface regarding the other surfaces of a closed environment must sum to 1.0. We use this relation to evaluate the results accuracy.



Face 2 is frontal.

Figure 16: Sketch of the test environment.

Notice that the number of sample points per patch used by PDM is constant, 25 points. In the first round of tests (Tables 3,4 and 5) we did not take that into account since we were more concerned about the three variations of the nondeterministic method. In the second round (Table 6), we reduce the number of rays for the nondeterministic methods in order to get a better comparison between the deterministic and nondeterministic methods.

From Tables 3 to 6, it seems that the nondeterministic methods are more accurate than the deterministic methods tested. However since the figures correspond to summations , it may be possible that a cancellation of error terms occurs when we perform the sum of the form factors of each face. In that case, for some geometries a deterministic method may be more suitable than a nondeterministic one.

The accuracy of the tested methods is affected by the singularity in the integrand which occurs when the two differential areas are too close. PDM handles the singularity using an analytic approach before the numerical evaluation. However it does not remove the singularity completely. In the case of nondeterministic methods an alternative to minimize this problem consists of changing the position of the sample points near the edge between two patches. This procedure may itself represent another source of error.

When the singularity does not occur, e.g. in the case of parallel patches, the results obtained using PDM are very accurate. The analytic form factors between the parallel faces of the test environment (Fig. 11) is approximately 0.1998 [9]. As we can see in Table 7, the figures obtained using the deterministic method are closer to the analytic value than those obtained using nondeterministic methods.

On the other hand, when the singularity does occur the task of determining which method is more accurate becomes more complex. Table 8 presents the relative errors regarding the form factors between perpendicular faces whose analytic value is approximately 0.200043 [9]. As we can see the average of the relative errors of nondeterministic methods is smaller than the average of PDM. In fact the values are 0.78%, 0.48% and 0.71% for PDM, SR and SG5 respectively. However we can also notice that in some cases the error regarding PDM may be smaller.

Final Remarks:

- Nondeterministic methods can be extended to specular case in a more natural way;
- Techniques to solve the singularity problem completely have not been extensively explored;
- Further tests shall take into account more complex environments, with a finner grid of patches and including occlusion and curved objects. We expect that these tests may indicate in which situation is more convenient to use a deterministic or a nondeterministic method;
- Statistical analysis of the accuracy is required;
- Since the errors of the form factors obtained using a nondeterministic method are non uniform is necessary to use statistics to evaluate their accuracy.

Table 3: Environment divided into 216 patches.

Method	PDM	SR	SG5	SG10
number of points/patch	25	1000	25	100
number of rays/point	25	1	40	10
total number of rays	625	1000	1000	1000
Σ of face l	1.013652	1	1	1
Σ of face 2	1.013652	0.997333	1.00186	1.00194
Σ of face 3	1.013652	1.00114	1.00233	1.00292
Σ of face 4	1.013652	0.998056	0.999	0.998778
Σ of face 5	1.013652	1.00367	1.00728	1.01075
Σ of face 6	1.013652	1.00464	1.00431	1.00333
Relative error (%)	8.19	2.43	1.68	2.02

Table 4: Environment divided into 486 patches.

Method	PDM	SR	SG5	SG10
number of points/patch	25	1000	25	100
number of rays/point	25	1	40	10
total number of rays	625	1000	1000	1000
Σ of face l	1.009103	1.0	1.0	1.0
Σ of face 2	1.009103	0.997815	0.998494	0.998852
Σ of face 3	1.009103	1.0009	1.00328	1.00312
Σ of face 4	1.009103	1.00023	0.997444	0.997938
Σ of face 5	1.009103	1.00049	1.00523	1.00502
Σ of face 6	1.009103	0.998506	1.00448	1.00373
Σ of relative error (%)	5.46	0.53	1.70	1.51

Table 5: Environment divided into 1014 patches.

Method	PDM	SR	SG5	SG10
number of points/patch	25	1000	25	100
number of rays/point	25	1	40	10
total number of rays	625	1000	1000	1000
Σ of face l	1.006303	1.0	1.0	1.0
Σ of face 2	1.006303	1.00034	1.00032	1.00042
Σ of face 3	1.006303	1.00208	1.0047	1.00472
Σ of face 4	1.006303	0.999124	0.997692	0.997704
Σ of face 5	1.006303	1.00108	0.999473	0.998828
Σ of face 6	1.006303	0.995243	0.999568	1.00027
Σ of relative error (%)	3.78	0.91	0.78	0.81

Table 6: Environment divided into 1014 patches with a limited number of rays (625) per patch.

Method	PDM	SR	SG5
number of points/patch	25	1000	25
number of rays/point	25	1	25
total number of rays	625	625	625
Σ of face 1	1.006303	1.0	1.0
Σ of face 2	1.006303	1.00016	1.00301
Σ of face 3	1.006303	0.97264	0.98485
Σ of face 4	1.006303	0.999451	0.994348
Σ of face 5	1.006303	1.00034	0.998968
Σ of face 6	1.006303	1.00235	1.00153
Σ of relative error (%)	3.78	0.61	1.27

Table 7: Form factors regarding the parallel faces divided into 169 patches each.

$Face_i - Face_j$	PDM	Rel. error(%)	SR	Rel. error(%)	SG5	Rel. error(%)
1 - 6	0.199841	0.02	0.199044	0.38	0.198570	0.62
2 - 3	0.199841	0.02	0.199044	0.38	0.199508	0.15
4 - 5	0.199841	0.02	0.199195	0.30	0.200398	0.30

Table 8: Form factors regarding the perpendicular faces divided into 169 patches each.

$Face_i - Face_j$	PDM	Rel. error(%)	SR	Rel. error(%)	SG5	Rel. error(%)
1 - 2	0.201615	0.78	0.200502	0.23	0.202556	1.26
1 - 3	0.201615	0.78	0.200634	0.30	0.202622	1.30
1 - 4	0.201615	0.78	0.199044	0.50	0.196781	1.63
1 - 5	0.201615	0.78	0.200625	0.29	0.199470	0.29
2 - 4	0.201615	0.78	0.198892	0.58	0.198692	0.68
2 - 5	0.201615	0.78	0.201051	0.50	0.200312	0.13
2 - 6	0.201615	0.78	0.200691	0.32	0.201941	0.95
3 - 4	0.201615	0.78	0.200161	0.06	0.198343	0.85
3 - 5	0.201615	0.78	0.198305	0.86	0.197955	1.04
3 - 6	0.201615	0.78	0.199138	0.45	0.200057	0.007
4 - 6	0.201615	0.78	0.202159	1.06	0.200133	0.05
5 - 6	0.201615	0.78	0.201164	0.56	0.200833	0.40

Appendix B - Plots of the Error Norms

We are going to present the graphs regarding the following error norms that we have computed for the test cases A,B,C,D,E and F:

- L_2 norm of the residual, Figures 15-20;
- RMS_{error}, Figures 21-26;
- percentage of "reflected" radiosity (*error*), Figures 27-32;

The stopping criteria and the tolerance used to plot these error norms against the time were the same as those used previously, $\max(r_i A_i)$ and $tol = 10^{-3}$ respectively. For RMS_{error} and *error* norms we used as an approximation to the exact solution vector the radiosities vector obtained using Gauss-Siedel Method iterated until the residual vector has norm less than 10^{-10}.

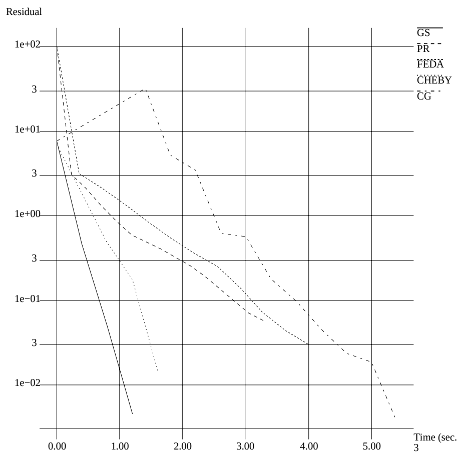


Figure 17: Residual for test case A.

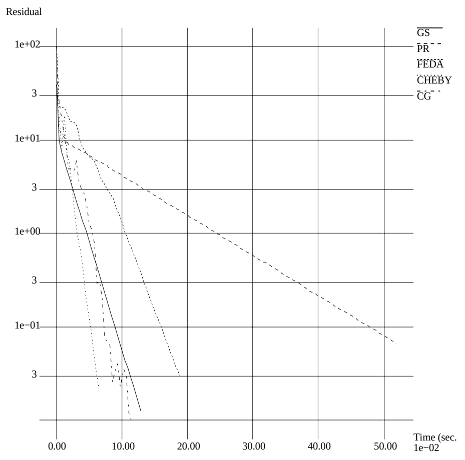


Figure 20: Residual for test case D.

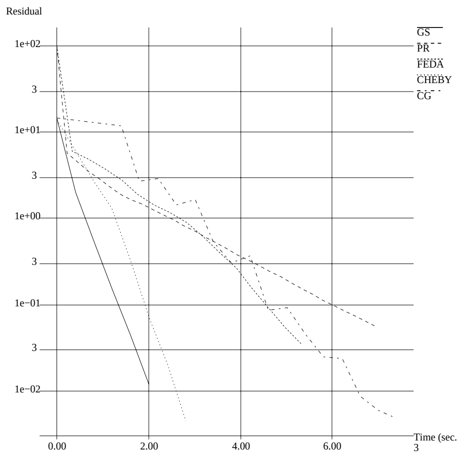


Figure 18: Residual for test case B.

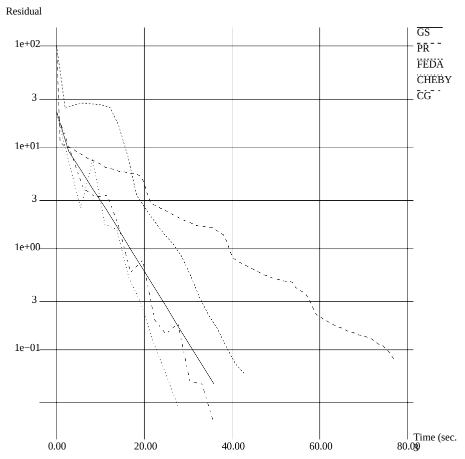


Figure 21: Residual for test case E.

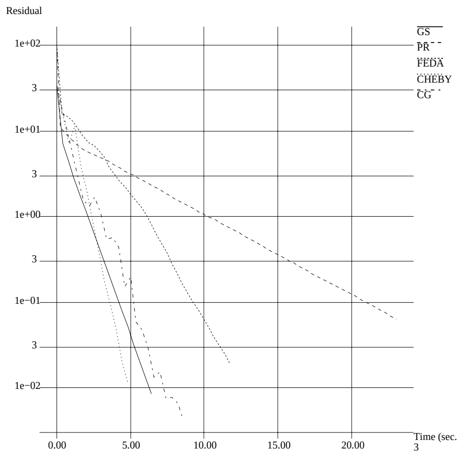


Figure 19: Residual for test case C.

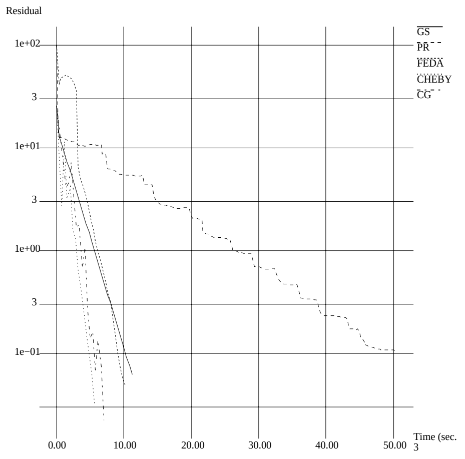


Figure 22: Residual for test case F.

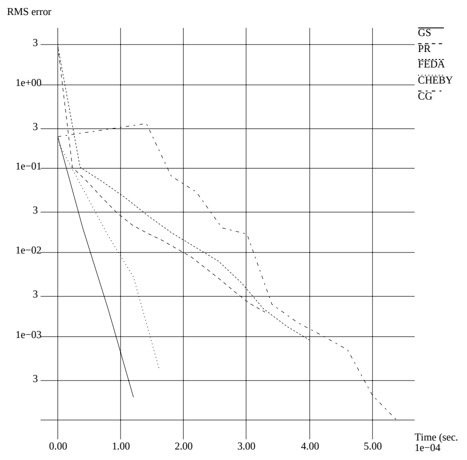


Figure 23: RMS_{error} for test case A.

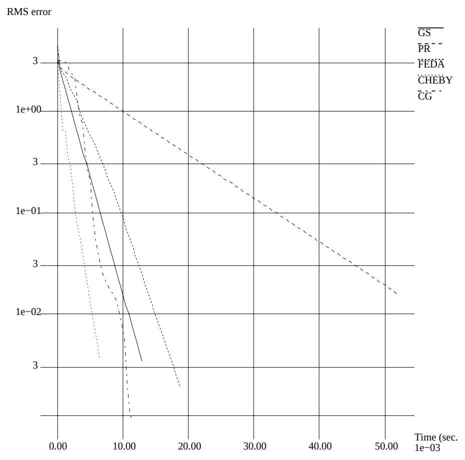


Figure 26: RMS_{error} for test case D.

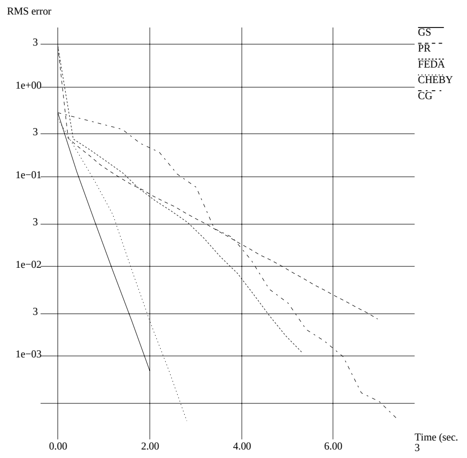


Figure 24: RMS_{error} for test case B.

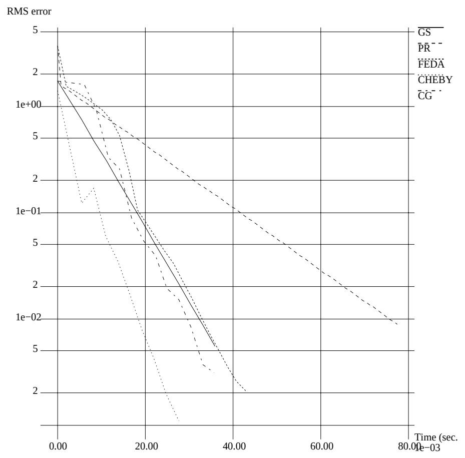


Figure 27: RMS_{error} for test case E.

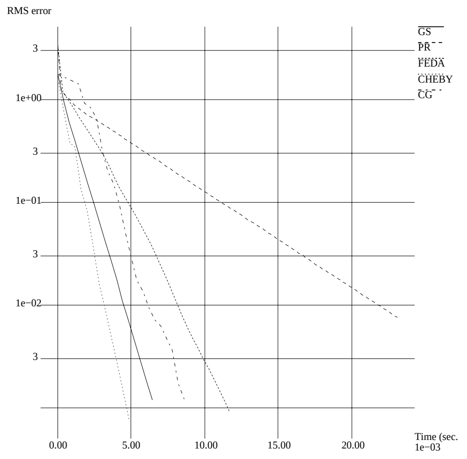


Figure 25: RMS_{error} for test case C.

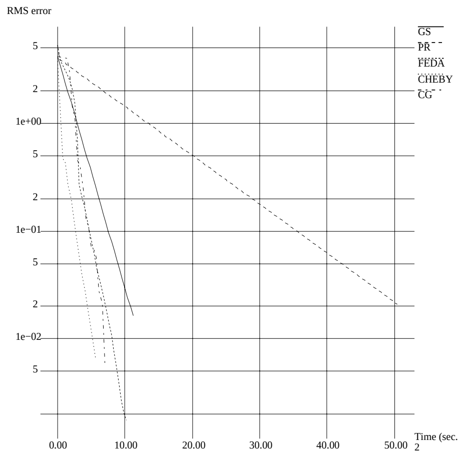


Figure 28: RMS_{error} for teste case F.

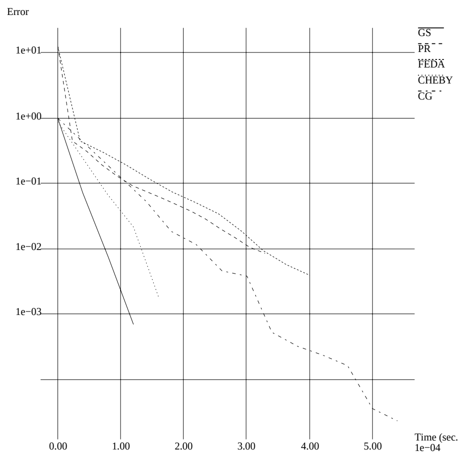


Figure 29: Error for test case A.

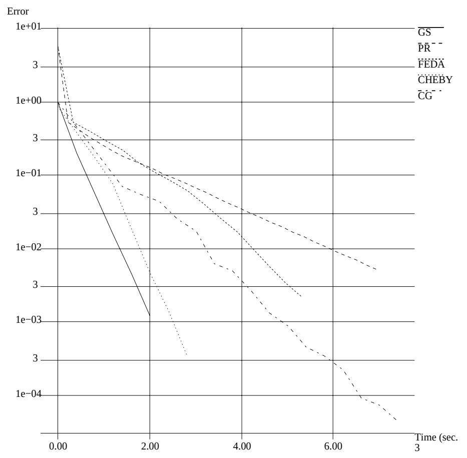


Figure 30: Error for test case B.

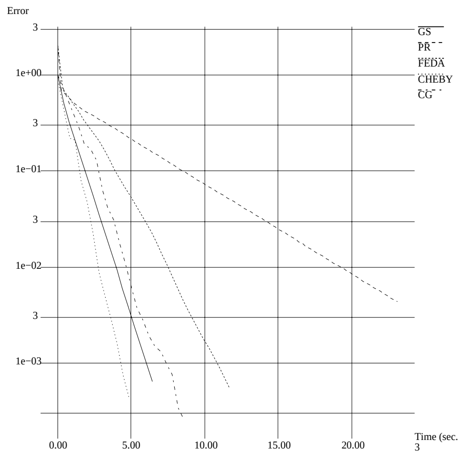


Figure 31: Error for test case C.

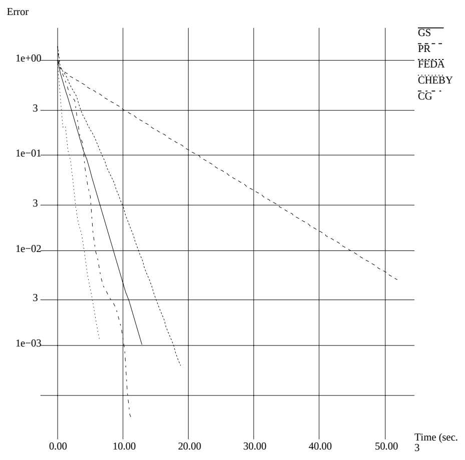


Figure 32: Error for test case D.

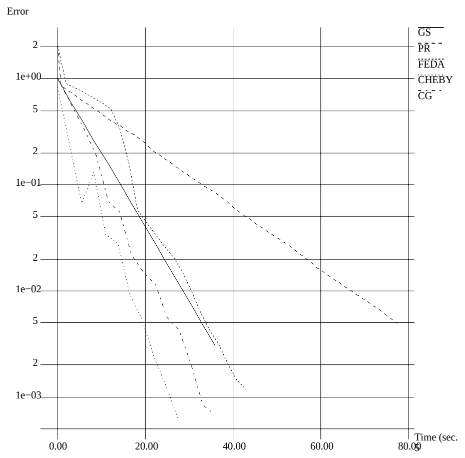


Figure 33: Error for test case E.

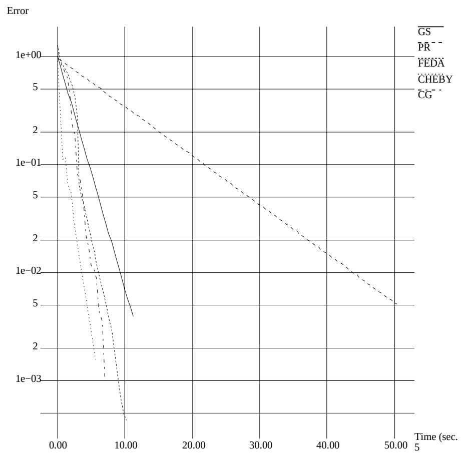


Figure 34: Error for teste case F.

References

[1] O. Axelsson and G. Lindskog, *On the eigenvalue distribution of a class of preconditioning methods*, Tech. Rep. Report 3, Goteborg Numerical Analysis, 1989.

[2] G. V. G. Baranoski, *The parametric differential method: An alternative to the calculation of form factors*, Computer Graphics Forum, 11 (1992), pp. 193–204.

[3] R. Barrett et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1 ed., 1994.

[4] D. Baum, J. Wallace, M. Cohen, and D. Greenberg, *The back-buffer algorithm: an extension of the radiosity method to dynamic environments*, The Visual Computer, 2 (1986), pp. 298–306.

[5] R. Burden and J. Douglas, *Numerical Analysis*, PWS-KENT Publishing Company, Boston, 4 ed., 1988.

[6] S. Chen, *Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system*, Computer Graphics, 24 (1990), pp. 135–144.

[7] M. Cohen, S. Chen, J. Wallace, and D. Greenberg, *A progressive refinement approach to fast radiosity image generation*, Computer Graphics, 22 (1988), pp. 75–84.

[8] M. Cohen and D. Greenberg, *The hemi-cube: A radiosity solution for complex environments*, Computer Graphics, 19 (1985), pp. 31–40.

[9] M. Cohen and J. Wallace, *Radiosity and Realistic Image Synthesis*, Academic Press Professional, Cambridge, 1 ed., 1993.

[10] M. Feda and W. Purgathofer, *Accelerating radiosity by overshooting*, in Proc. of the Third Eurographics Rendering Workshop, Consolidation Express, June 1992, pp. 21–32.

[11] D. W. George, F. Sillion, and D. Greenberg, *Radiosity redistribution for dynamic environments*, IEEE Computer Graphics and Applications, 10 (1990), pp. 26–34.

[12] S. Goertler, M. Cohen, and P. Slusallek, *Radiosity and relaxation methods*, tech. rep., Princeton University, 1993.

[13] G. Golub and C. V. Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, 2 ed., 1989.

[14] D. Greenberg, *Computers and architecture*, Scientific American, 264 (1991), pp. 104–109.

[15] G. Greiner, W. Heidrich, and P.Slusallek, *Proc. of the fourth eurographics rendering workshop*, in Proc. of Eurographics Rendering Worshop, June 1993, pp. 233–245.

[16] L. Hageman and D. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.

[17] P. Hanrahan, D. Salzman, and L. Aupperle, *A rapid hierarchical radiosity algorithm*, Computer Graphics, 25 (1991), pp. 197–206.

[18] P. Heckbert and J. Winget, *Finite element methods for global illumination*, tech. rep., University of California, Berkley, 1991.

[19] M. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, Journal of Research National Bureau of Standards, 49 (1952), pp. 409–436.

[20] L. Neumann, *New efficient algorithms with positive definite radiosity matrix*, in Proc. of the Fifth Eurographics Rendering Workshop, June 1994, pp. 219–237.

[21] Y. Saad, A. Sameh, and P.Saylor, *Solving elliptic difference equations on a linear array of processors*, SIAM Journal of Scientific and Statistical Computing, 6 (1985), pp. 1049–1063.

[22] M. Shao and N. Badler, *Analysis and acceleration of the progressive refinement method*, in Proc. of the Fourth Eurographics Rendering Workshop, June 1993, pp. 247–258.

[23] J. Shewchuck, *An introduction to the conjugate gradient method without the agonizing pain*, tech. rep., Scholl of Computer Science, Carnegie Mellon University, 1994.

[24] P. Shirley, *Physically based lighting for computer graphics*, ph.d. thesis, University of Illinois, 1990.

[25] ——, *Radiosity via ray tracing*, in Graphics Gems II, Ed. Academic Press, 1991, pp. 306–310.

[26] E. Stiefel, *Kernel polynomials in linear algebra and their numerical application*, in Further Contributions to the Solutions of Simultaneous Linear Equations and the Determination of Eigenvalues, National Bureau of Standards, Applied Mathematical Series - 49, 1958.

[27] W. Xu and D. S. Fussel, *Constructing solvers for radiosity equation systems*, in Proc. of the Fifth Eurographics Rendering Workshop, June 1994, pp. 207–217.