An Empirical Comparison of Radiosity Algorithms

Andrew J.Willmott and Paul S. Heckbert

12 May 1997
CMU-CS-97-115

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

This report presents an extensive empirical comparison of matrix, progressive, and wavelet radiosity algorithms for simulating diffuse interreflection in three-dimensional scenes. The algorithms are tested in their basic forms, without advanced variations such as clustering, discontinuity meshing, or Monte Carlo techniques. The three algorithms were implemented in a common code base to facilitate direct empirical comparison. A number of parameterized scenes were designed to test the basic methods' ability to deal with such issues as singularities, occlusion, high reflectance, and scene complexity. Each algorithm was run on the set of scenes at several parameter settings, and results were examined in terms of their error, speed, and memory consumption.

For the basic algorithms as we implemented them, our results show: Progressive radiosity with substructuring is best for simple scenes, but for moderately complex scenes it is outperformed by wavelet radiosity using the Haar basis. Wavelet methods use an immense amount of memory; without clustering they become totally impractical for complex scenes. The problem is particularly severe for higher order bases, less so for Haar. Visibility handling was also found to be a critical problem with higher order wavelets.

This study also provides a general framework for comparisons of global illumination techniques.

This work was supported by National Science Foundation Young Investigator award CCR-9357763 and ARPA contract F19628-93-C-0171.

Keywords: global illumination, matrix radiosity, progressive radiosity, wavelet radiosity, visibility, validation.

Table of Contents

1. Introduction . . . 1
1.1. Motivation . . . 1
1.2. Radiosity Algorithms . . . 2
1.3. Algorithms to be Compared . . . 3
1.4. Neglected Techniques . . . 4
1.5. Remaining Sections . . . 5
2. Radiosity Implementations . . . 7
2.1. Matrix Radiosity . . . 7
2.1.1. Matrix Solution . . . 8
2.1.2. Integral Equations for Radiosity . . . 8
2.1.3. Form Factor Calculations . . . 9
2.2. Progressive Radiosity . . . 10
2.3. Progressive Radiosity with Substructuring . . . 11
2.4. Wavelet Radiosity . . . 12
2.4.1. Wavelet Control Algorithm . . . 13
2.4.2. Quadrature . . . 14
2.4.3. The Wavelet Radiosity Oracle . . . 15
2.4.4. Termination Criteria . . . 16
2.5. Meshing . . . 17
2.6. Visibility . . . 18
2.7. Code Structure . . . 20
3. Experimental Method . . . 21
3.1. Experiment Scenes . . . 21
3.2. The Radiosity Testbed . . . 27
3.2.1. Timing and Checkpointing . . . 27
3.2.2. Reference Solutions . . . 27
3.2.3. Measuring Error . . . 28
3.2.4. Measuring Memory Use . . . 29
3.2.5. Result Equalisation . . . 30
3.2.6. Summary . . . 32
4. Results . . . 33
4.1. Performance Analysis . . . 33
4.1.1. Parallel and Perpendicular . . . 33
4.1.2. Blocker . . . 36
4.1.3. Box and Tube . . . 39
4.1.4. Complex . . . 41
4.1.5. Lights . . . 47
4.2. Other Experimental Results . . . 49
4.2.1. Components of Solution Time . . . 49
4.2.2. Analysing Visibility . . . 50
4.2.3. Wavelet and Progressive Solution Curves . . . 50
4.2.4. Over-Refinement . . . 52
4.2.5. Reflectance . . . 52
4.2.6. Varying Epsilon for Wavelet Radiosity . . . 56
4.2.7. Meshing . . . 58
4.3. Testing Theory . . . 58
4.3.1. Linear System Solution . . . 58
4.3.2. Progressive Convergence . . . 60
4.3.3. Ambient Correction Term . . . 62
4.3.4. Complexity of Wavelet Radiosity . . . 62
4.4. Hardware Used . . . 64

5. Conclusions . . . 67
5.1. Primary Conclusions . . . 67
5.2. Empirical Complexity Results . . . 68
5.2.1. Progressive Methods . . . 68
5.2.2. Wavelet Methods . . . 69
5.3. Problems Observed and Proposed Solutions . . . 69
5.3.1. Progressive Methods . . . 70
5.3.2. Wavelet Methods In General . . . 70
5.3.3. Higher-Order Wavelets . . . 70
5.4. Miscellaneous . . . 72
5.4.1. Fast Visibility for Wavelet Methods . . . 72
5.4.2. High Reflectance Scenes . . . 72
5.4.3. Performance in Complex Scenes . . . 72
5.4.4. Setting Parameters . . . 72
5.4.5. Matrix Radiosity . . . 73
5.5. Future Work on Radiosity Comparisons . . . 73
Appendix A . . . 79
A.1. Wavelet Coefficients . . . 79
A.2. An Example Run File . . . 81
A.3. An Example Mesh Output File . . . 82
A.4. Plotting Examples . . . 83
A.5. Code Snippet for Area-to-Point Form Factors . . . 83

1. Introduction

This report discusses the design and results of a project to compare empirically several radiosity methods. We begin with our motivation.

1.1. Motivation

Radiosity methods were first developed in the field of mechanical engineering [54] and they have since been generalized and optimized in a number of ways within the field of computer graphics [2, 14, 48]. The three basic algorithms are matrix radiosity [13], progressive radiosity [12], and wavelet radiosity [22, 25]. Although many variations on these algorithms have been proposed in recent years, some basic questions about the relative performance of these fundamental algorithms remain unanswered.

- Which methods are fastest?
- Which methods generate the most realistic pictures?
- Is the cost of wavelet radiosity really linear in the number of elements?
- What wavelet basis functions should I use?
- Are the discontinuities I've seen in some wavelet radiosity pictures a problem?
- Which method is best for getting good shadows? For extremely complex scenes?
- All things considered, which algorithm should I use?

The radiosity literature does not currently answer these questions in a very satisfying manner. While the papers introducing these techniques have demonstrated the strengths of each radiosity algorithm, and some have offered theoretical complexity analyses, radiosity algorithms have not been compared extensively in terms of actual speed and accuracy. Researchers typically implement and demonstrate their own new method, but seldom implement many other people's algorithms or perform extensive empirical comparisons. There seem to be several reasons for this: implementing a radiosity algorithm properly can be a complex and time-consuming process; there is little free, public radiosity software available; comparing results is not as simple as measuring CPU time; developing new algorithms is generally considered more exciting; and there are few incentives for performing large-scale empirical software tests in computer graphics. This pattern is not universal, however: ray tracing algorithms have been compared to each other much more extensively[1].

There have been several comparisons of radiosity techniques in the literature, but most of these have focused on comparing matrix and progressive radiosity, or variants of these algorithms. Cohen et al. showed that progressive radiosity computes a reasonable solution more quickly than Gauss-Seidel matrix radiosity [12]. Gortler et al. and Xu-Fussell compared basic progressive radiosity with overshooting variations, and found that overshooting can accelerate convergence significantly [21, 62]. Various methods for matrix radiosity, including Gauss-Seidel, conjugate gradients, Chebyshev's method, and two variants of progressive radiosity were compared by Baranoski et al. [4]. They found Chebyshev's method and conjugate gradients to be fastest. All these comparisons were typically made on a few scenes with identical meshes. Baranoski examined speed and accuracy as a function of scene reflectance on a scene consisting of a sphere in a box. Gortler studied two scenes (an office with some furniture, and an empty box) and two hypothetical radiosity-like matrices. We believe that in all of these studies, error was measured by computing a reference solution using the same mesh, but letting their fastest method run until "convergence". Measuring error in this way is a bit misleading in that it ignores the discretization error due to the mesh. Error was typically defined as the residual of the matrix (Ab-e), although Gortler et al. corrected for the effect of overall scene brightness on the error. In these studies, results were typically presented as a graph of error as a function of iteration number. Most of the comparisons between wavelet and higher order radiosity methods have been simple, unoccluded, "flatland" experiments [27,

1. Probably because (1) it is easy to write a simple ray tracer, (2) there is more of a "hacker culture" around ray tracing than around radiosity, and (3) standard test scenes were established early on [23].

43]. To our knowledge, no thorough empirical comparisons of wavelet radiosity with matrix or progressive radiosity have been done.

In order to provide some hard data on the relative performance of basic radiosity methods, to learn more about the strengths and weaknesses of these algorithms, and to develop a deeper understanding of the maths and physics of diffuse interreflection, we designed and carried out a large array of empirical experiments. We cannot answer all questions for all algorithms; so we restrict ourselves to the basic three; matrix, progressive, and wavelet, tested on a variety of scenes. Rather than piece together existing code or experiments, we re-implemented and re-ran the algorithms ourselves, since differences in programming style, compiler, computer, and test scene can cause large variations in running times. Re-implementing in a common code base allowed us to compare speeds of various algorithms directly. Such empirical comparison is valuable, since the theoretical performance of an algorithm is not always wholly representative. For example, in the area of spatial data structures for ray-tracing, octrees have excellent theoretical running times, but in practice they are often outperformed by simple grids. In the real world, constant factors and expected case behaviour are often more important than asymptotic, worst case behaviour.

The accuracy of methods were also compared, since speed is not the only concern. It is misleading, for example, to say that method A is faster than method B if A's result is much poorer than B's. Methods were therefore compared in terms of the trade-off between speed and accuracy that they made.

1.2. Radiosity Algorithms

To discuss the algorithms, we will need some notation. Let k denote the number of input surfaces (typically polygons), n the number of patches into which the surfaces are subdivided during meshing, and v the cost of determining visibility between a pair of points in the scene. In a simple implementation, $v=O(k)$, but with good data structures and enough memory, $v = O(k^{1/3})$ [10] or $O(\log k)$ [32] is achievable. Note that many radiosity papers neglect the v term in their complexity analysis.

Rather than compare different variations that are built on these fundamental algorithms, we examine their foundation: the three basic classes of algorithms for radiosity rendering. This is important because we feel that their relative merits have not been adequately discussed.

The first of these is **matrix radiosity**, which explicitly computes a large *form factor* matrix and solves several large systems of equations [20, 35, 13]. First, the scene is discretized into patches, then form factors are computed. The resulting $n \times n$ linear system of equations is typically solved using an iterative method such as Gauss-Seidel iteration [56]. Computing the matrix is the most costly step. It takes $O(n^2v)$ time because n^2 form factors must be computed and each one requires one or more visibility tests. Solving takes only $O(n^2)$ time in practice because only a small, fixed number of iterations each of cost $O(n^2)$ is typically required. The amount of storage required is $O(n^2)$. Visibility can be tested either using a hemicube [13], in which case the amortized cost of visibility can be $v=O(1)$, or using ray tracing [59], in which case v is larger. Because it is so expensive in time and space, we do not expect matrix radiosity to be competitive with the other methods, but we implemented it because it is useful as a reference.

The second of these is **progressive radiosity**, which progressively refines the image by computing the matrix and the solution incrementally [12]. Progressive radiosity iteratively "shoots" light from the brightest light sources and reflective surfaces, computing just one column of the form-factor matrix at a time. This is a variant of Southwell's relaxation technique for solving linear systems [21, 45]. Although this relaxation technique has been largely superseded by other iterative techniques in the numerical methods literature [5], it has proven useful for radiosity simulations. If s shooting steps are used, the time cost is $O(nsv)$. In practice it converges much faster than matrix radiosity, so $s \ll n$. Progressive radiosity has the added advantage that storage is only $O(n)$, since matrix columns are discarded after they are used.

The final class of methods is called **wavelet radiosity**, though the first of these algorithms, hierarchical radiosity [25], was developed without reference to wavelet techniques. These methods employ multilevel meshes to represent the radiosity function, and allow inter-patch interactions to take place between arbitrary levels of the mesh hierarchy [22,55,42]. Thus, unlike previous methods, wavelet radiosity methods do not use a fixed mesh, but a mesh that is adaptive to the other surface "viewing" them. When reflecting light to a distant surface, a given surface is meshed coarsely, but when reflecting to a nearby surface, it is meshed more finely. Various basis functions can be used to represent the radiosity across an element. Theoretically, wavelet radiosity methods cost $O(k^2v + nv)$ [14, p. 346].

All three of these can be regarded as different methods for approximating and solving the integral equation governing radiosity in the continuous domain [27]. The *kernel* of the integral equation, which is a continuous function equal to reflectance times geometric form factor, is approximated in one of the above ways to create the matrix for the system of equations that is solved. In the case of matrix radiosity, the matrix is stored explicitly, while in the other algorithms, the matrix exists only conceptually.

Currently the most commonly implemented radiosity algorithm is probably progressive radiosity. The commercial system from Lightscape [30], for example, uses progressive radiosity. Wavelet radiosity is primarily a research topic, and only a handful of implementations of higher-order bases exist. We suspect this is largely due to the added complexity in implementing the wavelet radiosity algorithms, especially for non-Haar bases; the details of wavelet radiosity implementation have not been well described in the literature.

1.3. Algorithms to be Compared

In this study we have chosen to compare various methods for approximating the radiosity kernel and solving the system of equations. To make the comparison fair, we attempted to choose the best available combination of techniques with each basic algorithm, using, for example, fast, accurate form factor formulas and system solvers.

We have chosen to compare the following radiosity methods:

1. **Matrix radiosity**. The scene is diced into patches according to a density measure; a maximum edge length is specified, and all patches must have sidelengths smaller than this. The matrix can be solved using either successive overrelaxation (a fast variant of Gauss-Seidel) or the conjugate-gradient method, which is even faster [18]. Since form factor calculation takes much longer than solving, the choice of solver typically has a small impact on the total time.

2. **Progressive radiosity without substructuring**. A simple one-level mesh is used. An ambient term is optionally included which estimates reflected light in the earlier iterations. This method saves space by only storing one column of form-factors at a time. However, form factors and visibility samples may be evaluated more than once, if a patch shoots multiple times.

3. **Progressive radiosity with substructuring**. Substructuring introduces a two-level hierarchy to the mesh; the coarser patches shoot light to a finer set of elements [11]. The mesh is further subdivided adaptively in regions of high radiosity gradient, such as shadow boundaries. Substructuring is generally regarded as being very desirable since it speeds up the algorithm greatly. Wavelet techniques take the idea further, going beyond two levels.

4. **Wavelet Radiosity**. The wavelet family of methods share a common algorithm and theory, but differ in the basis functions used to represent the radiosity over a patch in the scene. The bases we have implemented are the Haar basis [55], flatlets of order 2 and 3 [22], and multiwavelets, also of order 2 and 3 [1]. The higher order methods have more “vanishing moments”, meaning they can approximate smooth functions more sparsely (with fewer nonzero coefficients). They have disadvantages, however. The higher order methods create sparser approximations to the kernel, while the lower order methods are easier to integrate and fewer coefficients are required per link. Unfortunately, none of the basis functions we implemented are guaranteed to produce continuous solution functions; small jumps in the radiosity function are common. Meshing can be done in two ways: either refinement (mesh subdivision) can be done once [22], before solution begins, or “multigridding” can be used in conjunction with brightness-weighted refinement, in which the refinement process is affected by the current state of the solution [55, 25]. The latter method is generally regarded as more adaptive and therefore more accurate.

The leading contenders are the last two methods, substructuring and wavelet radiosity, and we do not expect matrix or progressive radiosity without substructuring to compete in terms of speed with the adaptive methods. We anticipate that the wavelet methods may have problems with complex scenes because of the k^2 term in their cost.

1.4. Neglected Techniques

In an attempt to keep the project manageable, we did not attempt to test all of the algorithmic options in the literature. Discontinuity meshing, importance-driven radiosity, and clustering, for example, were neglected. Use of these techniques can have great effect on the speed and accuracy of radiosity solutions, perhaps as significant as the differences between algorithms we have studied. These innovations deserve further study.

Discontinuity meshing takes simple adaptive mesh generation algorithms [11] a step further by attempting to compute accurately where shadow edges and other changes in visibility occur, and create a mesh that resolves the resulting discontinuities in the radiosity function [28, 31, 6]. This can be difficult to implement robustly and it is also not clear whether the cost of discontinuity meshing is worth the extra accuracy gained. Although we have not used discontinuity meshing here, several of the methods we implemented employ moderately adaptive meshing. Our implementations of progressive radiosity with substructuring and wavelets use simple adaptive quadtree refinement.

Importance-driven radiosity focuses computation on those parts of a scene that affect a particular image most, either directly or indirectly [53]. It is only applicable to the wavelet radiosity methods. For maximum generality, we chose to restrict ourselves to the domain of view-independent radiosity methods.

With clustering methods, groups of polygons are clustered together into object hierarchies in order to compute visibility more efficiently [38, 52]. This can be regarded as a generalization of hierarchical techniques that allows scenes with many surfaces (k large) to be approximated, where appropriate, by a scene with a far smaller number of surfaces. These techniques address one of the weaknesses of wavelet radiosity—that its complexity is $O(k^2v)$—although so far they have only been applied to the Haar basis. Related methods partition the scene to improve efficiency [57].

There are other generalizations of the radiosity method that expand its capabilities beyond diffuse surfaces to surfaces with general reflectance [49], and to simulation of the volumetric scattering, absorption, and emission of light in participating media [39], but we have ignored these techniques because they are very different from and much more complex than diffuse surface radiosity algorithms.

It is also possible to simulate diffuse interreflection using ray tracing methods instead of radiosity methods. This is the approach used by RADIANCE, an efficient, free software system for image synthesis [60, 61]. Some tests show that RADIANCE is faster than progressive radiosity on certain scenes [60, p. 464]. It is hoped that others will test RADIANCE on the scenes described in this study to facilitate comparison. Several test scenes for comparing realistic ray tracing and radiosity algorithms have been created by Holly Rushmeier and Greg Ward [37] and by Peter Shirley [47]. In this report we chose to use our own set of scenes so that we could parameterise them by characteristics such as reflectance and complexity.

Finally, no attempt was made to take account of psycho-visual effects, such as the eye's non-linear response to intensity, and variable response to colour [19, 24]. All radiosity calculations were carried out using a standard RGB colour representation, and we used a simple view-independent RMS error measure in which the R, G and B components are weighted equally. We chose this error measure for its simplicity.

The development of perceptual error measures is an area of active research. Since radiosity solutions are typically viewed, rather than used numerically for further computations, subjective quality for human viewers should be the ultimate goal of an image error metric in most cases. A good error measure is useful not only to evaluate the error of the final solution, but also to guide adaptive methods toward more efficient solutions. Ideally, a perceptual error measure should simulate the following properties of human vision: nonlinear response as a function of brightness, nonuniform spectral sensitivity, and variation in sensitivity as a function of spatial frequency. The simplest error measures simulate only the first two effects. If we used such a metric in this study, we do not expect that the ranking of radiosity methods would be greatly affected. Simulation of the third effect is more critical for the evaluation of global illumination algorithms, since it would pick up the step discontinuities and slope discontinuities (Mach bands) that are present in many solutions. However, any measure that varies with spatial frequency is view-dependent, so this would introduce the added complexity of choosing a good set of views from which to evaluate. Error metrics simulating all three of the effects listed above have recently been demonstrated [37].

1.5. Remaining Sections

The rest of this report is structured as follows: in Section 2 we describe those radiosity methods we chose to investigate, and how we implemented them. In Section 3 we present our experimental model, and in Section 4 we present our results. Section 5 draws some high-level conclusions from these results.

2. Radiosity Implementations

There are a number of different ways to implement the algorithms we have mentioned; for our experiments to be reproducible, and to allow the reader to judge the validity of our conclusions, we need to make clear those important decisions we took during the implementation phase. Here we cover the most salient of these for each of the algorithms mentioned in Section 1.3.

In this section we first look at how we implemented the three major classes of algorithms, and then look at the common meshing and form-factor strategies used in all the methods. Most of this section is a summary of previously used techniques, but it is important to be specific about what we have implemented, and explain those innovations or variations we have used. We cover matrix radiosity first, then progressive, and then wavelet radiosity. Finally we cover meshing and visibility for all of the methods. We assume that the reader is familiar with radiosity methods (see [14] for a good introduction).

2.1. Matrix Radiosity

In the matrix radiosity algorithm, firstly meshing is done, then the form-factor matrix is calculated, and finally the system of linear equations defined by this matrix is solved to find patch radiosities. As is standard, we assume radiosities are constant across each element. Variations on matrix radiosity that do not assume this are possible [27, 58], but not common. In our implementation an edge-length parameter `edge_len` controls how many patches the scene is divided into. Each polygon in the scene is first triangulated if it is non-rectangular, and then subdivided into a mesh such that the edge-lengths of all mesh patches are smaller than this parameter. Pseudocode for the matrix radiosity algorithm is given below in **Figure 1**. Note that the `Vis1` function casts a single ray between patches to test visibility.

```
// Do meshing, initialise emission, radiosity vectors, and form-factor matrix.

patches = CreateMesh(polygons, edge_len);
E = EmittedRadiosity(patches);  // vector of length n
B = E;                          // vector of length n
A = IdentityMatrix();           // n x n matrix

// Form the form-factor matrix.

for (i in patches)
     for (j in patches)
          A[i][j] -= FormFactor(j, i) * Vis1(i, j) * Reflectance(i);

// Solve AB = E for B

if (overrelax)
     SolveOverRelax(A, E, B, omega);
else
     SolveConjugateGradient(A, E, B);

// Post-process the mesh by calculating vertex radiosities and smoothing.

SmoothMesh();
```

Figure 1: Pseudocode for Matrix Radiosity

While the matrix corresponding to an integral equation tends not to be highly sparse (unlike those matrices formed when solving differential equations) the prevalence of occlusion in a typical scene and the co-planarity of many groups of patches mean that significant portions of the radiosity matrix will be zero. For this reason, and the large amount of storage required by the radiosity matrix, we have implemented the matrix as a sparse matrix data structure. **Figure 2** shows how matrix density (the fraction of non-zero terms in the matrix) varies with the scene complexity (and hence occlusion and number of surfaces) for the scene pictured in **Figure 17**. In a standard sparse matrix implementation, there is an overhead of an index for each stored (non-zero) element; the break-even point is a density of 0.5 or higher, depending on the real-number format used. Obviously, for non-trivial scenes, such sparse storage is worthwhile.

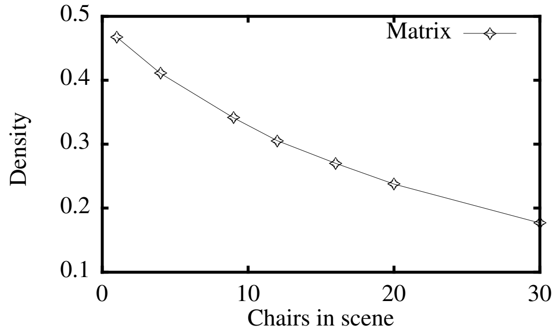


Figure 2: Matrix sparseness vs. scene complexity

2.1.1. Matrix Solution

Matrix radiosity requires the solution of a set of linear equations [56], and has typically been handled with an iterative method such as Gauss-Seidel. The Gauss-Seidel method repeatedly sweeps through the equations in a system of equations $Ab=e$, solving each equation in turn to update an element of the solution vector b. We have chosen to implement two methods which should be faster than Gauss-Seidel:

- Successive Overrelaxation

  Successive overrelaxation tries to accelerate the convergence of Gauss-Seidel by extrapolating by a factor ω the corrections to the vector x at each step. The value of ω should always be between zero and two, and $\omega=1$ yields Gauss-Seidel [56]. Our default is $\omega=1$. Previously, we found that an ω of around 1.4 gives good convergence rates for 'well behaved' radiosity matrices in flatland [29]; its optimal value depends on the eigenvalues of the matrix A.

- Conjugate-Gradient

  The conjugate gradient method solves a linear system by transforming it into the equivalent quadratic minimisation problem, solving the latter by a sequence of carefully chosen steps [56, 46].

  The conjugate gradient method solves symmetric positive definite systems [46], and our matrix A is positive definite, but not symmetric. Nevertheless, we have found empirically that the conjugate gradient method yields solutions as accurate as any other method on all of the radiosity systems we have tested. We do not yet have a theoretical explanation for the success of the conjugate gradient method when applied in this unorthodox manner. Even if the conjugate gradient method cannot be applied to all non-symmetric matrices arising in radiosity problems, it is possible to symmetrize such problems so that conjugate gradients can be applied in an orthodox manner [27, 33, 4, 34].

We investigate the performance of the Gauss-Seidel, successive overrelaxation, and conjugate-gradient methods in Section 4.3.1.

2.1.2. Integral Equations for Radiosity

Radiosity methods find approximate solutions to the integral equation:

$$b(x) = e(x) + \int_{x'} \kappa(x, x') b(x') dx' , \tag{1}$$

where

$$\kappa(x, x') = \rho(x)\frac{\cos\theta\cos\theta'}{\pi r^2}v(x, x'), \qquad (2)$$

and e and b are the emitted and total radiosity at x, ρ is the diffuse reflectance function, r is the distance between x and x', and v is the visibility function.Where two inter-visible surfaces touch, $r = 0$ and the kernel κ has a singularity. Note that, although the kernel may have singularities, the radiosity function itself does not.

2.1.3. Form Factor Calculations

In the radiosity method, the integral equation is discretized to form the system of equations, $b = e + Kb$, or $(I - K)b = e$, where $K_{ij} = \rho_i F_{ij}$, and the F_{ij} are purely geometric coefficients called *form factors*. Typically it is assumed that radiosity is constant across each element. The calculation of *form factors* is a vital part of any radiosity algorithm, and the approach we discuss here is also used in the other radiosity methods [14].

The area-to-area form factor from patch i to patch j is defined as:

$$F_{ij} = \frac{1}{A_i}\int_{A_i}\int_{A_j}\frac{\cos\theta_i\cos\theta_j}{\pi r^2}v_{ij}\,dA_j\,dA_i, \qquad \text{(area-to-area)} \qquad (3)$$

where v_{ij} measures the visibility between the two patches. (We will omit the v term from the remaining discussion.) If the source patch i is emitting a constant radiosity B_i from all points on its surface, then the radiosity received by patch j, when projected into a constant basis, is exactly $\Delta b_j = K_{ij}b_i$. While a closed-form solution for the double integral exists for polygons [44], it is generally too expensive to use in the form-factor calculations of radiosity algorithms. Instead the form factor is usually approximated by the point-to-point form-factor between the centres of the patches:

$$F_{ij} \approx \frac{A_j\cos\theta_i\cos\theta_j}{\pi r^2}. \qquad \text{(point-to-point)} \qquad (4)$$

This approximation works well when the two patches are sufficiently far apart that the inner term of the double-integral remains nearly constant for all pairs of points on the source and receiver patches. However it can fail badly when this is not the case, and when r tends to zero, which occurs when two patches meet or intersect, the point-to-point approximation tends to infinity. In these cases, we switch to the more accurate, albeit more expensive to evaluate[1], polygonal area-to-point form factor [7]. This can be written as

$$F_{ij} \approx \sum_k \frac{\hat{n}_j \cdot (\underset{\sim}{r}_{ijk} \times \underset{\sim}{r}_{ij(k+1)})}{2\pi\|\underset{\sim}{r}_{ijk} \times \underset{\sim}{r}_{ij(k+1)}\|}\theta_{k,k+1}, \qquad \text{(area-to-point)} \qquad (5)$$

where $\underset{\sim}{r}_{ijk}$ is the vector from the point of interest on the receiving patch to vertex k of the source patch, $\hat{n}_j$ is the surface normal at that point, and $\theta_{k,k+1}$ is the angle subtended by vertices k and $k+1$ from that point in radians. (See Appendix A.5 for ccode to implement this equation.)

Detecting situations in which we should switch approximations is straightforward when the two patches are separate; if the point-to-point approximation is larger than a certain threshold, we use the area-to-point approximation. When one patch meets another, however, this method fails, as the projected area of the source patch goes to zero, even though the form factor does not. The left-hand image in **Figure 3** shows the situation; the point-to-point approximation of the form-factor is used here to plot the radiosity function caused by a single source patch lying next to and at ninety degrees to the receiver. This is done by evaluating the point-to-point form factor from the centre of the source patch to many points on the receiving patch. If we use

1. The point-to-point estimate requires 3 dot products to compute. The area-to-point estimate requires 4n dot products, n cross products, and n arctangent calls, where n is the number of edges in the source polygon. For comparison, the closed form solution for the form factor (area-to-area) requires a large number of arctangent, log, and other calls for every pair of source and destination edges. The code for this closed form is approximately 400 lines of C code, as opposed to 20 lines for the area-to-point form, and 5 for the point-to-point version.

the size of this form factor estimate as a threshold, we will fail to switch to the area-to-point formula close to the edge of the emitting patch, as there the point-to-point approximation drops rapidly, in spite of the close proximity of the source patch. To solve this problem, we do not compare the point-to-point F_{ij} to the threshold; instead we compare a modified point-to-point form factor estimate to the threshold:

$$E_{ij} = \frac{A_j(\hat{n}_j \cdot r)\max(\hat{n}_i \cdot r, \mathrm{r\_max}_i)}{\pi r^4}, \qquad \text{(modified point-to-point estimate)} \qquad (6)$$

where $\mathrm{r\_max}_i$ is the radius of a bounding sphere for patch *i*. This ensures the projected area $(\hat{n}_i \cdot r)$ of the source does not drop below a certain, size-invariant threshold, and thus the modified form factor will increase, rather than decrease, close to the edges of the source patch. Note that this formula is not attempting to be accurate when θ_i is near 90 or r is small; it is only attempting to ensure that the area-to-point form factor will be used in such cases. When θ_i is small and r is large, Equation 6 gives the same results as Equation 4. Pseudocode for our estimated form factor is given below:

```
FormFactor(i, j) :
     //   Calculate the form factor between patches i and j
     if (ModifiedEstimateFF(i, j) < threshold)      // Equation 6
          return(PointToPointFF(i, j));             // Equation 4
     else
          return(AreaToPointFF(i, j));              // Equation 5
```

Figure 3 shows, from left to right, `PointToPointFF`, `AreaToPointFF` and `FormFactor` for two abutting patches. Barely noticeable in the corrected function at right is a slight slope discontinuity where the changeover from the point-to-point to the area-to-point form factors takes place. Our software uses `FormFactor` by default, but when extra accuracy is required, we can use `AreaToPointFF` in all cases.

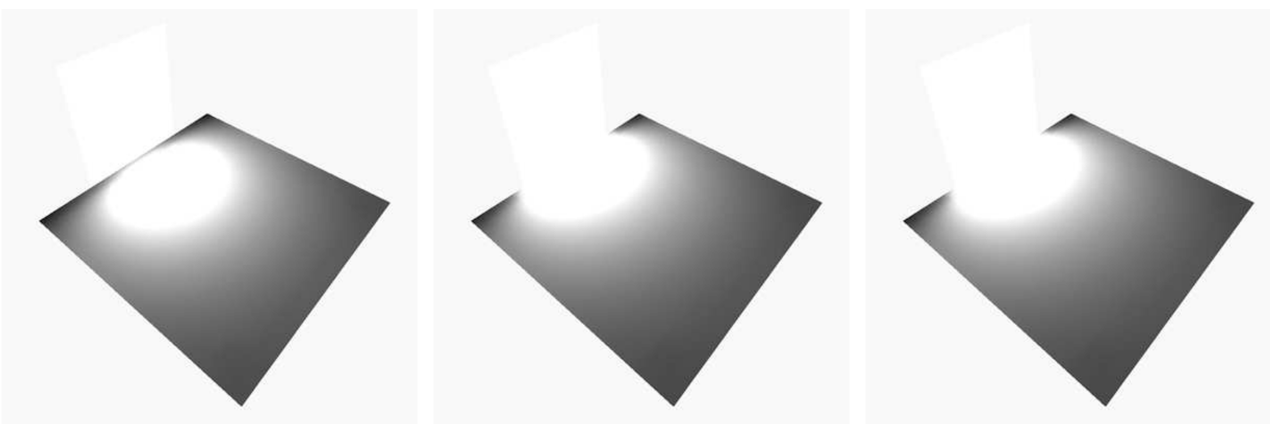


Figure 3: Abutting patches with simple point-to-point form factors (left), accurate, area-to-point form factors (middle), and point-to-point form factors with area-to-point correction close to the source patch (right).

2.2. Progressive Radiosity

We generate the mesh for progressive radiosity in the same way as for matrix radiosity. The standard algorithm for progressive radiosity is used [12] — on each iteration we shoot radiosity from the patch with the largest unshot power (the product of the patch's unshot radiosity and its area) to the rest of the scene's patches. This process proceeds until the unshot power of the next patch to shoot drops below a preset fraction, `t_ratio`, of that of the first patch shot. This fraction is the *termination error*, and once it has been reached we refer to the algorithm as having *converged*.

We also optionally add in an ambient radiosity term to all patches before display, to correct for the unshot radiosity still in the scene. This ambient radiosity is given by multiplying the current sum of unshot radiosity by the *environmental reflectance*, $R = (R_r, R_g, R_b)$ [14], where:

$$R_c = \left(1 - \frac{\sum_i \rho_{ic} A_i}{\sum_i A_i}\right)^{-1}, \tag{7}$$

and $\rho_i = (\rho_{ir}, \rho_{ig}, \rho_{ib})$ is the (RGB) reflectance of patch *i*. The effect of this term is investigated in Section 4.3.3.

The progressive radiosity algorithm is summarised below:

```
// Initialise meshing, unshot radiosity and radiosity vectors

patches = CreateMesh(polygons, edge_len);
S = EmittedRadiosity(patches);  // unshot radiosity: vector of length n
B = S;                          // vector of length n

// Iterate shooting steps

while (true)
      i = FindMaxPowerIndex(patches);
      radToShoot = S[i];
      S[i] = 0;

      for (j in patches)   // Shoot from patch i to patch j
            B[j] += radToShoot * FormFactor(i, j) * Vis1(i, j) * Reflectance[j];
            S[j] += radToShoot * FormFactor(i, j) * Vis1(i, j) * Reflectance[j];

      if (firstIteration)
            firstPower = area(i) * radToShoot;
      else if (area(i) * radToShoot < firstPower * t_ratio)
            return;
```

Figure 4: Pseudocode for Progressive Radiosity

2.3. Progressive Radiosity with Substructuring

Progressive radiosity can be extended by allowing the shooting and receiving meshes to differ in density, and also by adaptively refining the receiving mesh [12]. In our implementation of substructuring, the density of the shooting mesh is specified in a similar manner to matrix radiosity. The patches in this mesh are hierarchical, and are subdivided until they meet a second, smaller, edge-length limit `edge_len2`, which dictates the initial number of elements in the receiving mesh. (We will refer to the subareas in regular meshes as *patches*, and those in adaptive meshes as *elements*.) Progressive radiosity is performed on these two meshes, and after each shooting step, any element which has a large radiosity gradient is subdivided further (See **Figure 43**). If any such subdivision takes place, the shooting step is undone and then repeated. There is also a minimum edge length `edge_min`, below which elements are not subdivided, to prevent excessive or unnecessary meshing.

We use the standard deviation of the RGB radiosities at each of an element's vertices to approximate its gradient. These vertex radiosities are calculated by averaging the radiosities of all patches that meet at a vertex in the mesh, and are also used for displaying the smooth-shaded version of the mesh. This gives us the error measure shown in Equation 8, where v_i is the radiosity at vertex i, and *n* the number of vertices of the element. If this error is above a preset `epsilon`, the element will be subdivided.

$$\text{gradient} = \sqrt{\frac{1}{3}\sum_{c \in \{r,g,b\}}\left[\frac{1}{n}\sum_i v_{ic}^2 - \left(\frac{1}{n}\sum_i v_{ic}\right)^2\right]} \qquad (8)$$

Figure 5 shows the differences between the vanilla progressive radiosity algorithm, and the substructuring version.

```
// Initialise meshing, unshot radiosity and radiosity vectors

patches = CreateMesh(polygons, edge_len);
elements = CreateHierarchicalMesh(patches, edge_len2, edge_min);
S = EmittedRadiosity(patches);  // unshot radiosity: vector of length np
CopyToElements(S,B);            // B is a vector of length ne

// Iterate shooting steps

while (true)
      i = FindMaxPowerIndex(patches);
      radToShoot = S[i];
      S[i] = 0;

      for (j in elements)  // shoot from patch i to element j
            B[j] += radToShoot * FormFactor(i, j) * Vis1(i, j) * Reflectance[j];
            S[B[j].parent] += radToShoot * FormFactor(i, j) * Vis1(i, j) * Reflectance[j];

      // Refine

      subdivOccurred = false;
      do
            for (j in elements)
                  if (gradient(j) > epsilon)
                        subdivisionOccurred = true;
                        B[j] -= radToShoot * FormFactor(i, j) * Vis1(i, j) * Reflectance[j];
                        subdivide(j);
                        for (k in children(j))
                              B[k] += radToShoot * FormFactor(i, k) * Vis1(i, k) *
                                    Reflectance[k];
      until (!subdivOccurred);

      if (firstIteration)
            firstPower = area(i) * radToShoot;
      else if (area(i) * radToShoot < firstPower * t_ratio)
            return;
```

Figure 5: Pseudocode for Substructuring Radiosity

2.4. Wavelet Radiosity

We implemented wavelet radiosity as described by Gortler et al. [22], although there were a number of areas where either the description was not complete enough to follow, or we diverged from the described approach; we describe those areas here. In particular, we only stored the smooth coefficients of the wavelet functions in our representation; the wavelet coefficients were implicit, in contrast to the approach taken in [9]. We used only the brightness-weighted refinement version of the algorithm for our results, as experiments with the refine/solve approach convinced us it was vastly inferior [25]. The bases we implemented were Haar, the F2 and F3 flatlets, and the M2 and M3 multiwavelets. The one-dimensional versions of these are plotted in

Figure 6. Note that whereas the flatlet bases are disjoint, the multiwavelets are cumulative: M1 is just the multiwavelet $\{\Phi_0\}$, M2 consists of $\{\Phi_0, \Phi_1\}$, and M3 is $\{\Phi_0, \Phi_1, \Phi_1\}$. Also, none of these basis functions is continuous, so unless post-process smoothing is done, the radiosity function will not in general be C^0, which can be visually disconcerting.,

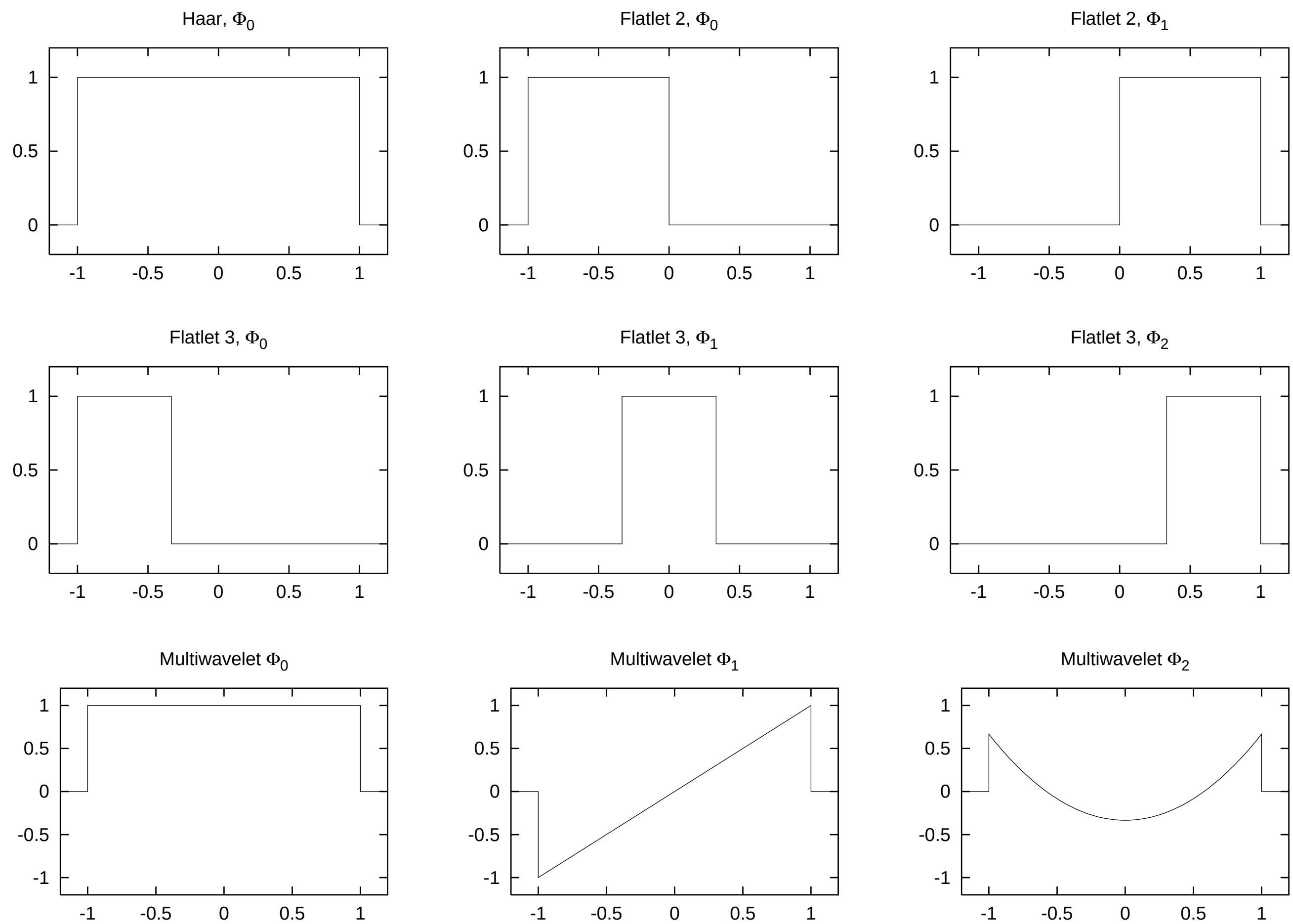


Figure 6: The flatlet and multiwavelet basis functions

As with the other algorithms, we do not attempt a tutorial here; for a general introduction see [55, 48, 14]. For more specific material relevant to our implementation, see [22, 42].

There are three important parts to the wavelet radiosity algorithm: the control algorithm, the radiosity push-pull and inter-patch transfer routines, and the various oracles. The coefficients used in all wavelet push, pull, and transfer operations are listed in Appendix A.1. We discuss important aspects of the other two parts of the algorithm in the rest of this section.

2.4.1. Wavelet Control Algorithm

We can either refine the mesh and then solve it in two separate passes, as described in [22], or use a multigrid approach where refinement and solution steps are intermixed [55]. When intermixing the refinement and gather steps, we wish to avoid over-refining the mesh before we have performed enough solution steps for the current radiosity estimates to be a good guide to subdivision. We accomplish this by allowing a link in the scene to be refined at most once during on each refinement pass. This seems to work well. An alternative solution proposed by [14] is to start the form-factor error limit at a higher value than the desired final value, and then decrease it slowly according to some schedule. Pseudocode for the wavelet algorithm is as follows:

```
trees = MakeTrees(polygons);    // Create one quad-tree for each base polygon

// Create initial links

for (i in trees)
      for (j in trees[1..i-1])
            if (Vis16(i, j) > 0)
                        CreateLinkBothWays(i, j); // link from patch i to j, and vice-versa

// Iterate, alternately solving and refining.

while (true)
      // One solution step
      resError = 0;
      for (i in trees)
            trees[i].Gather();              // gather radiosity across the links
      for (i in trees)
            resError += trees[i].PushPull();// convert gathered radiosity to wavelet basis

      // Refine mesh
      subdivOccurred = false;
      for (i in trees)
            subdivOccurred |= RefineFurther(epsilon, edge_min);

      if (firstIteration)
            origErr = resError;
      else if (!subdivOccured && resError < t_ratio * origErr)
            return;
```

Figure 7: Pseudocode for Wavelet Radiosity

In this code `Vis16` estimates visibility by tracing 16 rays, as in Section 2.6. The Gather stage is separated from the Push/Pull stage, as in [25] and [14]. While it is tempting to merge Gather into the Push/Pull stage to save storage[1], we have found that this leads to stability problems for the first few iterations of the algorithm, when the number of elements is small.

2.4.2. Quadrature

Higher order wavelet bases require a generalisation of the standard form factor to account for the non-constant basis functions. The kernel must be integrated against each possible pair of source and receiving basis functions to produce the transfer coefficient between the two. This integration is carried out by various quadrature methods, which use a weighted sum of kernel samples to estimate the integrals, as described in Gortler et al. [22]. We will discuss here our handling of visibility and singularities when calculating the transfer coefficients.

Visibility

We tried two quadrature methods for the higher order wavelets, *fractional visibility*, which scales the results of quadrature with the estimated visibility between two patches [25,22], and *visibility-in-quadrature*, which leaves visibility in the integrand [16]. For the former, we treat visibility by using Hanrahan's visibility estimate (`Vis16`) to scale the results of quadrature, as described in Section 7.4 of Gortler et al. This is analogous to the original hierarchical radiosity algorithm, where the fractional visibility between two patches was estimated, and used to weight the transfer of radiosity. Thus the integration is done assuming no occlusion, and then scaled by the fractional visibility (`Vis16`). For the latter, visibility is sampled for each quadrature

1. This would be the equivalent of Gauss-Seidel iteration, where the radiosities are updated in place. The algorithm as presented is similar to Jacobi iteration.

point, and incorporated directly into the integration. Most of our experimental results for the wavelet methods use fractional visibility, as we only became aware that visibility-in-quadrature had been used in the literature towards the end of the project.

Computationally, our implementation of fractional visibility requires tracing 16 rays for each link, using a jittered 4×4 grid on both patches, regardless of basis order. For visibility in quadrature, the quadrature is computed by a summation of M^4 terms corresponding to a non-jittered $M \times M$ grid of sample points on each patch. For first order methods, a single ray would be traced, for second order, 16 rays (2×2 grid), and for third, 81 rays (3×3 grid).

Quadrature In The Presence of Singularities

A singularity occurs where two patches meet; e.g., in the corner of a room where they share an edge, and the denominator of the kernel goes to zero. When using multiple-point quadrature to calculate coupling coefficients for the multiwavelet basis, the presence of the singularity in the kernel can cause problems, because the function diverges from the low-order-polynomial assumption, as detailed in [64]. That paper suggests switching to the use of the Gauss-Jacobi basis functions in such situations to avoid the problems caused by singularities in the radiosity kernel. This eliminates the singularity in the radiosity kernel by projecting the multiwavelet basis into the same set of basis functions, multiplied through by a quadratic centred on the edge along which the singularity occurs. **Figure 8** shows the Gauss-Jacobi basis functions. Although this approach avoids quadrature error in the integration/transfer coefficients step, this comes at the expense of projection error, as most of the time the radiosity over a patch is close to constant, and a constant function is poorly fitted by the Gauss-Jacobi basis. Although the integration itself becomes more accurate, there is still a considerable source of error in the transfer coefficients.

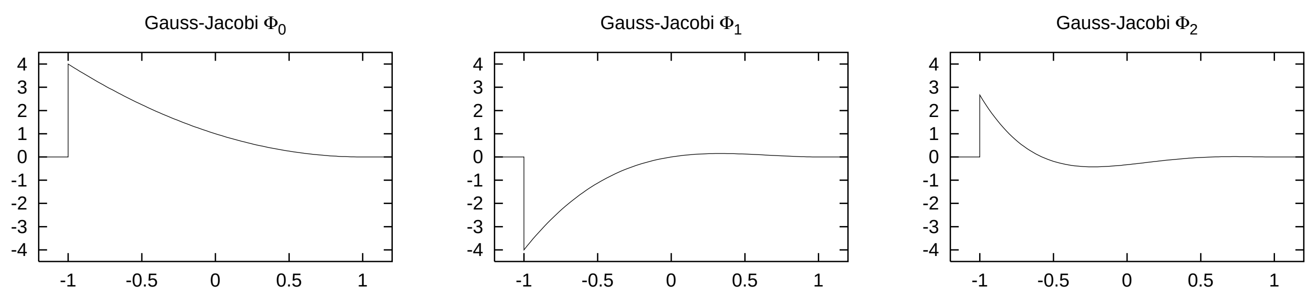


Figure 8: The Gauss-Jacobi basis functions

Schröder details an alternative approach to this, which he calls "C-C transfer functions" [41]. We used this approach rather than using the Gauss-Jacobi basis. In situations where singularities arise we project the multiwavelet basis into the equivalent flatlet basis, and then just use the flatlet transfer scheme, which handles singularities by using the analytical patch-to-point form factor function. The flatlet basis is a much better fit for the multiwavelet bases, and we already have a method for eliminating singularity concerns in flatlet couplings, as described above. The scheme also has the advantage of being much simpler to implement than the Gauss-Jacobi approach, as it reuses much of the flatlet code. Finally, [41] demonstrates that in general the C-C transfer rule gives equivalent experimental results to the Gauss-Jacobian rule.

To decide when an interaction has a singularity, we test for patches that touch along at least an edge during the initial linking phase; the link between two such patches is flagged as containing a singularity. Whenever such links are subdivided, the test must be repeated for each of the sublinks. Any radiosity transfer between patches flagged in this manner uses the C-C transfer rule.

2.4.3. The Wavelet Radiosity Oracle

For each of the wavelet bases we must be able to produce an estimate of the error in the radiosity transfer between two patches if a link were to be formed between those patches. We can then use this estimate to force all interaction errors below a certain limit; by doing so, the error in the radiosity solution can be forced below any chosen value, as described in [42].

For patches using the Haar basis, we used the error oracle outlined in [25], where it is shown that when the patches are separated, the form-factor itself is a good measure of the interaction error. Thus if the form factor estimate (weighted by the amount of radiosity being transferred across the link) is greater than `epsilon`, the link is subdivided. We choose which end of the link to subdivide based on the projected area (cosine of the angle to its normal times its area) of the source and receiver from the point of view of the other patch. If the source has the largest projected area, new links are formed between its children and the receiver, and vice versa. This subdivision strategy is also employed for the flatlet and multiwavelet bases.

In the original hierarchical radiosity paper, visibility does not affect patch refinement [25]. Cohen and Wallace [14] suggest that visibility could be used to modify the form-factor epsilon value for refinement, so that links with partial visibility are forced to subdivide more than completely unobscured or obscured ones. We follow that approach here; for links with partial visibility, epsilon is multiplied by a fraction `vis_epsilon`. Experiments led us to use a value of 0.15 for this parameter for the work presented in this report. The oracle itself does not weight error estimates with the visibility across the link, as this leads to insufficient subdivision for hard shadows. We also use the *triage* optimisation suggested in the original paper; if a link is either completely obscured or completely unobscured, visibility is not recalculated when it is further refined.

For a flatlet basis of order M, each of the $M \times M$ transfer coefficients in the link is identical to a normal Haar link. To calculate the error in the interaction as a whole, we take the maximum of the coefficient errors between each pair of sub-elements in the receiving and source flatlets, thus ensuring all of these transfer coefficients are below a certain error bound.

The multiwavelet bases provide a more difficult challenge. There are currently no simple geometrical error estimates for the multiwavelet bases as there are for the Haar basis. The approach detailed in [25] is to estimate the error in a link by estimating how far the kernel deviates from a polynomial of order M. This is done by using the quadrature samples to construct an interpolating polynomial of that order, and then measuring the difference between this and the real kernel by sampling at points distinct from these.

This interpolating approach has a number of drawbacks, which include:

- In the case of brightness-weighted refinement, it decouples the transfer coefficients from the source patch's radiosity coefficients.

  Thus the link error from a source patch with a constant radiosity and one with radiosity increasing quadratically from top to bottom are treated identically. In practice we could afford more error in the transfer coefficients in the case of the constant source patch, as this frees up extra degrees of freedom for integrating the kernel in our quadrature rules.

- The error estimate is more expensive in terms of operations per coefficient than the Haar or Flatlet bases.

It is possible that better methods can be developed which both address the first shortcoming, and produce comparable error estimates with less work. In the future we hope to compare this approach to alternatives in the literature [3], and our own measure, both of which estimate the error between the radiosity function over the receiver and the function after it has been projected into the basis functions of the receiver.

We have chosen to use the outlined approach, however, as it is the standard approach in the literature. Instead of using Nevilles's algorithm [36] to construct the polynomial, we calculate the samples of the interpolating polynomial from the quadrature samples with a single $M \times M$ matrix multiply, using a pre-computed transformation matrix.

2.4.4. Termination Criteria

The multigrid wavelet methods, like the progressive radiosity method, refine their solution over time once the initial linking phase is complete. Though there is no direct way to measure the error in the current solution, we can use the sum of changes in the radiosity of each of the base patches in the scene to estimate the delta between the previous iteration's solution and the current one. This gives us an estimate of the residual error between the current solution and the converged solution (which will typically not be the same as the true solution because of the finite mesh size), and will tend to zero as the solution converges.

- For a two-stage wavelet algorithm, we would iterate in the solution stage until our estimated residual error dropped below a set fraction, `t_ratio`, of the first iteration's residual error.

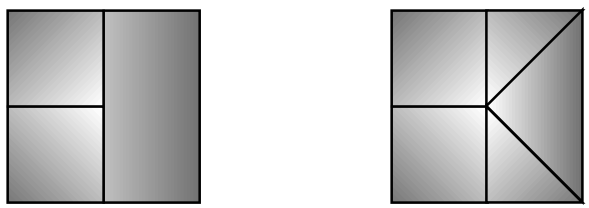
- For the multigrid algorithm, we wait until no further link subdivision takes place, and then until the error ratio is satisfactory, as above.

In practice we could probably terminate the multigrid algorithm before all links have been subdivided to the proper level if the error was low enough. We avoid doing so in our experiments, partly so we can gain insight into when this would be useful, and partly because our previous use of such an approach has led to variable results. The residual error estimate has enough noise that it is hard to pick termination limits that will consistently lead to termination in a ‘good’ place.

2.5. Meshing

Initial Meshing

Rectangle/triangle meshes are used for all the radiosity methods [14]; that is, all meshes are formed from inter-connected rectangles or arbitrarily-shaped triangles. As described previously, the meshes can either be evenly divided into a grid, such that the edge-lengths of all patches are below a user-controlled threshold, or subdivided hierarchically, in a quadtree [40]. For hierarchical meshes a balanced mesh and anchoring are used to ensure the absence of T-vertices [6], which can lead to shading discontinuities, as in **Figure 9**. A *balanced* hierarchical mesh is one in which the level of subdivision of a leaf element differs from that of its neighbours by at most one. An *anchored* mesh is one that eliminates T-vertices by triangulating leaf nodes. An example of such a mesh can be seen in **Figure 10** and **Figure 11**.



Ordinary mesh Anchored mesh

Figure 9: Shading discontinuities at a T-vertex

Post-processing

Haar wavelets, flatlets, and our implementations of matrix and progressive radiosity produce a piecewise constant radiosity function. The second and third order multiwavelets (M2 and M3) produce piecewise linear and piecewise quadratic functions, respectively. Traditionally, all of the piecewise-constant methods except flatlets are post processed for Gouraud shading, to make their solutions continuous and smoother looking [12, 13, 25]. We use the same approach. For multiwavelets, we display directly from the basis functions, as did Gortler et al. [22]. The resulting functions will thus be piecewise linear or piecewise quadratic, with possible discontinuities between elements. Flatlets are first converted to the same order multiwavelet basis, and then displayed in a similar manner. They may also exhibit discontinuities. The radiosity function resulting from post processing is used for both error measurement and display. The details of this post-processing are described below.

The first-order meshes (those for matrix and progressive radiosity, and wavelet radiosity with the Haar basis) are smoothed by calculating radiosity RGB values for each vertex of the mesh, and then using those values to Gouraud-shade the mesh. A vertex’s radiosity is calculated by averaging the radiosities of all patches or elements that share it. The displayed radiosity functions will therefore be continuous in value (but probably not in derivative).

We chose to smooth flatlet solutions, even though they were not smoothed in the original wavelet radiosity paper [22], because we felt that not to do so would put them at a disadvantage to the Gouraud-shaded Haar basis, and the multiwavelets. Complete smoothing is not easy, however. While we could apply the T-vertex avoidance methods to the constant-shaded sub-elements of each flatlet, this would require considerable special-case coding for each different flatlet basis. We chose instead to project the flatlets into their equivalent multiwavelet bases before smooth-shaded display. This makes them piecewise linear or quadratic, and allowed us to reuse the multiwavelet drawing and sampling code. The drawback of this approach is that continuity between elements is not enforced. This change of basis is cheap, however; the transformation to the multiwavelet basis

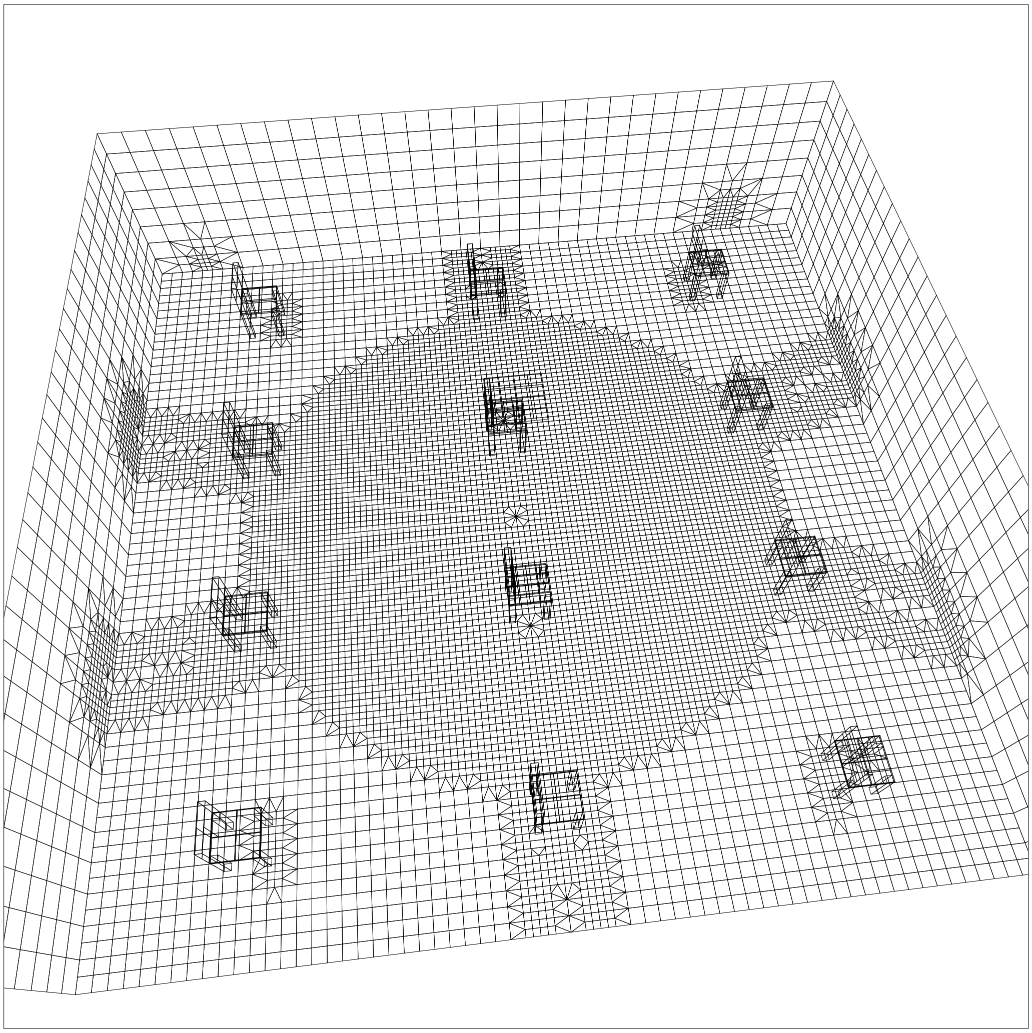


Figure 10: Substructured mesh with T-meshing, wire-frame view.

requires only a tensor multiplication (two 2×2 or 3×3 matrix multiplies) as opposed to a full basis transform (which would require a 4×4 or 9×9 matrix-vector multiply.)

The flatlet and multiwavelet meshes are not anchorable, due to the added implementation complexity this would entail, but they are balanced to ensure that the number of inter-element discontinuities that do occur are minimised. For these higher order methods, we could enforce continuity everywhere, as we do when Gouraud-shading the constant basis functions, and this would make shadows look much smoother, but it would not necessarily be more accurate. Based on experiments in 2-D, post-process smoothing of radiosity solutions does not reduce their objective error relative to piecewise constant solutions [29].

2.6. Visibility

Ray-casting is used for all inter-patch visibility tests, in order to standardize visibility testing between algorithms. To compute a form factor, two visibility methods are used:

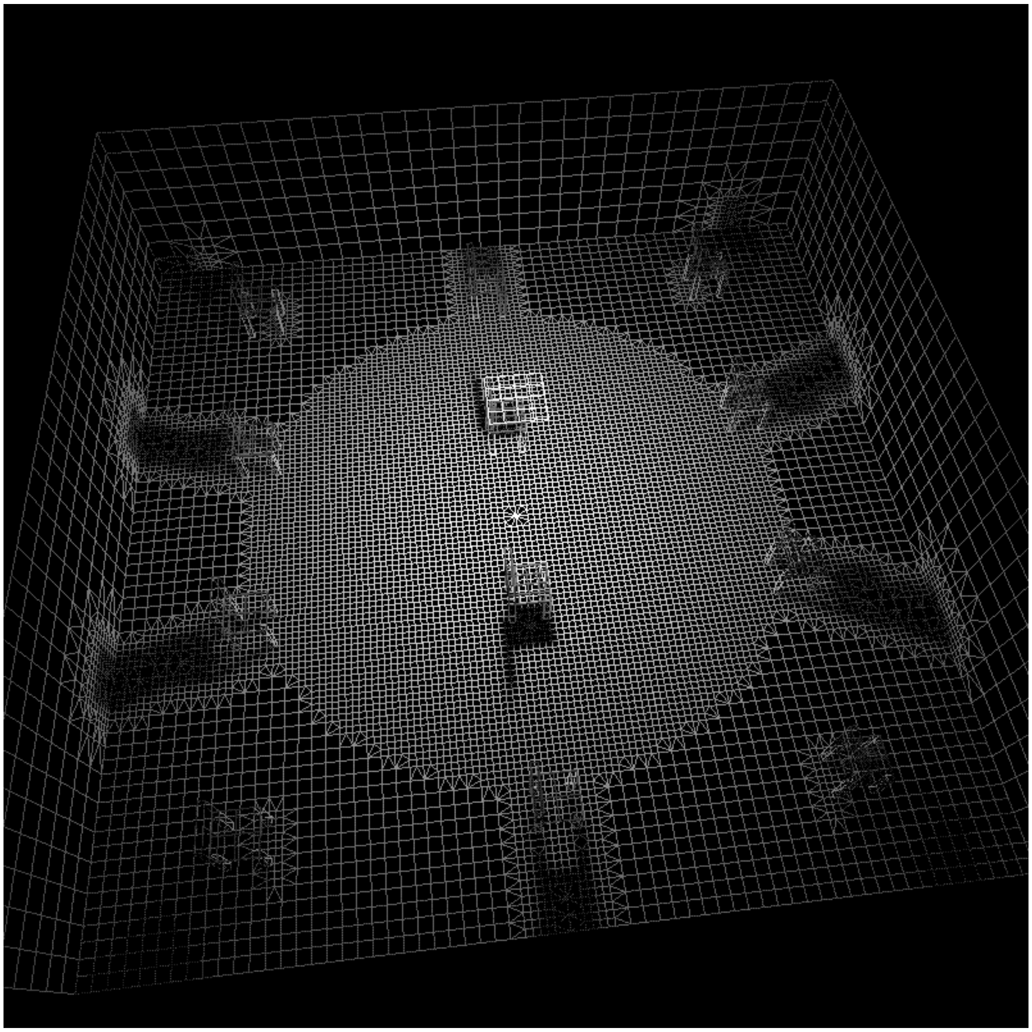


Figure 11: Substructured mesh with T-meshing, shaded view.

- `Vis1`: For matrix and progressive methods, a cheap technique using a single ray cast between the centres of the source patch and destination patch. 1 is returned if the ray does not intersect any other object, and 0 if it does.
- `Vis16`: For wavelet methods, 16 jittered rays are cast between the source and destination patches as described in [25]. The number of unoccluded rays over 16 is then used to estimate the fractional visibility between the patches. This is then used to scale the quadrature samples, as in [22].

Rays are cast only if it can be determined that the patches face each other, that is, at least one vertex of the source patch lies in front of the plane of the destination patch, and vice versa. A Fujimoto grid scheme [15] is used for ray-casting acceleration; the scene is spatially subdivided into a lattice of voxels, and a Bresenham-like algorithm used to follow the path of each ray through this grid. When using such a scheme, the grid size is typically chosen such that the number of voxels is $O(k)$, which leads to an expected visibility cost of $v = O(k^{1/3})$ for reasonably homogeneous scenes [10]. For simplicity we chose a grid size large enough to efficiently ray trace our most complex scene—the maximum dimension is always 40 voxels—and used that size for all our scenes, so that our expected visibility cost is $v = O(k)$, albeit with a very low constant. Finally, our scenes consist of planar polygons only—there are no curved surfaces.

2.7. Code Structure

The algorithms mentioned above have been implemented in a common, object-oriented testbed. And as much code is shared as possible, in order to bring out algorithmic differences in the algorithms, and reduce implementation-dependent ones. For instance, the differences between the wavelet methods implemented are completely encapsulated by a few methods of the link and patch classes; all the wavelet control code is shared. (The current design also allows for the intermixing of different wavelet basis functions in the same scene, although we have not yet explored this capability beyond mixing triangular and rectangular basis functions from the same family.) The wavelet quadtree classes build on those used in the substructuring mesh, which in turn build on the vanilla patches used by the progressive and matrix radiosity algorithms.

3. Experimental Method

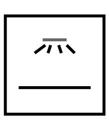
3.1. Experiment Scenes

Before constructing a set of test scenes, we decided on those properties of the radiosity algorithms that we wished to investigate, namely:

1. **Radiosity Distributions**
   We wished to test how well a particular radiosity method could reproduce the radiosity function over the surfaces of the scene, especially in areas of high gradient, such as when a light source is close to a reflective surface.
2. **Interreflection**
   As the reflectance in a scene increases, the amount of indirect illumination grows. This will affect the running times of some algorithms more than others. For instance, as the average number of bounces of light through the environment increases, the progressive radiosity algorithms do more work to distribute light through the scene. Also, highly reflective scenes act as a stress test for the methods; under such conditions we expect some of them to become unstable. They also tend to highlight problems methods have with the singularities in the radiosity kernel.
3. **Occlusion**
   With area light sources in a scene, shadows can range from sharply discontinuous, when the shadowing object is close to the receiver, to smooth penumbras, when it is closer to the emitter. Shadows are notoriously hard to simulate in radiosity methods. Handling them well is mostly a matter of good meshing.
4. **Scene complexity**
   The geometrical complexity of the scene (for instance, the number of primitives, and their positions) will obviously affect the running time of the algorithms. Also, the number of light sources in the scene will affect the convergence times to a greater or lesser extent, depending on the method employed.

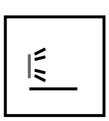
To test these properties, we constructed a number of experiments, each of which consisted of a scene, and some parameter of that scene which could be varied. The radiosity algorithms were then run against each of the scene variations in turn. The next section describes these experiments.

Each experiment was composed of a scene, and a scene parameter which was varied in order to test a particular quality of the radiosity algorithms. This parameter affected either separation, surface reflectance, or scene complexity. The following is a description of the six experimental scenes we used; the scenes themselves are shown from **Figure 12** to **Figure 18**.



- Parallel Experiment

As a simple test of direct illumination (without interreflection) we constructed a scene which consisted of two square patches; one a light emitter, and one a light receiver. The scene parameter was the separation of the two patches, which is variable between 0 and 2. Two example configurations of the scene are shown in **Figure 12**. This configuration of patches produces largely smooth radiosity gradients, except when the patches are close.

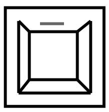


- Perpendicular Experiment

To investigate the effects of singularities in the radiosity kernel we place the light source and receiver perpendicular to each other, and again vary the distance between them. In this case, when the separation reaches zero, the light source and receiving patch touch, and a singularity lies along their common edge. In this case the measures outlined in Section must be used. **Figure 13** shows the light source at two different separations for this experiment.

- Blocker Experiment

In this scene a ‘blocker’ polygon is added to the parallel scene in order to test visibility handling. This occluder is reflective on the top, but black on the bottom, so there is no interreflection. The scene parameter in this case is the height of the blocking patch (its distance from the receiver). As this height is reduced to zero, the shadow edges approach a step discontinuity. As it increases towards two, the umbra shrinks, and eventually vanishes, leaving only the smooth penumbra. We would expect different algorithms to handle these two extremes with varying levels of accuracy. **Figure 14** shows the blocker positioned close to the emitter, and close to the receiver.

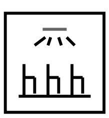


- Box Experiment

To test how high levels of inter-patch reflection affected the algorithms, we used a scene similar to the famous ‘Cornell box’ [20], and varied the reflectance of the walls. Of particular interest in this setup was the multiple bounces of light between two patches meeting at a corner, and the effects of high reflectance on stability in such a highly-coupled situation. **Figure 15** illustrates this scene. Note that the radiosity of the light source was scaled inversely with reflectance, so that the scene brightness remains approximately constant.

- Tube Experiment

To test how the algorithms handle long chains of radiosity bounces between different patches, we constructed a long tube, with a light source at one end of it, as can be seen in **Figure 16**. In our initial experiments this scene contained baffles in order to restrict reflections to one long chain of patches, but this lead to too little propagation. Removal of the baffles allowed multiple-bounce reflection between adjacent pairs of patches, but the geometry of the scene ensures that this is small compared to the magnitude of multi-bounce paths of radiosity. The scene parameter controls the reflectance of the tube; the higher this becomes, the more the light propagates down the tube. (In this experiment, the radiosity of the light source is held constant.) This is also a good stress test for susceptibility to over-approximation of form factors, as most light transfer takes place at extremely close quarters. To make visual inspection of solutions easier, radiance is displayed on both sides of the wall polygons.



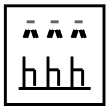
- Complex Experiment

To measure the effect of scene complexity we constructed a scene consisting of a room, a single overhead light source, and a variable number of chairs. **Figure 17** shows two examples of the scene. Though the number of surfaces in the scene could be varied widely, the visual complexity of the scene did not vary that much, as the chairs, and their contribution to the illumination of the room, are relatively insignificant compared to the large walls and the floor. In our experiments we used 4, 6, 9, 12, 16, 20, 25, 30, or 36 chairs. We wished to test the algorithms on more complex scenes, but beyond this size scene (around 1000 polygons) all but the progressive methods were straining the memory limits of our hardware. (The total number of polygons in a complex scene with m chairs is $5 + 27m$.)

There are a number of possible approaches to the construction of this scene:

- The room can be held at a constant size while an increasing number of chairs is packed into it. This is an *unscaled* complex scene.
- It can be scaled horizontally so that all chairs remain equidistant (which is equivalent to replicating an initial ‘cell’ containing a single chair). This is a *horizontally-scaled* scene.
- It can be scaled both horizontally and vertically, so that the surface area of the room grows as the number of chairs. We refer to this as the *scaled* complex scene, or simply as the ‘complex scene’.

Cleary et al. discuss scaled and unscaled scenes in a theoretical study [10]. We chose the third alternative for almost all of our experiments. This way of scaling the scene can be justified by analogy to a large auditorium, where ceiling height typically rises as the capacity of the hall is increased. It is also analogous to a terrain in which surface details are refined in a fractal manner: as details are added, they are scaled down.



- Lights Experiment

We used a similar scene in a second experiment to measure the effect of light source complexity on the algorithms. This time the number of chairs was held constant at twelve, while progressively more lights were added to the room, as in **Figure 18**. This not only increased the number of primary radiosity sources, affecting the progressive radiosity algorithm, but increased the complexity of the shadows cast on the floor. We used 1, 4, 9, and 16 lights in our tests.

Table 1 summarises the experiments and the parameters that were varied within each experiment.

Experiment	Parameter Varied	Property Tested
Parallel	Patch separation	Distribution
Perpendicular	Edge separation	Distribution
Blocker	Blocker height	Occlusion
Box	Reflectance	Corner interreflection
Tube	Reflectance	Multi-bounce interreflection
Complex	Number of chairs	Geometrical complexity
Lights	Number of lights	Complexity in number of light sources

Table 1. Experiment Summary

Each experiment also required a set of method parameters for the radiosity algorithms investigated. The standard values for these parameters are listed below in **Table 2** below, and the values used for specific experiments are shown in **Table 3**. It should also be noted that for those scenes where there was no occlusion, the ray-caster was turned off.

Parameter	Purpose	Default value
`edge_len`	The original input polygons are meshed so that all patches have an edge-length of this size or less.	0.25
`edge_len2`	For the progressive radiosity with substructuring method, the receiving mesh is initially subdivided so that all elements have an edge-length of this size or less.	same as `edge_len`
`epsilon`	Controls the refinement of links in wavelet radiosity, and the subdivision of elements in progressive radiosity with substructuring.	0.05
`t_ratio`	The termination error ratio for progressive and wavelet radiosity.	0.001
`edge_min`	The minimum edge-length for elements; elements smaller than this are not subdivided further.	0.01

Table 2. Radiosity Method Parameters

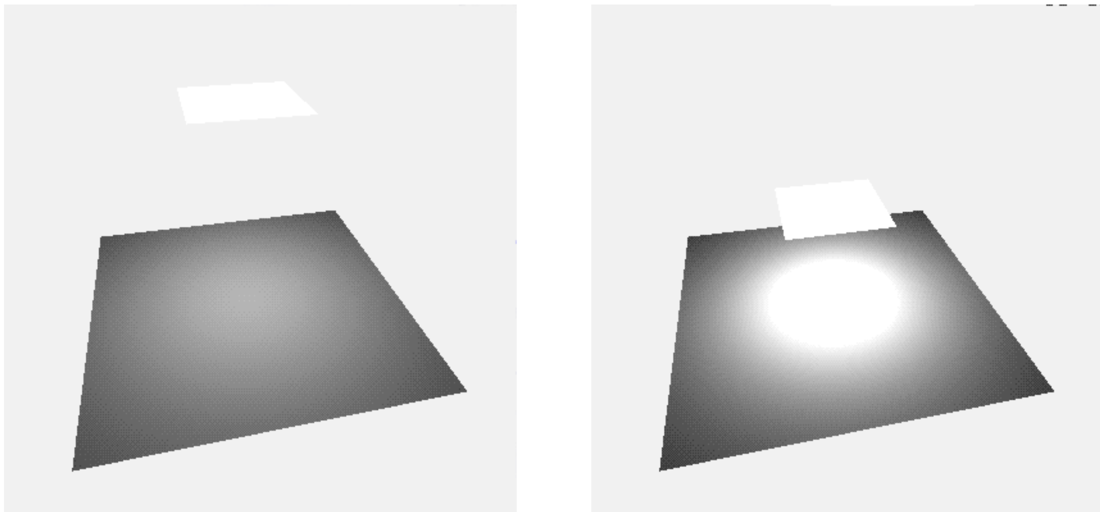


Figure 12: Parallel Experiment. Separation of 0.8 and 0.2.

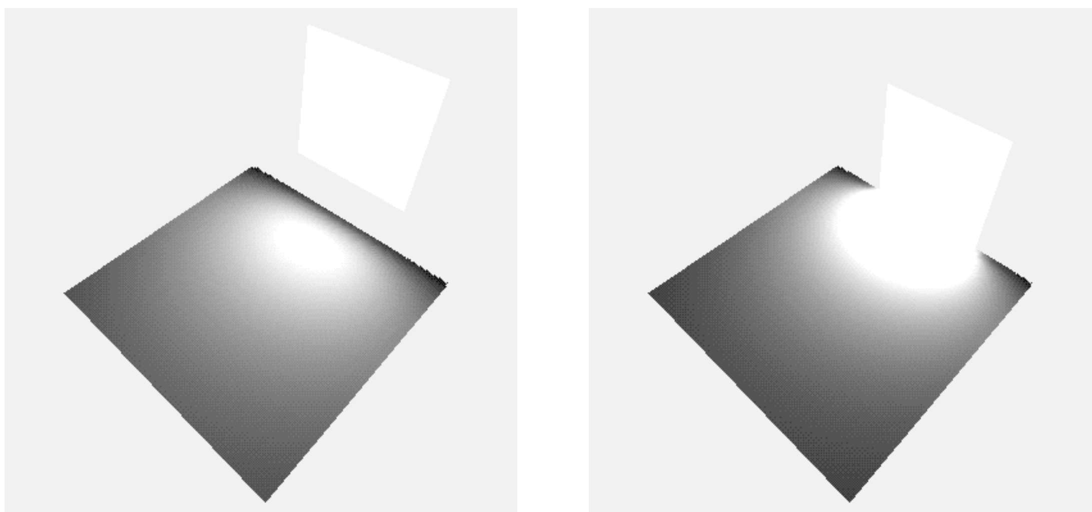


Figure 13: Perpendicular Experiment. Separation of 0.4 and 0.0.

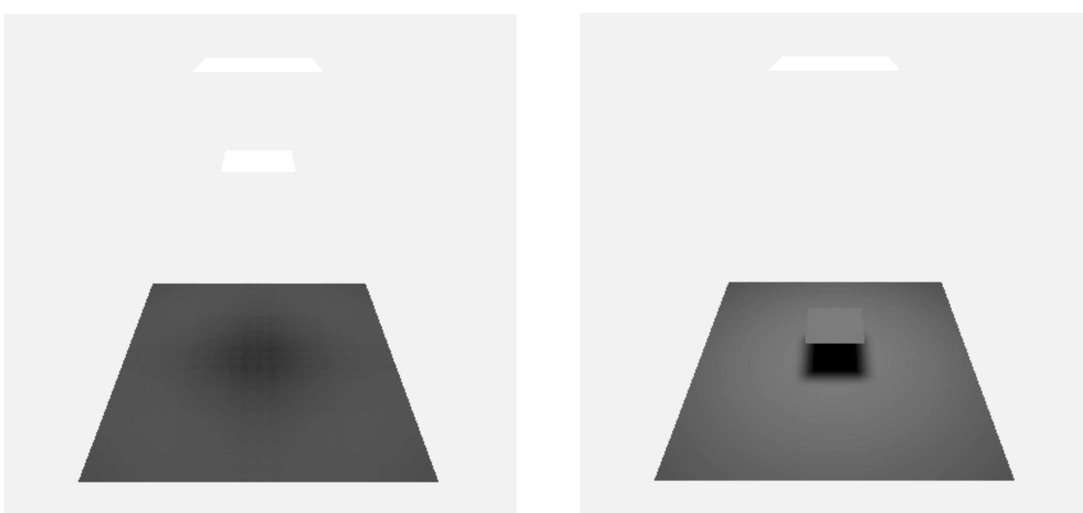


Figure 14: Blocker Experiment. Blocker-receiver separation of 1.4 and 0.4.

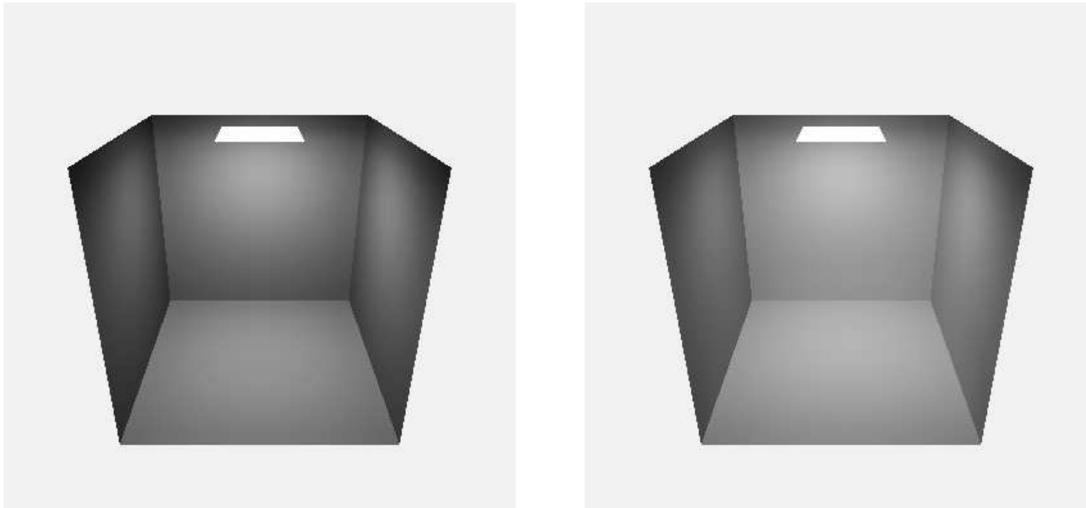


Figure 15: Box Experiment. Reflectance of 0.3 and 0.9.

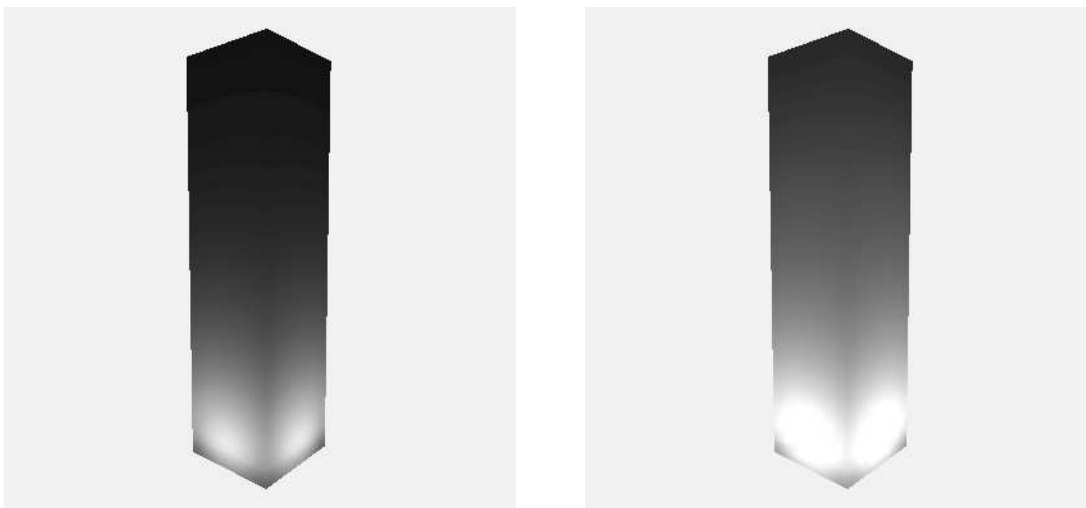


Figure 16: Tube Experiment. Reflectance of 0.3 and 0.9.

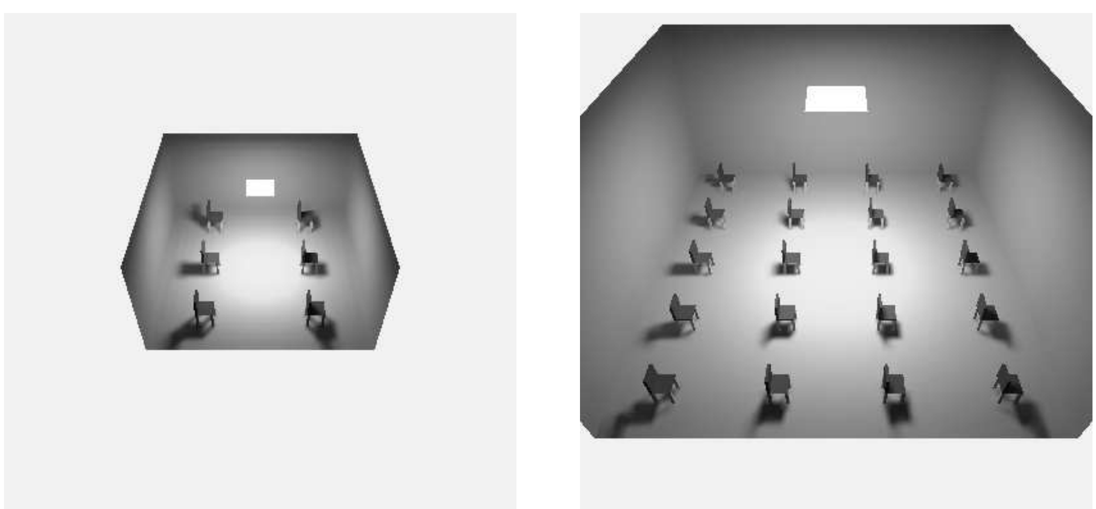


Figure 17: Complex Experiment. 6 and 20 chairs.

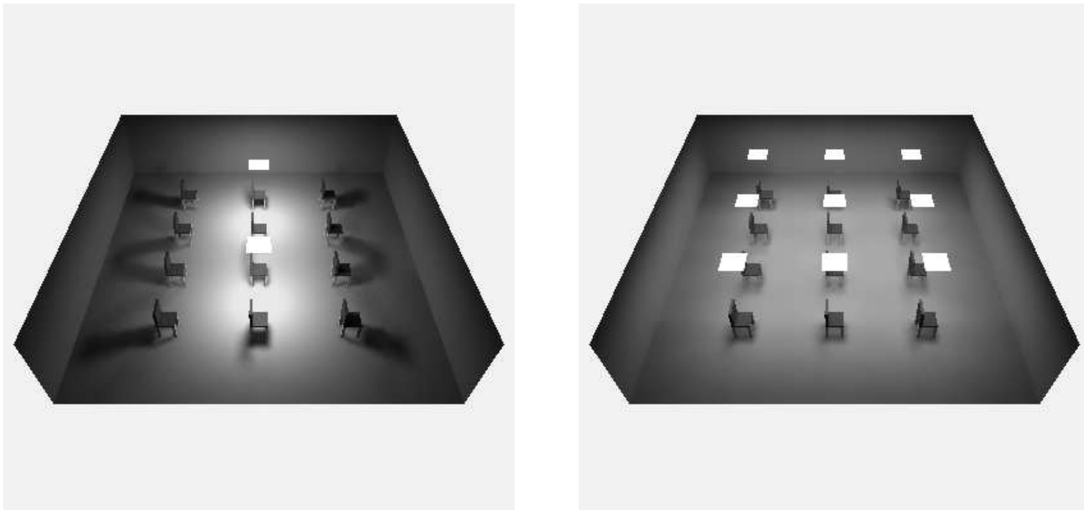


Figure 18: Lights Experiment. 2 and 9 lights.

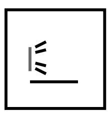
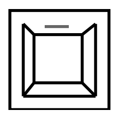
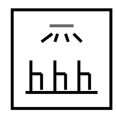
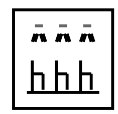
Method	 Parallel	 Perpendicular	 Blocker	 Box	 Tube	 Complex	 Lights
Matrix	edge_len 0.125	edge_len 0.125	edge_len 0.125	edge_len 0.25	edge_len 0.25	Not run	Not run
Progressive	edge_len 0.125 t_ratio 0.5	edge_len 0.125 t_ratio 0.5	edge_len 0.125 t_ratio 0.5	edge_len 0.25 t_ratio 0.001	edge_len 0.25 t_ratio 0.001	edge_len 0.25 t_ratio 0.001	edge_len 0.3 t_ratio 0.001
Progressive, Substruct.	edge_len 0.18 t_ratio 0.5 epsilon 0.07	edge_len 0.18 t_ratio 0.5 epsilon 0.07	edge_len 0.14 t_ratio 0.5 epsilon 0.05	edge_len 0.5 edge_len2 0.3 t_ratio 0.001 epsilon 0.11	edge_len 0.5 t_ratio 0.001 epsilon 0.2	edge_len 0.4 edge_len2 0.2 t_ratio 0.001 epsilon 0.05	edge_len 0.4 edge_len2 0.2 t_ratio 0.001 epsilon 0.2
Wavelet (All bases)	epsilon 0.02 t_ratio 0.0001	epsilon 0.02 t_ratio 0.0001	epsilon 0.05 t_ratio 0.0001	epsilon 0.007 t_ratio 0.0001	epsilon 0.01 t_ratio 0.0001	epsilon 0.01 t_ratio 0.001	epsilon 0.01 t_ratio 0.001

Table 3. Parameter Summary

The primary objective in choosing the above parameter settings was to minimise the bias towards one method or another. Generally, we tried to produce roughly equal running times for all methods (apart from matrix radiosity), subject to the constraint that all methods produce a satisfactory solution. The density of the progressive mesh was chosen to give a result in a reasonable amount of time, and the same mesh (and thus `edge_len`) was used for matrix radiosity. For progressive radiosity with substructuring, a coarser initial mesh was chosen (i.e., one with a larger `edge_len`), to take advantage of the decoupling of the source and receiving meshes. Where `edge_len2` is not listed, the initial receiving mesh was the same as the source mesh, and was then refined until all element gradients fell below `epsilon`. Where it is listed, the initial receiving mesh was finer that the shooting mesh, in order to ensure that shadow details or high illumination gradients were better captured. The `epsilon` parameter was chosen by trial and error to meet the goals above. The termination ratio was generally chosen to be 0.001, except where there was no interreflection. In such situations, the algorithm should be terminated as soon as all of the radiosity from the light source has been shot; setting `t_ratio` to 0.5 accomplishes this.

For wavelet radiosity, `epsilon` was chosen to equalise times and ensure a passable solution. The wavelet `t_ratio` was generally set to 0.001, except in the reflectance scenes, where it was set to 0.0001 to ensure that the solution properly converged once no more links were refined.

3.2. The Radiosity Testbed

In this section we describe the procedures we used to test the radiosity algorithms against the scenes in each experiment. All of our experiments were organised in a similar way. For each scene, the scene parameter was sampled at a number of settings, and for each of these settings a batch renderer was used to produce radiosity simulations and statistics. This renderer takes a scene description, as well as a number of parameters specifying which method and settings to use. Rather than an image, the renderer outputs a scene file containing a description of the mesh used, and the radiosity distribution over that mesh. Once computed and stored, this output mesh can later be retrieved and viewed in a scene viewer, compared with another mesh file, or rendered to an image file.

Both progressive radiosity and wavelet radiosity produce solutions that improve over time, so intermediate results are possible. Because of this, rather than only storing results for the final solution produced by the methods, we also generated statistics and meshes over the course of the simulation.

3.2.1. Timing and Checkpointing

For each radiosity simulation, the algorithm is checkpointed a number of times over the life of its run. Generally we ran the algorithm once to get an idea of the range of times over which it produced interesting results, and then set the next run so that the check points were evenly distributed over this range. At each of these checkpoint times, we halt the currently-running algorithm and the clock, write out the current state of the solution to a mesh file and pertinent statistics to a run file, and then restart the clock and continue the simulation. Thus a complete radiosity run against a certain scene produces a number of such mesh files, and a run file which summarises its performance over time. An example run file is shown in Appendix A.2, and an example mesh file in Appendix A.3. The mesh files are later compared to a reference mesh (a very accurate solution), and the error statistics generated added to the run file, from which graphs can be produced.

To avoid checkpoints occurring when the mesh is in an intermediate state, the renderer and checkpointing code run in cooperative threads. The renderer only yields to the checkpointing thread at times when it has finished a small, atomic section of work, and the state of its internal variables is consistent.

All timing measurements used the `times` system call to measure elapsed process CPU times. Thus, our time measures are independent of any overhead due to thrashing (heavy paging activity from disk). The experiments were run on unloaded machines to minimise interference from other processes.

3.2.2. Reference Solutions

In order to measure and compare the errors of the various radiosity methods, we would like to know the exact solution, but this is only possible for the simplest scenes, where there is no interreflection. To measure error for more general scenes, we take the pragmatic approach of generating a reference solution of the highest accuracy possible, and regarding that solution as exact [27]. For the two simplest test scenes, used in the parallel and perpendicular experiments, the lack of occlusion and interreflection allow us to compute an analytical solution. These use the formula for a patch-to-point form factor, as given in [14]. Numerically-robust code for this factor can be found in Appendix A.5, and the analytical solutions for two patches of equal size in parallel and perpendicular configurations can be seen in **Figure 19**.

For all but the parallel and perpendicular scenes, we generated a set of reference solutions. For the occlusion experiment and the interreflection test scenes, we used matrix radiosity, with analytical patch-to-point form factors, and a mesh at least four times as dense as any of the solution meshes that were generated. Matrix radiosity was used because it produces close to an exact solution for a given mesh, whereas most other algorithms give up some of this accuracy for speed. Instead of the normal visibility sampling strategy for the matrix method (`Vis1`), we used a high-accuracy inter-patch visibility estimator, similar to that used in the wavelet algorithms (`Vis16`).

While we could afford to run the reference solutions for much longer than we ran the experimental solutions[1], for the complexity and lights experiment, the matrix radiosity method proved too slow and, more importantly, too memory intensive for the available hardware. Instead, we generated our reference solutions with a progressive radiosity algorithm, using the afore-men-

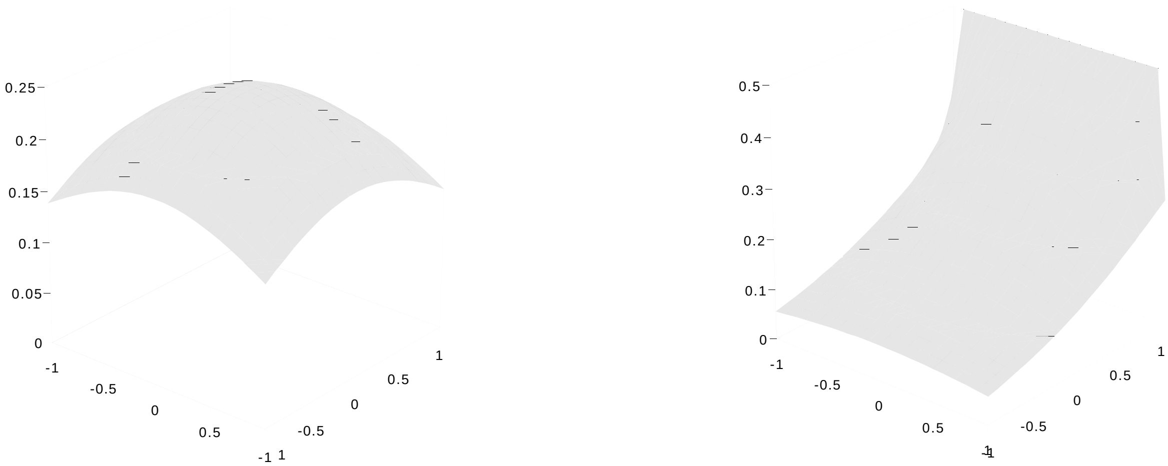


Figure 19: Analytical radiosity solutions

tioned high-accuracy inter-patch visibility test, and running it until the residual unshot radiosity was 0.0001 of the initial unshot radiosity. Thus the algorithm was run long enough to generate a solution of equivalent accuracy to the matrix algorithm, but the memory overhead of progressive radiosity was small enough to preventing thrashing on our machines. Substructuring was used, so the accuracy of the area-to-point form factors should be good, since the patches are larger than the elements.

In summary, to generate reference solutions, we have used an analytical solution for trivial cases, and for other cases chosen the solution and form factor methods we have most confidence in, and applied them to denser meshes with significantly more time and space available to them than for the experimental methods. The images shown in **Figure 12** through **Figure 18** are of reference solutions.

For all but the parallel and perpendicular scenes, our reference solutions are not exact, so we would like to verify that they are quite accurate. This is a very difficult problem, and the best we can do at present is to observe the errors between our other algorithms and our reference solutions. A necessary, but not sufficient, condition for correctness of our implementations is that they all converge to the same answer as mesh element size shrinks to zero and the number of iterations goes to infinity. Our graphs exhibited this trend; we achieved relative errors of 2% or less for most scenes with our best approximation. If our reference solutions were very inaccurate then it would be very unlikely that this would occur; it would require that all of our algorithms (wavelet, progress, and matrix) be biased or incorrect in the same way.

3.2.3. Measuring Error

We define error in terms of differences from our reference solutions. The most common way to compare two scenes in computer graphics is to use image-based error measures. Usually the pixel-wise root-mean-square (RMS) of the difference of the two images is used; we take the difference of the two images, square each pixel difference, sum them, and take the square root. The closer this is to zero, the better the fit between the images.

In the case of radiosity, a view-independent global illumination technique, this measure does not adequately capture the error in the scene; only the error as measured from a single viewpoint. As can be seen in **Figure 20**, the viewpoint chosen can greatly affect the error being measured. Two alternatives suggest themselves: we can take images of the scene from a number of ‘representative’ viewpoints, and average their error together, or we can bypass image comparisons altogether, and directly measure the error over the meshes produced by two different methods. For any but the simplest scenes, picking a small, repre-

1. For each scene and parameter combination, there is one reference scene, and on the order of 100 solution scenes.

sentative set of images is next to impossible, so we chose the latter path. All radiosity meshes originating from the same scene share the same set of base polygons, which leads us to a simple algorithm for scene comparison: we take the area-weighted average RMS error over each pair of base polygons as our measure of scene error, where

$$\text{rms\_error} = \sqrt{\frac{\sum_i \int (f_i - g_i)^2 dA_i}{\sum_i A_i}}. \tag{9}$$

Here f_i is the radiosity function over polygon i in the first mesh, g_i is the corresponding radiosity function in the second mesh, and A_i is the area of that polygon.

Obviously the RMS error between two meshes is scaled by the amount of radiosity emitted by light sources in the scene. To make our error measure invariant of scene brightness, we can normalise it by calculating the (area-weighted) average reflected radiosity over the reference solution, and dividing through by it. This normalisation constant is defined in Equation 10 below, where g_i is the emitted radiosity for the reference mesh.

$$\text{norm} = \frac{\sum_i \int (g_i - e_i) dA_i}{\sum_i A_i} \tag{10}$$

The reflected radiosity is used rather than the total radiosity because it is only the reflected radiosity that the radiosity algorithms calculate; the energy emitted directly from light sources is fixed. Note that our error, defined as rms_error / norm, is a unitless quantity, and will be invariant with global scene scale.

In theory, this error could be measured exactly; it involves constructing a super-mesh which contains both meshes being compared, and then using either quadrature or an analytical technique to calculate the necessary integrals over each element of this mesh. As error measurement was a post-process, however, there were few efficiency concerns, and following the 'exact' method would have required the implementation of a number of comparison algorithms between the various mesh types used. For ease of implementation we instead used Monte Carlo sampling over each pair of base polygons to estimate the error between them. This solution was composable; as long as every mesh implemented a 'sample' method for returning the radiosity at any point on the base polygon, the same routine could be used to compare any pair of meshes.

Thus, for each base polygon P_i in the scene a set of n_i uniformly-distributed sample points $\underset{\sim}{x}_{ij}$ were chosen, where n_i was proportional to A_i, the area of the polygon in world-space. The scene error was then calculated as follows, where f_i is the radiosity function over a polygon in the first mesh, g_i is the corresponding radiosity function in the second, reference, mesh:

$$\text{error} = \frac{\sqrt{\sum_{c \in \{r,g,b\}} \sum_i \frac{A_i}{n_i} \sum_j (f_{ic}(\underset{\sim}{x}_{ij}) - g_{ic}(\underset{\sim}{x}_{ij}))^2} \sqrt{\sum_i A_i}}{\sum_{c \in \{r,g,b\}} \sum_i \frac{A_i}{n_i} \sum_j (g_{ic}(\underset{\sim}{x}_{ij}) - e_{ic}(\underset{\sim}{x}_{ij}))} \tag{11}$$

The A_i / n_i term corrects for n_i being an integer; it will be close to constant for all polygons. Note that we are weighting the three colour channels equally.

3.2.4. Measuring Memory Use

Another of the statistics we wished to measure for each method was memory use. Rather than measure the actual memory used, which would have included the overhead used for storing the mesh and checkpointing, we calculated the memory used based on the minimal amount of information the renderer would need for a particular method. **Table 4** shows this calculated memory use for each of the methods; we assumed 8 bytes for a real number (the IEEE 'double' format), so that storing an RGB triple takes 24 bytes, and 4 bytes per pointer/index. For wavelet radiosity, M is the order of the wavelet basis used: 1 for Haar, 2 for F2/M2, and 3 for F3/M3.

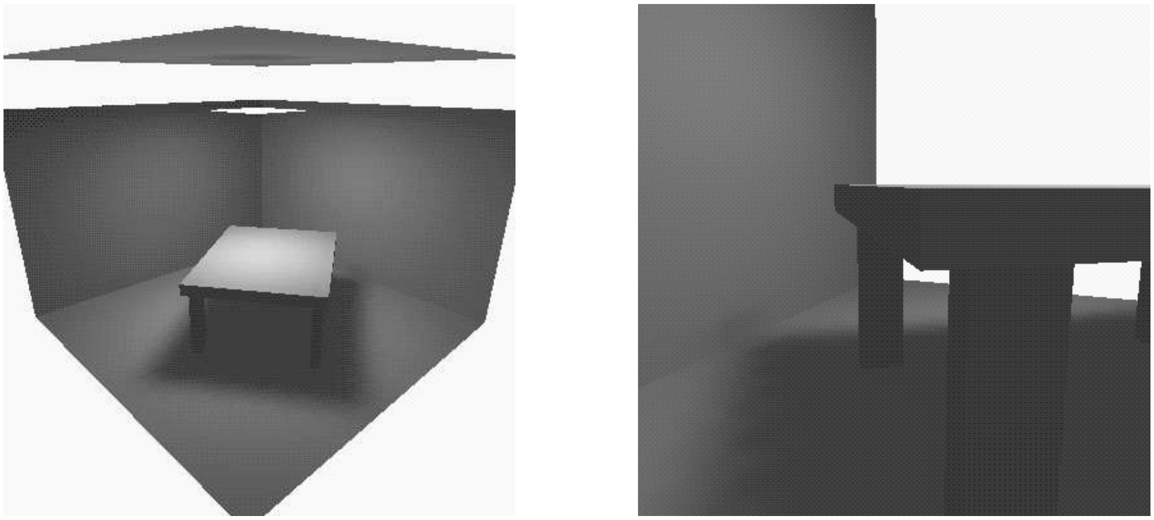


Figure 20: A radiosity scene from two different view points. The error is more significant in the image on the right.

Method	**Memory Use (bytes)**
Matrix	$24 \times$ patches + $28 \times$ non_zero_form_factors (The radiosity vector B, and sparse matrix A.)
Progressive	$96 \times$ patches (The radiosity vectors S, B, and one row of form factors.)
Progressive with Substructuring	$24 \times$ patches + $52 \times$ elements (S, B, the form factors, and the element to patch indices.)
Wavelet	$(16 + 24 \times M^4) \times$ links + $(20 + 24 \times M^2) \times$ elements (There is an overhead of 2 pointers and a real number per link, and five pointers for an element. An element contains M^2 coefficients, and a link M^4.)

Table 4. Radiosity Method Memory Use

3.2.5. Result Equalisation

For each experiment we run, we can produce a plot of the solution error over time for every combination of scene and method. We refer to these as *time/error* curves (see **Figure 22**). Each curve is a run, and each data-point is a snapshot of that run at some point in time. The right-most point shows the error at convergence (when the algorithm terminates), and the previous points show the error at intermediate times. Unfortunately, this mass of data can be confusing; it requires we either pack many curves onto a reasonable number of graphs, or vice-versa. For performance analysis we need a method for summarising the differences between the various methods for different values of the scene parameter.

One way to present the results is to plot the time each method took to terminate, and also plot the error each method produced at the end of its run, as in **Figure 21**. In these graphs, each curve corresponds to a given method run at various parameter settings, and each point represents an individual run. From these graphs we can see at a glance which method produces the smallest error, and which takes the longest to converge. Unfortunately this makes it hard to compare a method which produces a reasonable error quickly with one that produces a slightly better error, but takes much longer to do so; in many situations we might prefer the first method.

To combat this we introduce the idea of a time-equalised error plot. For a plot of a given set of methods, we find the minimum convergence time of any method, and then find the error in the methods at that time by interpolation. This is illustrated in **Figure 22**: the substructuring method terminates first in the graph on the left at about 140 seconds, so all the other methods are 'rolled back' to this point, and the resulting interpolated error values used to produce the highlighted data points in the graph on the right.

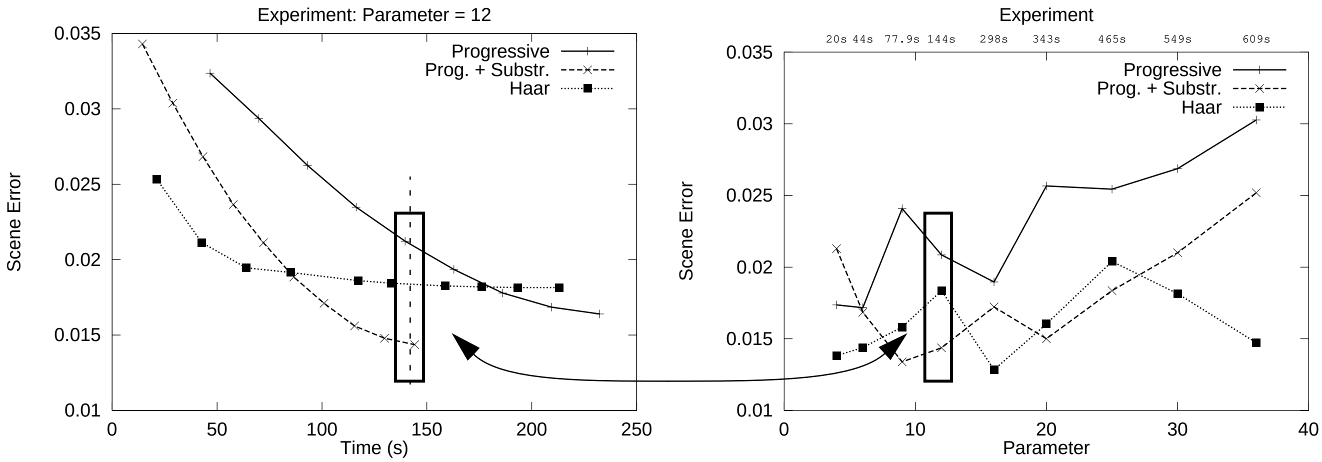


Figure 22: Producing an Equalised Plot. The numbers along the top of the equalised plot denote the time used to produce each set of data points.

This approach to equalisation must be used with caution, as it is only applicable to methods which produce a progressively improving solution, where cutting off the method after a certain time is valid. This is certainly the case with progressive radiosity, and is typically the case with wavelet radiosity. We must bear in mind that we could also change the character of the time/error curve by changing epsilon for either of these methods. Plots including matrix radiosity cannot be equalised, as it does not produce an accurate solution until well into its run. We have also chosen not to use equalisation for any plots involving progressive radiosity in the first three experiments, as in these cases rolling back the progressive methods leaves radiosity from part of the light source unshot, which would unfairly penalise the progressive methods.

We must also ensure that the time/error curves of each method should largely overlap in time, as we don't wish to compare the error of one method early in its run with that of another towards the end of its run[1]. Another disadvantage is that because a different time value is used for each experiment parameter, the shape of the curves is no longer as significant as with a standard

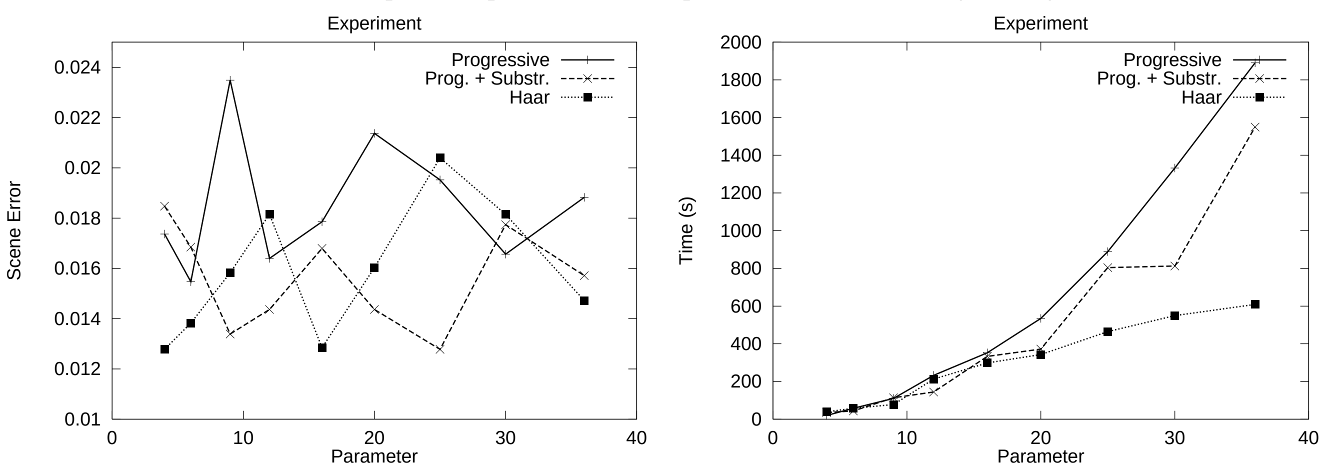


Figure 21: Time and Error plots at Convergence.

1. It is easier to control the convergence times of the various methods than their error after convergence, which is why we use error plots equalised by time, rather than time plots equalised by error.

error plot; it is the order of the methods at each different experimental parameter that matters. In order to clarify the equalised plots we will label each set of data points on them with their corresponding time.

At first glance our equalisation method might seem artificial, since we are reporting an error value for an interpolated solution that was never computed. But in most cases, it would be possible to tune the meshing, oracle, or termination criteria so that a solution with approximately that error could be computed in that time. We believe that equalisation, while it has its limitations, typically achieves a fairer comparison of the methods than a graph of error at convergence.

Since we have found it easier pragmatically to produce time-equalised error plot rather than error-equalised time plots, most of the plots in the next section are of error, and not time.

This should not be misinterpreted as an indication that we are more concerned about accuracy than speed. Fortunately, if a time-equalised plot shows that method A is more accurate than method B, and if both methods have monotonically decreasing time-error curves, then this implies that method A is faster than method B at achieving a given a given error. More precisely, if method A has error e_A at time t, method B has error e_B at time t, $m_A < m_B$, and both methods have monotonically decreasing time-error curves, then method A is faster than method B at achieving any error in the range $[e_A, e_B]$, and usually a much bigger error interval.

3.2.6. Summary

Our data set is five-dimensional: it is parameterised by experiment, scene parameter, radiosity method, time, and statistic. Statistics include scene error, memory use, and current number of patches and/or elements. Naturally, this data consumes an enormous amount of storage space: the 500 radiosity runs produced 590MB of run data and (compressed) mesh files, not including final rendered images. Over 40,000 lines of code were written for this project, broken up as follows:

Code	Lines
Vector/Matrix Library (C++)	12,000
Graphics Library (C++)	8,000
Radiosity Code (C++)	18,000
Experiment and Plotting Scripts (awk/gnuplot)	2,000
Total	40,000

The total time for the project, including the design, coding, experimentation, and analysis phases, was two years. We present the results of that analysis in the next section.

4. Results

Our final results comprised a large database of statistics and stored radiosity representations. We used a customised plotting package (Appendix A.4) to pull out graphs of the results along any of the relevant axes, for example: scene parameter, time, error, number of patches. In this section we present the most interesting of the graphs and images produced, and discuss their implications.

We start by looking at the most significant results, the performance of the various methods in the different experiments, in Section 4.1. As well as being of most interest, this gives a good overview of the type of results we got. We then take a more in-depth look at particular results from the experiments in Section 4.2. Finally, we compare some of the theory behind the different radiosity methods to our results in Section 4.3. In order to keep descriptions manageable, we will refer to progressive radiosity without substructuring as 'progressive', and progressive radiosity with substructuring as 'substructuring'. We refer to hierarchical radiosity and wavelet radiosity using the Haar basis interchangeably. Finally, plot styles are kept constant from graph to graph: the same method will always have the same data symbol and line style.

4.1. Performance Analysis

In this section we look at the performance of the radiosity algorithms surveyed. Where we can, we use the equalised plots described in Section 3.2.5. For those graphs showing progressive radiosity methods in the first three experiments (parallel, perpendicular and blocker) we do not do this because it would overly penalise the progressive methods. Likewise, graphs involving matrix radiosity are not equalizable. In these cases we instead show plots of error after convergence and time to convergence. For most of the experiments we will compare the relative performance of the progressive and hierarchical methods, and then performances of the various wavelet methods (of which hierarchical radiosity is one).

An important factor to bear in mind is that typically only the scene parameter was varied over the course of any given experiment; all of the other parameters were held constant.

4.1.1. Parallel and Perpendicular

Parallel: Progressive vs. Hierarchical

The parallel scene is intended to test how well each radiosity method can match the simplest type of radiosity function, one without singularities, and with no occlusion. **Figure 23** and **Figure 24** show the performance plots for the parallel experiment for the progressive and hierarchical radiosity methods.

The time taken by the progressive and substructuring methods is roughly equivalent; at smaller separations, the gradient of the radiosity function increases, and substructuring starts to subdivide heavily, so it takes longer than progressive. At larger distances little subdivision occurs, and because it has a coarser initial mesh, substructuring outperforms progressive. Both substructuring and wavelet radiosity achieve roughly the same level of error regardless of separation, being adaptive methods, whereas progressive radiosity is limited by the size of its mesh: as the separation decreases the radiosity function develops steeper gradients, and its error increases rapidly.

For greater distances, hierarchical does well, but as the separation decreases, it must create many links, and is clearly outperformed by the progressive methods. (The number of links increases from 1000 to over 150,000 over the course of the experiment.) One problem with the hierarchical approach is that it subdivides the source and receiver evenly; an asymmetric approach such as that which originally motivated progressive radiosity with substructuring would be more efficient; the source does not need to be subdivided as finely as the receiver. Also, at some point it becomes more efficient in time as well as memory to use the patch-to-point formula for the form-factor rather than approximate it by subdividing the source and adding up the point-to-point form-factors for the subpatches.

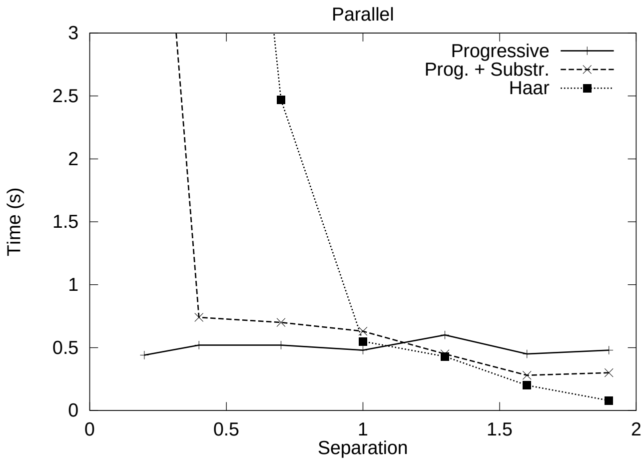


Figure 23: Parallel experiment, time to convergence for progressive and hierarchical radiosity.

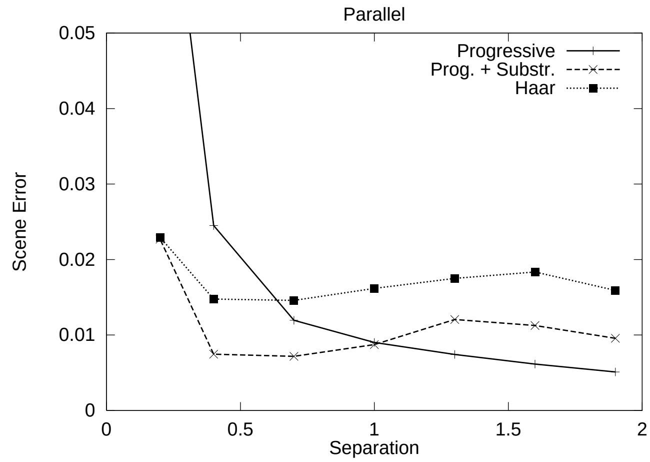


Figure 24: Parallel experiment, error after convergence for progressive and hierarchical radiosity.

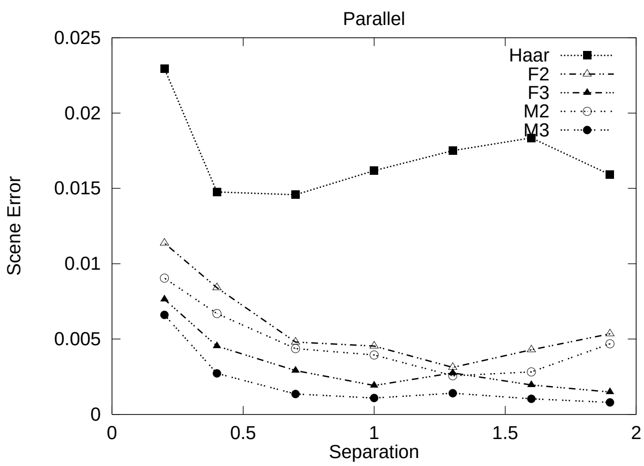


Figure 25: Parallel experiment, error after convergence for wavelet radiosity.

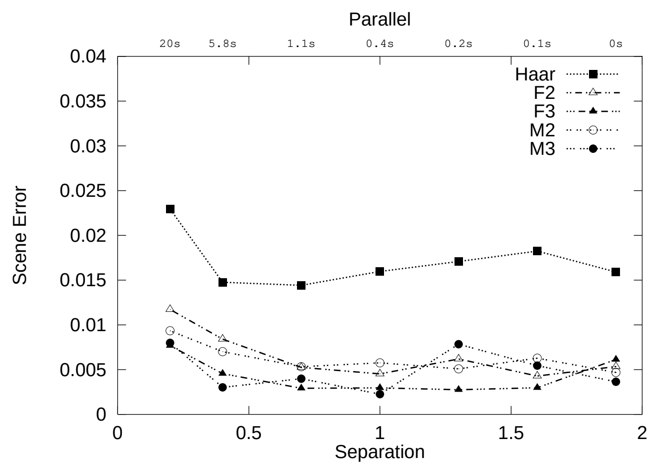


Figure 26: Parallel experiment, equalised error for wavelet radiosity.

So, at greater separations, hierarchical radiosity performs the best, but at low separations, it creates too many links to be competitive with substructuring.

Wavelet

Turning to the wavelet bases in the parallel experiment, **Figure 25** shows the error of these methods after they have converged, and goes as theory would have us expect[1]. The most accurate methods are the higher-order methods, and multiwavelets are a little more accurate than the corresponding flatlets. Most of the methods have their highest error when the separation is small, and this drops off as the separation increases. **Figure 26** shows the errors after they have been equalised by time, as described in Section 3.2.5. In this case the F3 basis is typically the best, as the error-advantage of the M3 basis is cancelled out by the longer time it takes to converge. Haar is considerably less accurate than the higher-order methods in this experiment, and its convergence time is typically in the middle of that of the other bases. For example, for a separation of 1.0, F2 is the fastest to converge in 0.4s, Haar takes 0.55s, and M3 is the slowest at 0.63s. All the wavelet methods are slow for low separation. Note

1. Note that in the wavelet graphs, the Haar symbol is a box, the flatlet symbol a triangle, and the multiwavelet symbol a circle. Also, M2/F2 data lines consist of repeating patterns containing two dots, and F3/M3 patterns contain three.

from the top of the graph that the first column of data points, at a separation of 0.2, is equalised at 20s, whereas those for separations greater than one are equalised at times less than a second.

We now look at the perpendicular experiment, to see whether the advantages of progressive radiosity at low separations and the F3 wavelet method at high separations still hold in the presence of the singularity in the radiosity kernel.

Perpendicular: Progressive vs. Hierarchical

The perpendicular scene is in the same vein as the parallel scene with the addition of the approach to the singularity in the radiosity kernel, which occurs when the separation is zero and the edges of the source and receiver touch. Thus it serves as a somewhat sterner test of the radiosity methods' ability to fit the radiosity function.

In **Figure 27** we see that hierarchical radiosity produces a reasonably constant error, whereas the progressive methods have large error for a small separation, which then decreases as separation increases. This is due to the fixed resolution of its mesh for progressive radiosity, as in the parallel experiment. For substructuring, at smaller separations more of the elements have the maximum radiosity gradient allowed for the given setting of epsilon, so the error increases.

Figure 28 shows the corresponding convergence times for these methods: here it is progressive radiosity that has a stable completion time, whereas hierarchical radiosity spends much more time for small separations than it does for larger ones, due to the larger number of links it must create at small separations.

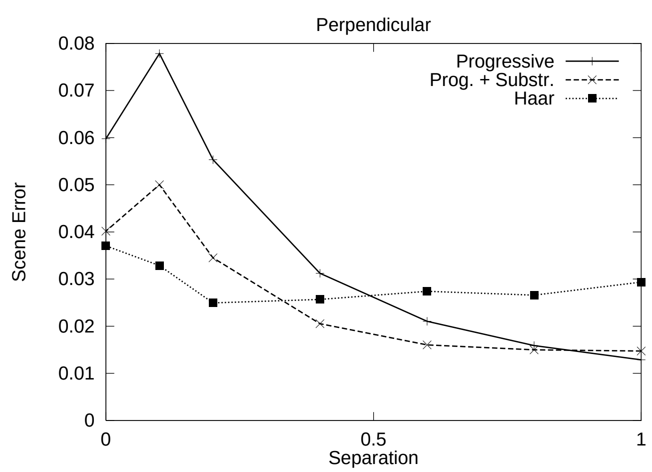


Figure 27: Perpendicular experiment, error after convergence for progressive and hierarchical radiosity.

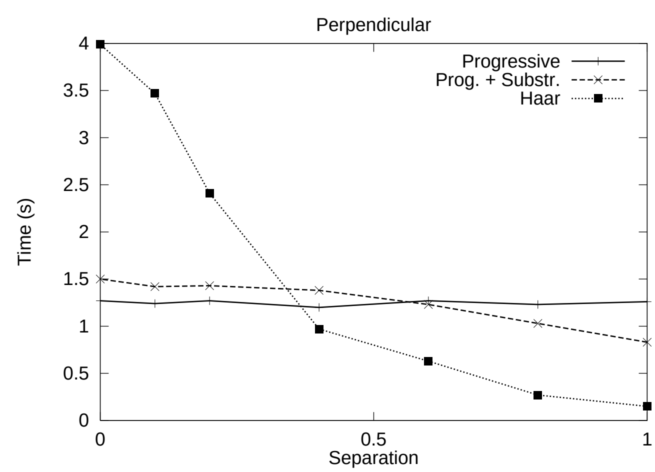


Figure 28: Perpendicular experiment, time to convergence for progressive and hierarchical radiosity.

These graphs illustrate the different time/error trade-offs the algorithms make; if we are more interested in constant error than time for a certain set of parameters, the hierarchical method would be the best choice for this experiment, whereas for a (relatively) constant time solution the progressive methods are better.

Ideally, an algorithm should terminate at a constant scene error, but be capable of being halted at an earlier time, with correspondingly larger error. Hierarchical radiosity seems to meet this goal best for the perpendicular experiment.

Wavelet

We now look at how the different wavelet bases perform in the perpendicular experiment. The equalised plot in **Figure 29** shows similar results to the corresponding parallel experiment plot, although all the higher-order bases are closely grouped. Once again there is a large difference between zero-order wavelets (Haar) and the higher-order versions, although now Haar consistently has the smallest convergence time; at a separation of 0.6 it takes 0.63s, as opposed to F2's 0.7s, and M3's 2.11s.

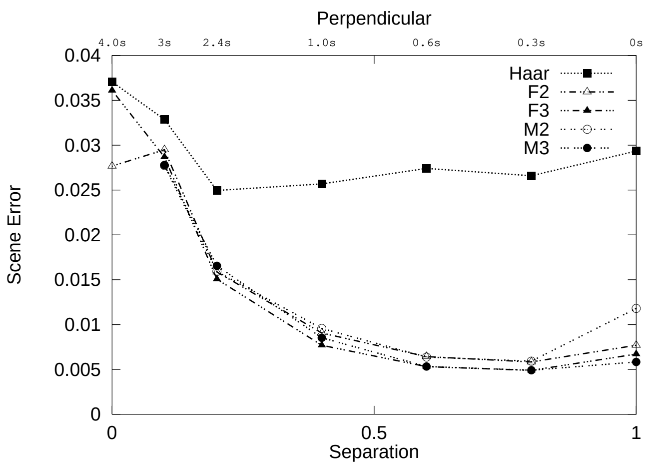


Figure 29: Perpendicular experiment, equalised error for wavelet radiosity.

So, again the higher-order methods outperform the Haar basis in a simple two-patch test. The introduction of occlusion to the equation should affect these results; we examine the blocker experiment next to see how they change.

4.1.2. Blocker

Figure 30 shows a pair of typical solutions and their meshes for the occlusion experiment. From the mesh pictures we see that substructuring subdivides the receiving patch more heavily than hierarchical radiosity, whereas hierarchical radiosity subdivides the light source more. For the separation shown (0.7), substructuring provides the more accurate solution, with an error of 0.012 compared to hierarchical radiosity's 0.031.

Progressive vs. Hierarchical

We start with the time to convergence of the progressive and hierarchical methods. The two important features of the time plot in **Figure 31** are that substructuring and hierarchical radiosity are in a dead-heat for low separations, whereas for large receiver-blocker separations (and thus smaller source-blocker separations) hierarchical radiosity's convergence time balloons. We might expect substructuring to be more accurate than hierarchical radiosity when shadows are sharp. From the corresponding error plot in **Figure 32** we see that whereas substructuring achieves the lowest error when the blocker is closer to the receiver, and hierarchical radiosity the worst, this order is reversed as the blocker approaches the source. Significantly, the cross-over point occurs almost exactly when the umbra disappears, at a separation of $1\frac{1}{3}$.

Hierarchical radiosity takes much longer to produce a result for low source-blocker separations because it is trying to solve a harder problem. Whereas substructuring will only subdivide the receiver, hierarchical radiosity can also subdivide the source, and in this case it must subdivide heavily; the source is only a little larger than the blocker, so source elements must be quite small for them to be close to completely visible from the elements of the receiver. This subdivision leads to many extra rays being cast, as we can see in **Figure 33**.

In summary, the substructuring approach achieves the best results when the shadows are harder, and hierarchical when the shadows are softer, although it takes much longer to do so, and the differences are less marked. As in the parallel experiment, this would indicate that for hierarchical subdivision it might be advantageous to bias subdivision in favour of the receiver.

Wavelet

To look at how occlusion affects the different wavelet bases, we examine the equalised error plot in **Figure 34**. In contrast to the earlier experiments, the results here are not so clearly in favour of the higher order methods. Surprisingly the F3 basis now does the worst, presumably due to a combination of the shadow-resolution problems we discuss in Section 4.1.4, and the creation of slightly more links than the M3 basis, which means a greater number of rays must be cast. (At a separation of 1.8, F3

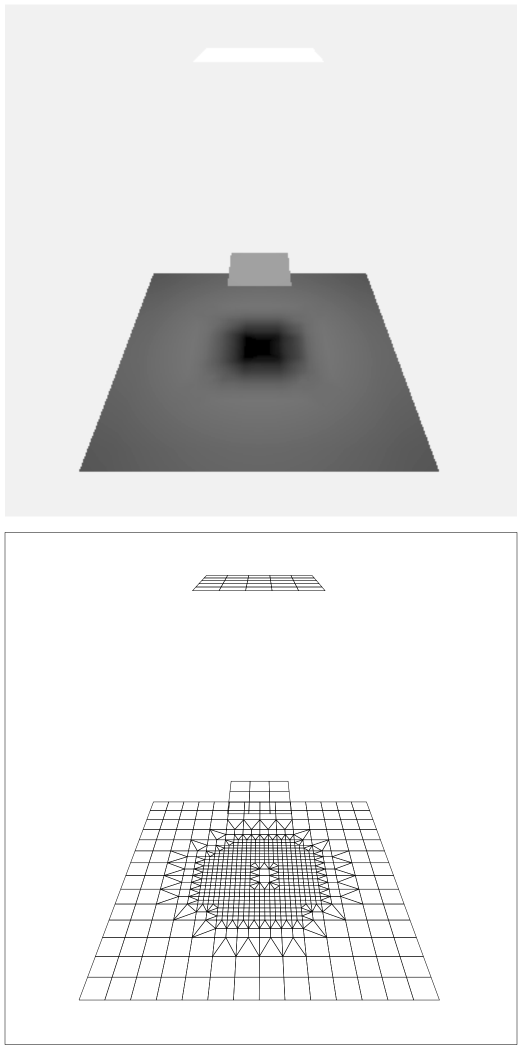


Figure 30: Blocker experiment, image and corresponding mesh for substructuring radiosity (left) and hierarchical radiosity (right) at separation = 0.7.

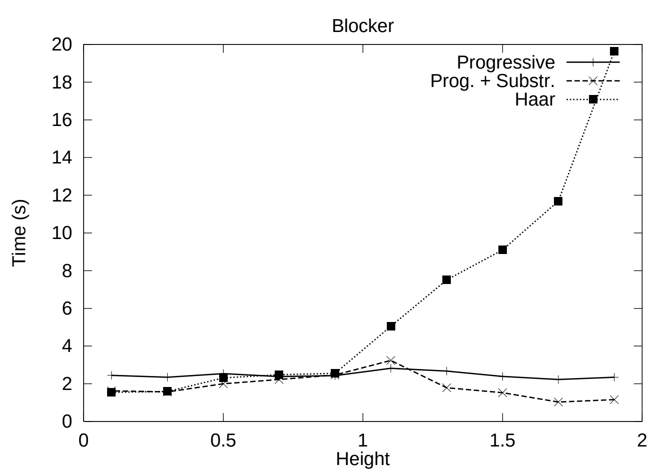


Figure 31: Blocker experiment, time to convergence for progressive and hierarchical radiosity.

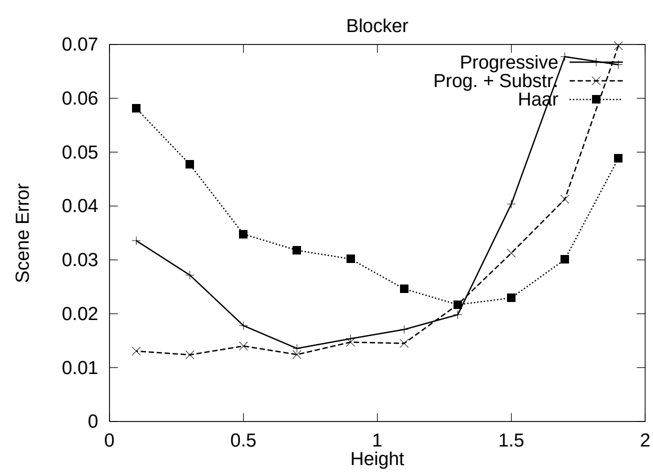


Figure 32: Blocker experiment, error at convergence for progressive and hierarchical radiosity.

has 4,000 links and M3 has 3,500; the difference is less marked for smaller receiver-blocker separations.) The Haar basis does better than it did relative to the other wavelet methods in the parallel experiment, largely because it creates many more links than them (18,000 at a separation of 1.8) and thus casts more rays, resulting in a better visibility resolution.

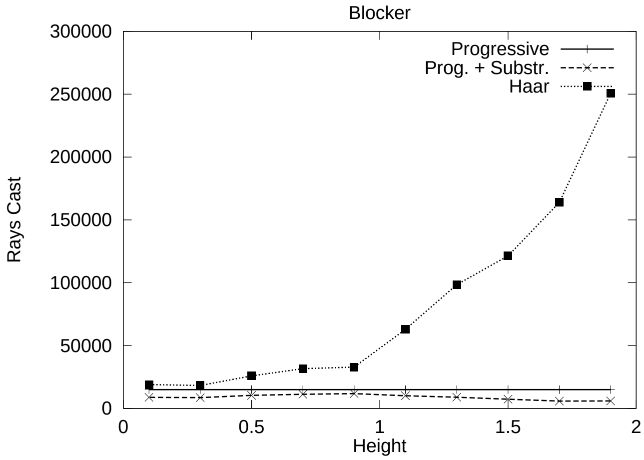


Figure 33: Blocker experiment, rays cast at convergence for progressive and hierachical radiosity.

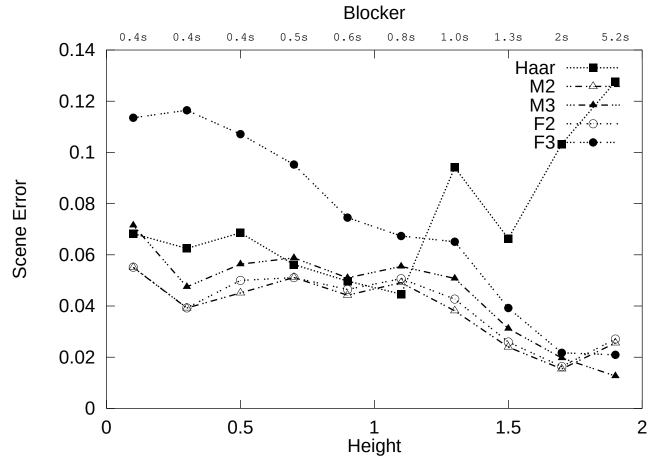


Figure 34: Blocker experiment, equalised error for wavelet radiosity.

As we can see from **Figure 35**, the visual results are highly dependent on the choice of quadrature method. (See Section 2.4.2.)

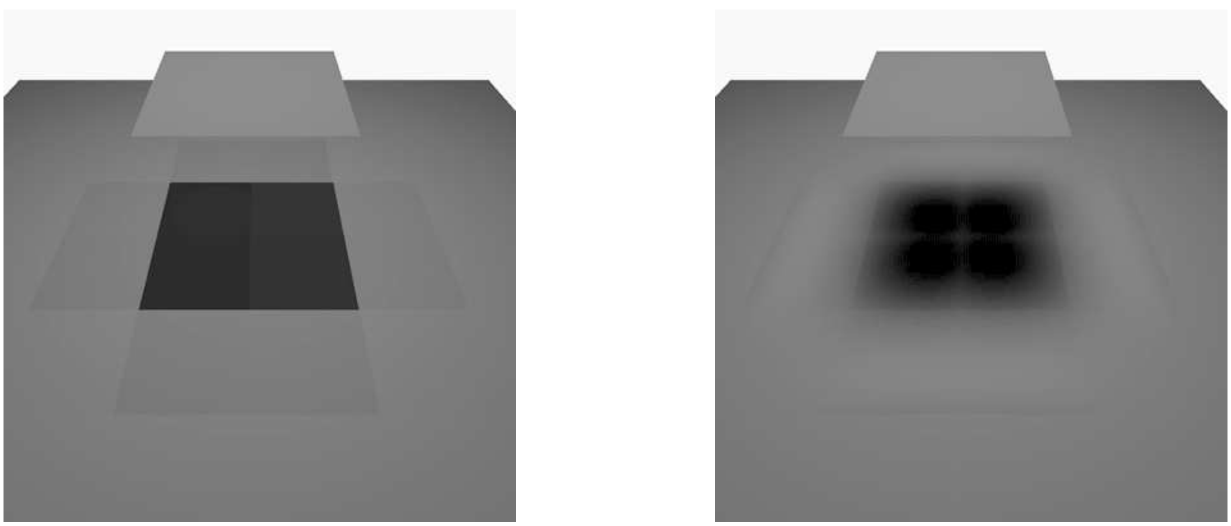


Figure 35: Different wavelet visibility methods with the M3 basis. On the left, fractional visibility is used, and on the right, visibility-in-quadrature. The solution on the left is blocky, while the solution on the left, although much smoother, suffers from ringing.

With the fractional visibility method described by Gortler et al., and used in our experiments, shadows appear overly blocky. With the visibility-in-quadrature method of Gershbein et al., the shadows are much smoother, but prone to the ringing artifacts mentioned by Zatz [64]; there are also still some small discontinuities between elements. The latter method is visually superior in this example.

4.1.3. Box and Tube

We now move to the effect of reflectance on the radiosity methods, which we investigate with the tube and box experiments. We start with the box experiment and results for hierarchical and progressive radiosity, and also include the results for (the normally impractical) matrix radiosity as a benchmark.

Box: Progressive vs. Hierarchical

Figure 36 and **Figure 37** show the error and time at convergence for the matrix, progressive, and hierarchical radiosity methods. The matrix method produces a relatively constant error in constant time; the other methods take increasing amounts of time to converge as reflectance increases, and hierarchical radiosity's error also increases[1]. (Note that the matrix curve is coincident with the progressive curve.) The most interesting aspect of this graph is that the error curves for the progressive and matrix methods are almost identical. This suggests that the error caused by the discretization of the mesh greatly outweighs the error due to the unshot radiosity in the progressive method, even in high-reflectance scenes. (The matrix and progressive methods shared identical mesh parameters, and thus meshes, and the progressive method used `t_ratio` = 0.001.) Surprisingly, above a scene reflectance of 0.8, matrix radiosity is actually the fastest method.

An equalised plot of the hierarchical and progressive methods produces almost identical results to **Figure 36**. As we can see from that graph, the substructuring approach does well for all reflectances, although its time to completion does increase steadily with the level of reflectance. Hierarchical radiosity does particularly poorly at high reflectance levels; although it creates a larger number of links here, and thus takes longer to converge, its error level still deteriorates as the reflectance increases.

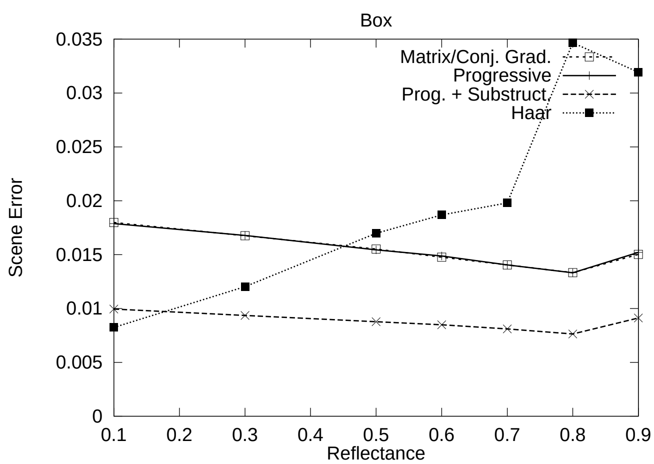


Figure 36: Box experiment, error after convergence for matrix, progressive, and hierarchical radiosity.

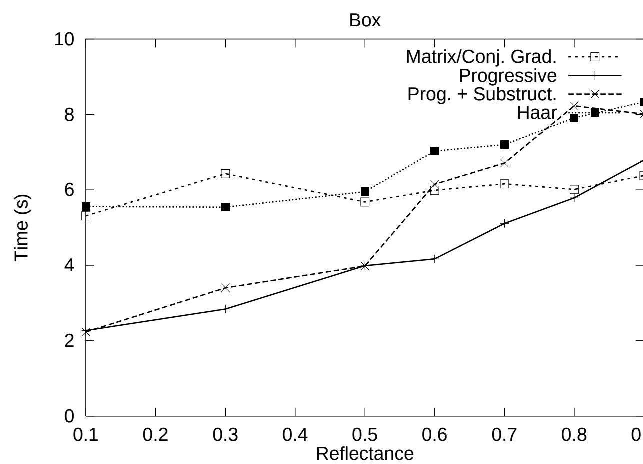


Figure 37: Box experiment, time to convergence for matrix, progressive, and hierarchical radiosity.

Wavelet

Fortunately, other wavelet bases do a better job in this experiment than Haar. An equalised plot for the different wavelet bases is shown in **Figure 38**, where F3 appears the most effective method, especially as reflectance increases. (Indeed, it is the only wavelet method whose error does not start to rise dramatically with increased reflectance.) In fact, if we compare the F3 error line to that of substructuring in the previous graph, we see both hover around an error level of 0.01.

For the box scene, then, substructuring outperforms hierarchical radiosity, especially at high reflectance levels, but the F3 wavelet basis does as well as substructuring. We now look at how the tube scene changes these results.

---

1. The error for progressive and matrix radiosity appears to be decreasing, but recall that scene error is measured relative to the reflected radiosity of the scene; the absolute error in the box scene is increasing in this case.

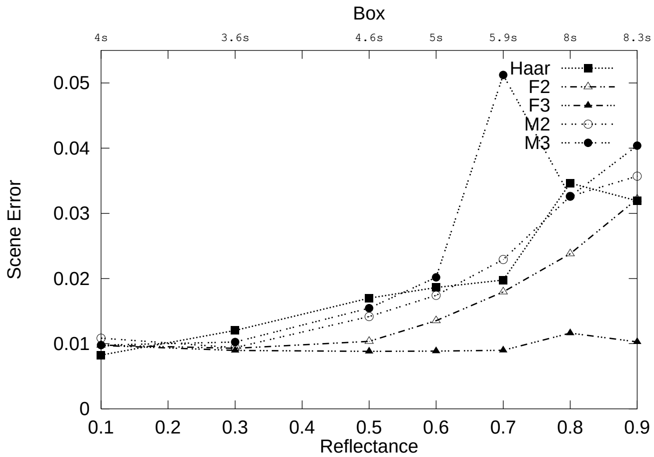


Figure 38: Box experiment, equalised error for wavelet radiosity.

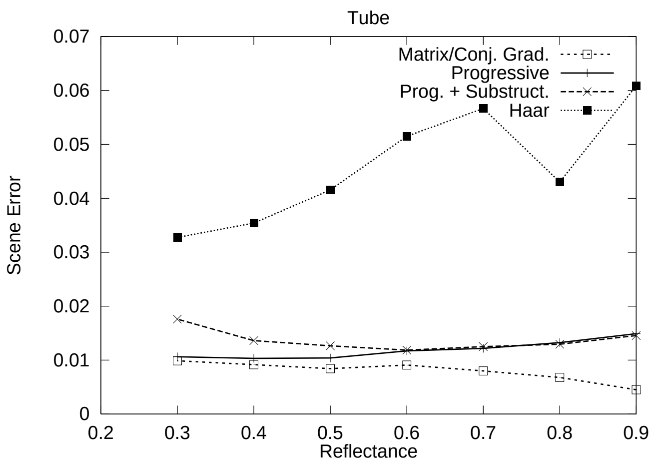


Figure 39: Tube experiment, error after convergence for matrix, progressive, and hierarchical radiosity.

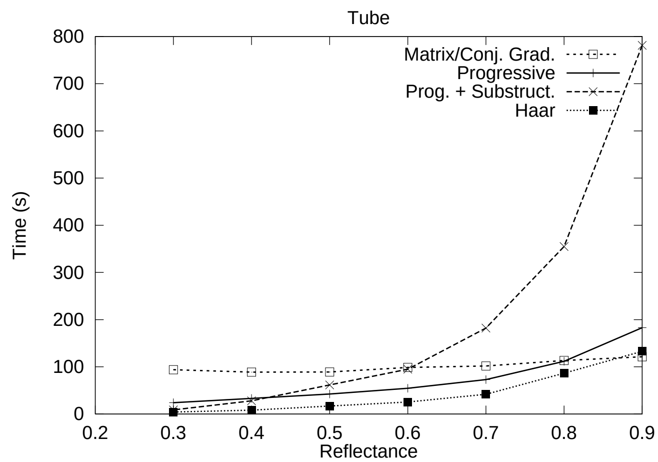


Figure 40: Tube experiment, time to convergence for matrix, progressive, and hierarchical radiosity.

Tube: Progressive vs. Hierarchical

Due to a number of factors, the tube experiment proved the most demanding experiment for all the methods except matrix radiosity. For the progressive methods this was due to the long chains of shooting steps necessary to illuminate the entire tube, and for the wavelet methods the high gradient of the radiosity function along the walls, together with the singularity where the patches meet, caused problems.

Figure 39 shows the error results for the tube: the hierarchical method does poorly here, and as reflectance increases, the progressive methods drop below the accuracy of matrix radiosity, which serves as a baseline for our comparison. At high reflectances, the large eigenvalues in the matrix equation governing the simulation mean that shooting converges more slowly, and the residual error for a fixed epsilon grows.

The error plot in **Figure 40** is notable for one result; the time taken for substructuring increases super-linearly with reflectance. This is largely due to the phenomenon of *over-refinement*, which we explore further in Section 4.2.4. Briefly, during early shooting steps, the gradient is high near the bottom of the tube, causing that area to be heavily refined, even though later shooting steps will “fill in” the tube’s illumination, smoothing the gradient and eliminating the need for such a fine mesh. Thus, while the substructuring takes up time creating a finer mesh, this mesh does not produce an appreciably more accurate result than progressive radiosity, as we saw in **Figure 39**. To get a better picture of the time/error trade-off, the equalised plot of

Figure 41 shows dramatically the problems substructuring and hierarchical radiosity have with this scene, as both are well out-performed by progressive radiosity. **Figure 43** shows images of the solution for progressive and substructuring radiosity run on the tube scene at a reflectance level of 0.7. The extent of the over-meshing by substructuring radiosity is clearly shown by the mesh pictures below these images.

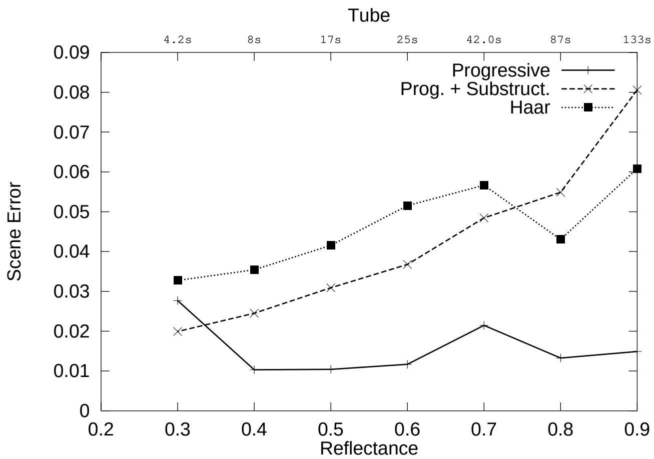


Figure 41: Tube experiment, equalised error for progressive and hierarchical radiosity.

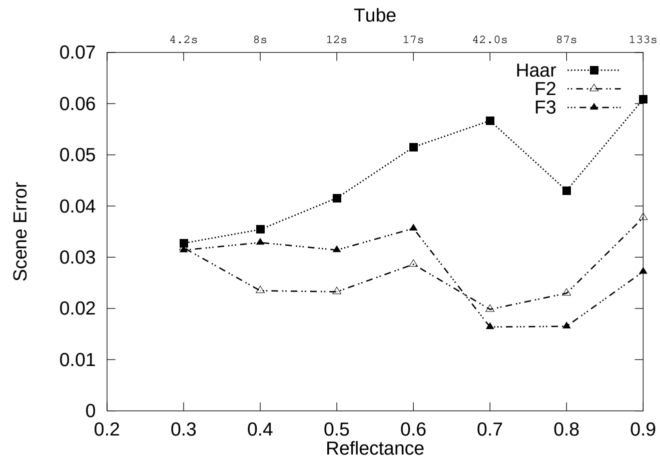


Figure 42: Tube experiment, equalised error for wavelet radiosity.

Figure 40 also shows the dramatic increase in time needed by the progressive and hierarchical techniques as reflectance increases. While substructuring becomes 89 times slower for reflectance 0.9 relative to 0.3, the progressive and hierarchical radiosity become 8 and 32 times slower over the same interval. Gortler et al. noted that progressive radiosity is efficient when power is concentrated on a small fraction of the patches in a scene, and less well when it is broadly distributed (as in a scene with indirect illumination and high reflectance) [21]. Our results support this conclusion. Clearly, high reflectance makes all of the methods run much more slowly. Meanwhile, matrix radiosity is relatively oblivious to variations in reflectance. This issue is pursued further in Section 4.3.1.

Wavelet

In **Figure 42** we see similar results for the wavelet bases as for the box experiment; as reflectance increases, the F3 basis becomes the most effective method. For lower reflectances, the quicker convergence times of the F2 basis give it a slight edge. The M2 and M3 bases are not plotted because for higher reflectances (above 0.6) they used more than the available amount of memory (128Mb) and were terminated.

4.1.4. Complex

Now we come to the complex experiment, which is perhaps our most interesting experiment. **Figure 44** shows a pair of typical solutions and their meshes for the complex scene with 16 chairs. In this case hierarchical radiosity has meshed the walls more finely than the substructuring method, and has also subdivided the floor directly under the light source. Substructuring radiosity has subdivided largely around shadow boundaries, although it is noticeable that in one or two spots the shadows seem blocky. In this particular case the hierarchical solution is more accurate (with an error of 0.13 vs. 0.16 for substructuring), but we will see later that this is not always the case.

As usual, we start by comparing the progressive and hierarchical radiosity methods.

Progressive vs. Hierarchical

Figure 45 shows the equalised progressive/hierarchical plot for the complex experiment. Except for the lowest complexity scenes, the substructuring and hierarchical methods outperform the progressive method, as we might expect, due to the pres-

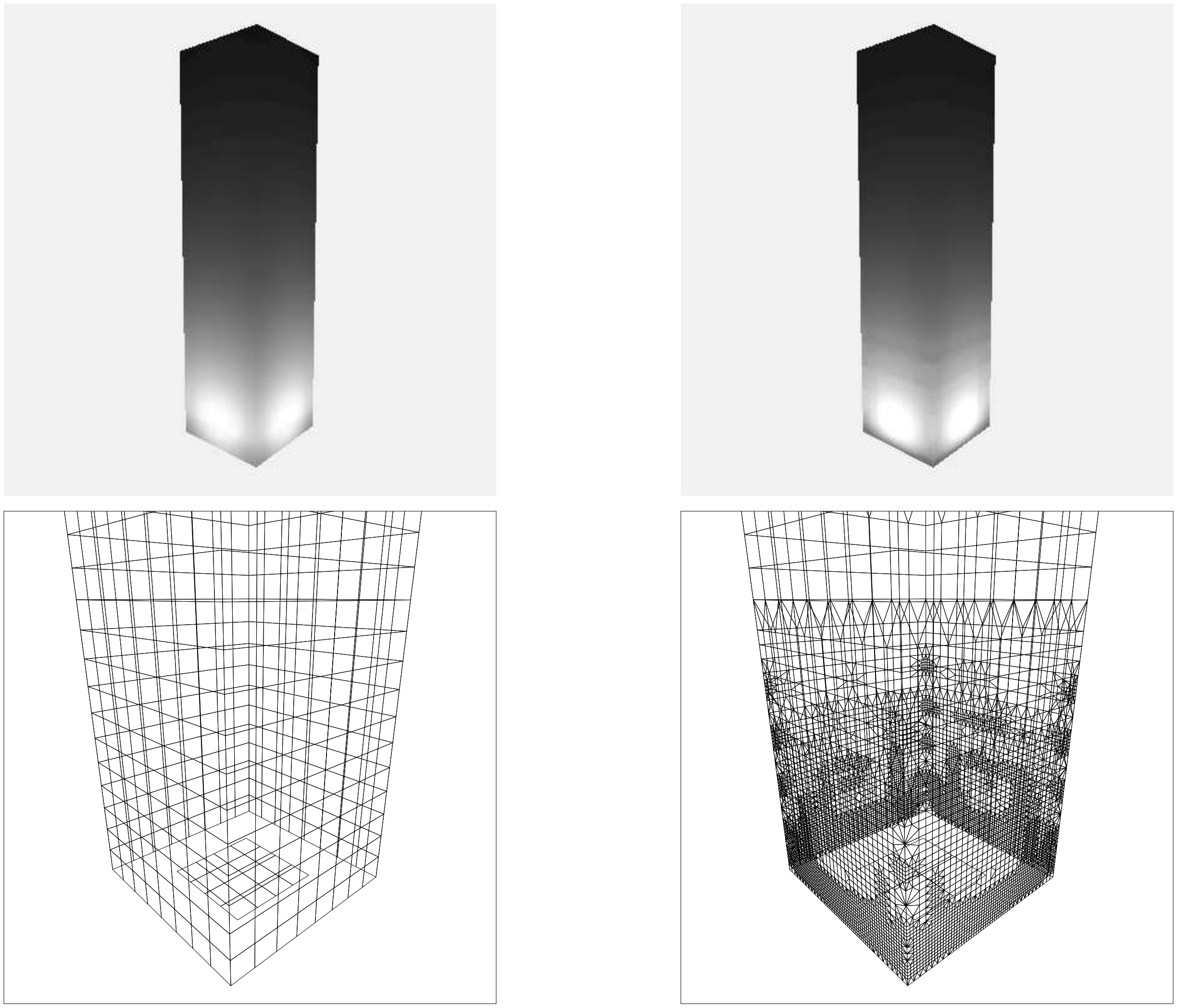


Figure 43: Tube experiment, image and corresponding mesh for progressive radiosity (left) and progressive radiosity with substructuring (right) at reflectance = 0.7.

ence of a number of shadow discontinuities in the scene. These two methods offer much the same error until the last scene, where the hierarchical method gains the edge. At this stage, the progressive methods spend more and more time shooting light from the walls of the scene, which are increasing in area, whereas the hierarchical method uses roughly a constant number of links between a wall and other parts of the scene. (We have observed that for an unscaled complex scene (not pictured), progressive radiosity is not penalised so heavily in this manner, and outperforms hierarchical radiosity.) In this scaled scene, however, hierarchical radiosity is the better choice for a scene of 30 or more chairs.

The reason that hierarchical radiosity starts to do better than substructuring radiosity is apparent from the time-to-convergence graph shown in **Figure 46**.Whereas the convergence times of the progressive methods increase quadratically, the convergence time of hierarchical radiosity is roughly linear with scene complexity. This qualitative difference in speeds of the two methods is one of the most interesting results from the complex experiment.

The two methods produce different shadowing artifacts, as can be seen in **Figure 47**, which shows a close-up of the shadows cast by a chair for the reference, substructuring, and hierarchical solutions. The substructuring shadow is blocky to the left of the chair, because the gradient was not large enough to cause further subdivision. This is exacerbated because later shots further subdivided the mesh in that area, preventing the shadow from being smoothed as much as it might have been. (Smoothing takes place on the scale of the leaf elements, so the shading of an element that was subsequently further subdivided is smoothed only at its edges, rather than over the whole element.) The primary problem with the hierarchical shadow is that a

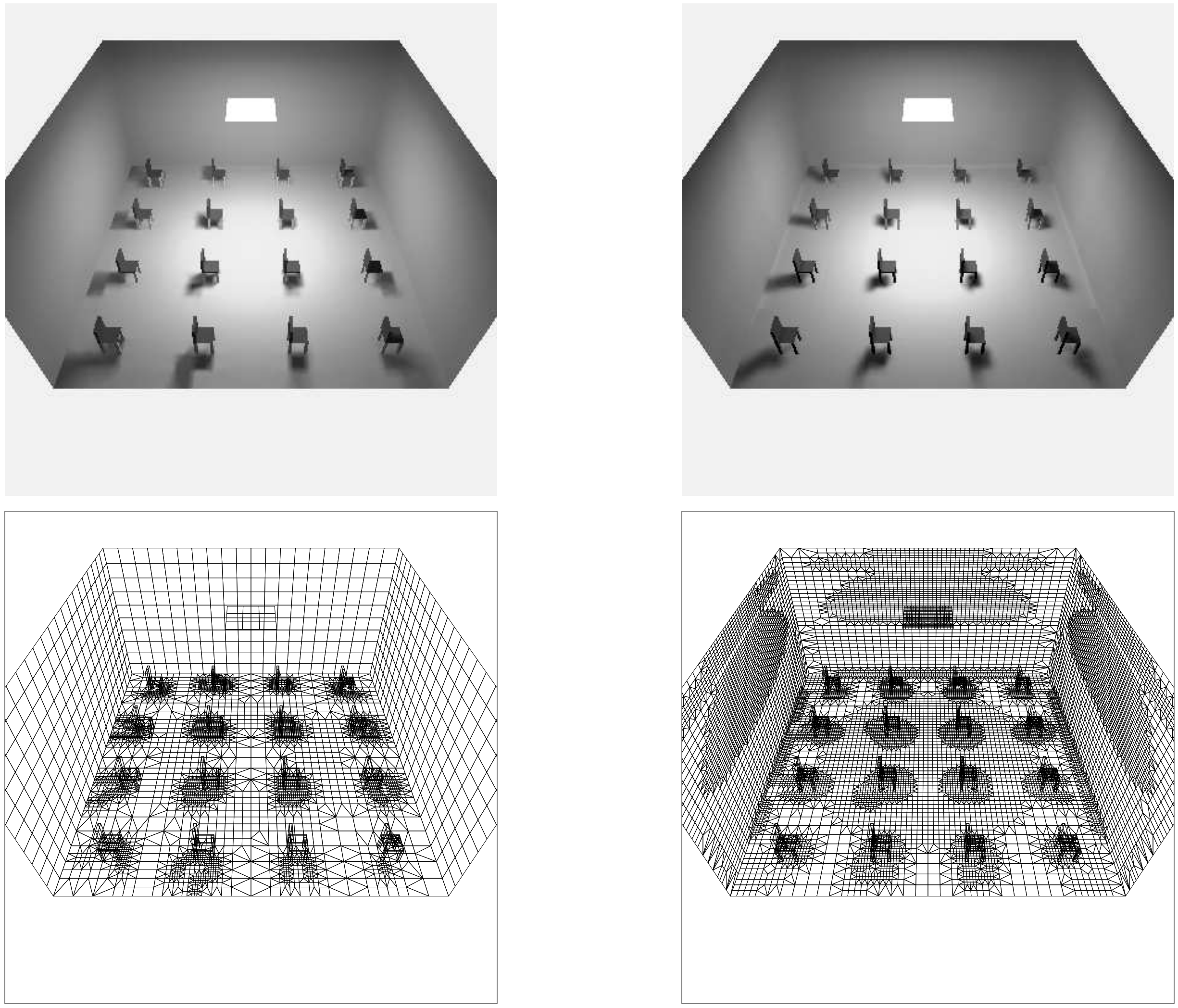


Figure 44: Complex experiment, image and corresponding mesh for substructuring radiosity (left) and hierarchical radiosity (right) for 16 chairs.

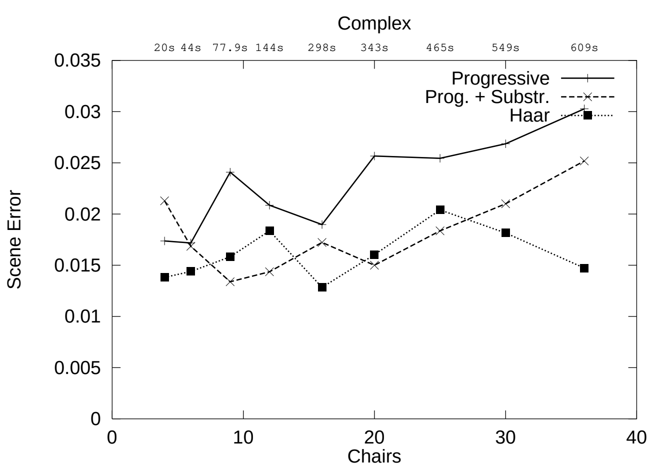


Figure 45: Complex experiment, equalised error for progressive and hierarchical radiosity.

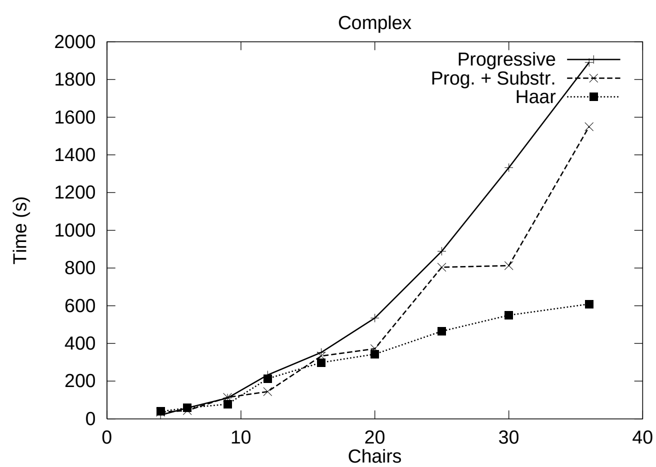


Figure 46: Complex experiment, time to convergence for progressive and hierarchical radiosity.

section has been missed towards the back of the chair because of triage—at a higher level in the element hierarchy all rays missed the rear-left chair leg, so all of its child elements were incorrectly also assumed to be completely unoccluded. Finally, substructuring misses the shadow on the seat of the chair, due to the initial receiving mesh not being fine enough to capture it.

Wavelet

Visibility testing for higher-order wavelets does not perform well in the complex experiment for either of the two multiwavelet quadrature methods we tried (see Section 2.4.2), as can be seen in **Figure 48**. The top image shows a solution using the fractional-visibility method, where visibility is factored out of the quadrature: it is overly blocky. The middle image shows the visibility-in-quadrature approach, which looks smoother, but still has (sometimes large) discontinuities between elements. We found that the objective error of both these solutions (and those for other complex scenes) were much the same; the greater smoothness of the visibility-in-quadrature approach does not result in a better fit to the reference solution. Both methods are clearly inferior to the reference and the Haar basis solutions in **Figure 47**.

The cause of the problems with the fractional-visibility approach is that it assumes visibility is constant across the element, even though with the higher-order methods the radiosity is no longer constant. The visibility-in-quadrature approach handles the variation of visibility across individual elements better, but because it samples visibility on a 2×2 grid, rather than a 4×4 grid, it suffers much more from aliasing problems. Because of this, fractional visibility should be used instead for refinement decisions, and visibility-in-quadrature only for calculating the final transfer coefficients when a link is formed: we have found that the use of triage with visibility-in-quadrature leads to large ‘dropouts’ in shadows.

Similar artifacts to the ones shown here are present in Figure 14 of Gortler et al. [22], which uses fractional visibility. These discontinuities are much less noticeable in Figure 8 of Gershbein et al. [17], which uses visibility-in-quadrature, although the dimness of the picture also plays a part. The discontinuities in **Figure 48** are much more apparent because our scene is well-lit by the overhead light source, and we have zoomed in on a small part of the floor.

In **Figure 49** the equalised error plot shows the Haar basis is clearly more accurate than the higher order methods, for a given amount of time, for the complex scene. Most of the other bases are grouped closely together, except for the M3 basis, which is the least impressive by some distance. The Haar basis does considerably better than the other bases because of the visibility factors discussed above; although the higher-order methods use visibility to control subdivision in the same manner as the Haar basis, the visibility is still treated as constant between each linked pair of elements. In cases where the visibility error predominates, the benefit of the higher-order wavelets are masked by the constant-order visibility factors. Worse yet, because the higher-order methods create fewer links, they are more affected by visibility error than the Haar basis is.

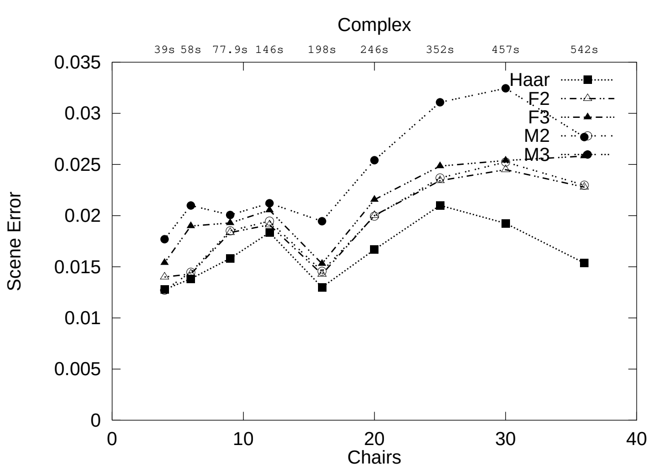


Figure 49: Complex experiment, equalised error for wavelet radiosity.

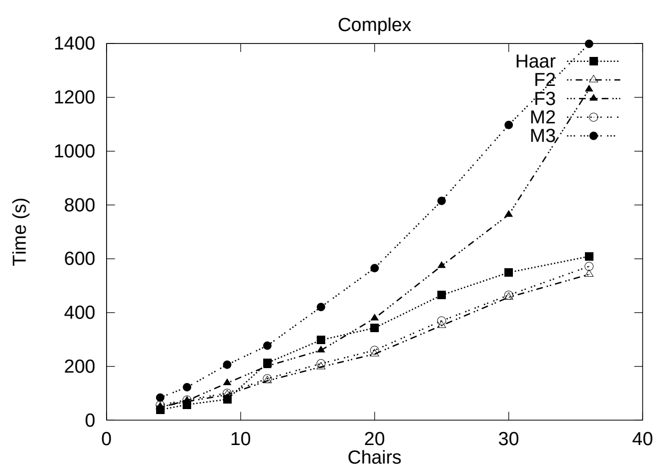


Figure 50: Complex experiment, time to convergence for wavelet radiosity.

Finally, comparing the multiwavelet bases to the flatlet bases, we see that the extra time overhead of the multiwavelet bases apparent in **Figure 50** counts against them, especially in cases such as these where a large number of links are created. Fitting

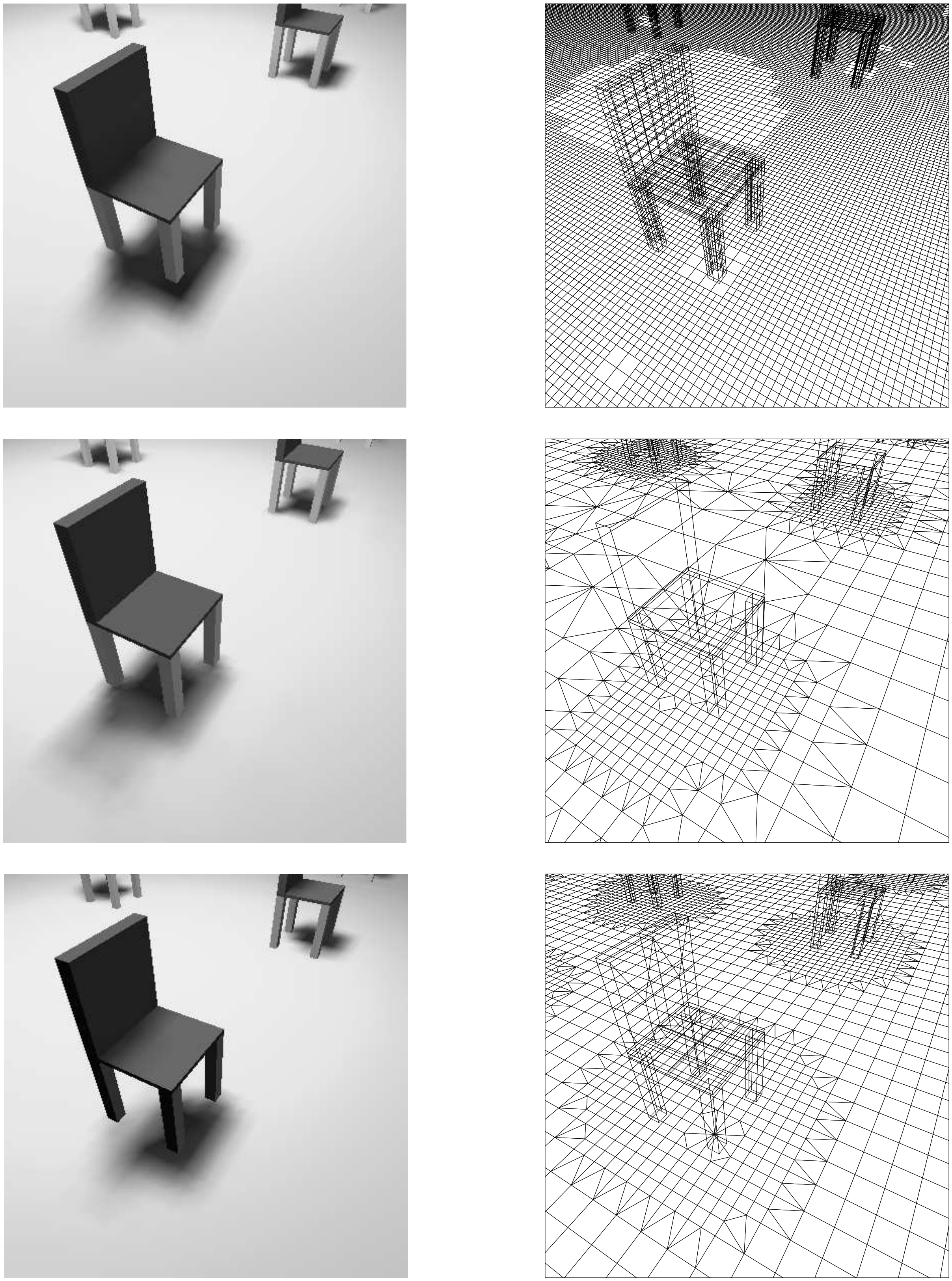


Figure 47: Shadow resolution for different radiosity methods. Top, the reference solution, middle, progressive radiosity with substructuring, and bottom, wavelet radiosity with the Haar basis.

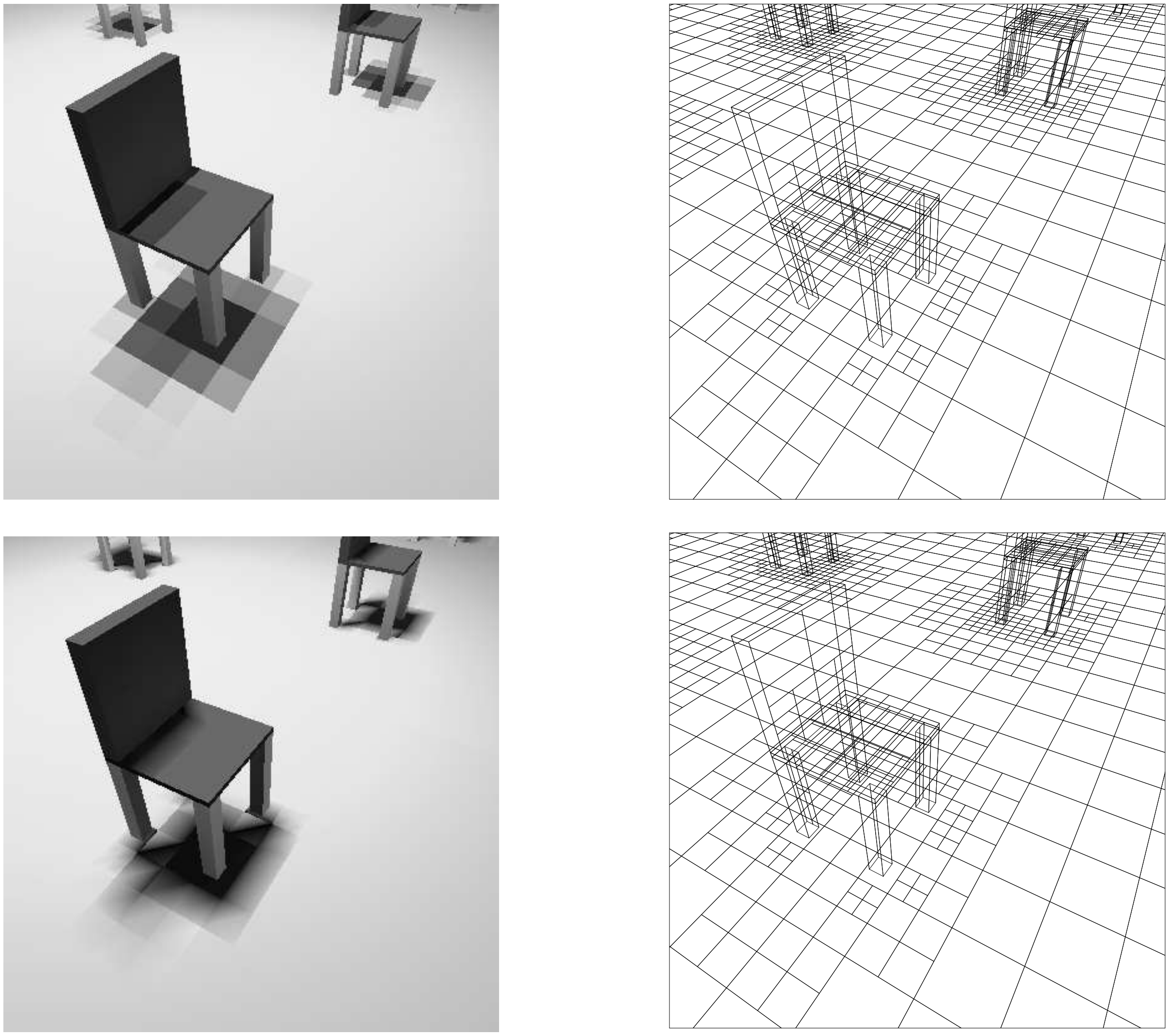


Figure 48: Top, wavelet radiosity with the M2 basis (visibility is factored out of the quadrature), and bottom, the M2 basis with visibility incorporated into the quadrature.

the function $a + bx^c$ to the data series shows that while the Haar basis has approximately linear time complexity, the second-order wavelets are $\Theta(k^{1.3})$[1], and the third-order wavelets approach $\Theta(k^2)$.

The biggest drawback of the higher-order wavelet radiosity techniques, and to some extent wavelet radiosity itself, is memory consumption. As **Figure 51** shows, the memory consumption of the wavelet algorithms is large, especially compared to the memory usage of the progressive methods, shown in **Figure 52**, which never exceed 1MB of storage in these tests. The multi-wavelet and flatlet bases of the same order have similar memory requirements. In this particular run, the Haar basis reached 40MB of storage, and the M3 basis reached 550MB of storage for 36 chairs, thus requiring a lot of paging. All the higher-order methods produced a scene with up to 50% higher error than the Haar basis, so this was not a case of these methods trying to do too much work, and thus consuming too much memory. (Because of paging the clock times for the higher order wavelets were many times longer than the CPU time graphed in **Figure 50**, so that graph is perhaps too generous to them. Also, this is for a scene of 977 polygons—only moderate complexity.)

1. We use the $O()$ notation to indicate asymptotic complexity bounds, and $\Theta()$ to indicate empirically measured behaviour. O denotes an asymptotic upper bound, and Θ symbolizes asymptotic proportionality.

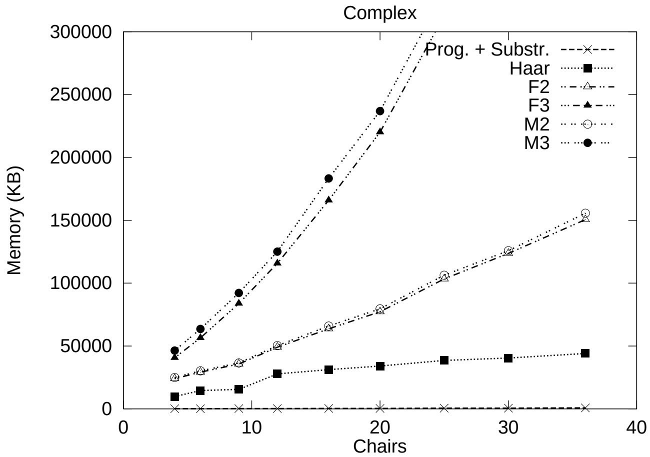


Figure 51: Complex experiment, wavelet memory usage.

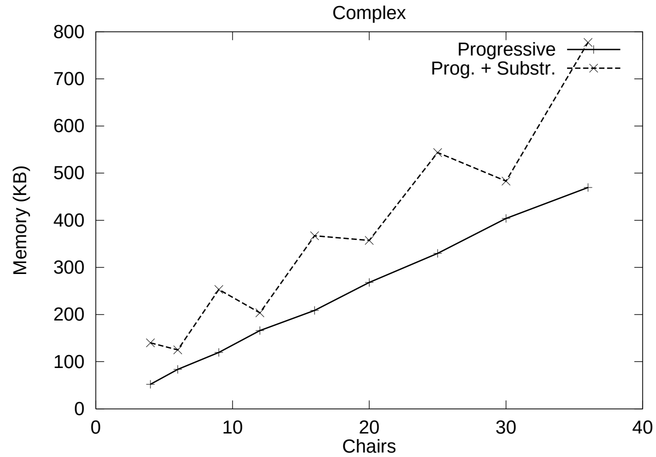


Figure 52: Complex experiment, progressive memory usage.

An obvious suspicion here is that for some of these runs we may have chosen too high an accuracy, and created too many links. To combat this we tried to choose an epsilon that produced the lowest possible memory use for a reasonable amount of error. The effects of adjusting epsilon for wavelet radiosity are examined in Section 4.2.6. Lowering the levels of reflectance in the complex scene would also help reduce memory use, but these levels are not excessive when compared to general radiosity scenes.

Apart from the large constant difference between the memory use of the hierarchical and progressive methods (the Haar basis starts at 10Mb for the 4-chair scene, the substructuring method at 150KB), there is also an asymptotic difference. The progressive methods' memory consumption is linear with scene complexity, as we would expect, given that the surface area of the scene is directly proportional to the number of chairs in the scene. The higher-order wavelet methods' is superlinear: empirically memory use is close to $\Theta(k^{1.28})$ for second order methods and $\Theta(k^{1.62})$ for third order methods. The data for the Haar basis is harder to fit, but is approximately linear. In the wavelet radiosity algorithm, memory and time complexity are primarily controlled by the number of links created: Section 4.3.4 discusses how the link complexity varies between $\Theta(k)$ and $\Theta(k^2)$.

Unfortunately, the limits on memory are often hard, while those on time are not. Most modern workstations are equipped with at least 64Mb of memory, but as we have seen, this still would entail using a large amount of swap.While clustering will help this problem, it is not clear whether it will solve it satisfactorily. For very large scenes, while the wavelet methods may have better time/error performance than progressive radiosity, their memory consumption will prove prohibitive. This is particularly true of higher-order wavelet radiosity.

4.1.5. Lights

In our final experiment we examine the affect of varying the number of light sources on the radiosity methods. For this we use a scene similar to the complex scene, in which there are m light sources, and 12 chairs, for a total of k = 328 + m polygons.

Progressive

The running times of the progressive methods will obviously be affected by m; as the number of lights increases, the number of patches to be shot from also increases. We can imagine constructing a worst-case scene where running time increases linearly with the number of light sources, but how running time is affected in more conventional scenes is an interesting question. In **Figure 53** we see the results of running the progressive methods on the lights experiment; reassuringly they are sub-linear. This is due to the overlapping of the light sources' areas of influence, which only increases as we add more lights. The more these areas overlap, the more shooting steps can be 'shared' between two light sources, and the further below our hypothetical linear relationship the actual curve will drop. The subdivision performed by the substructuring methods does not appear to alter the shape of its curve to any great extent.

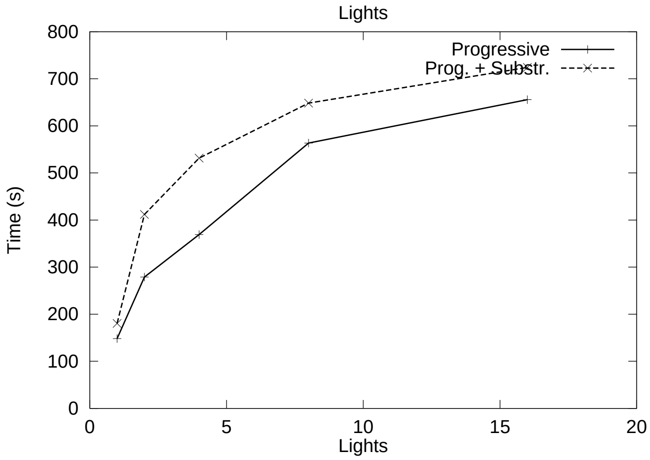


Figure 53: Lights experiment, time to convergence for progressive radiosity methods.

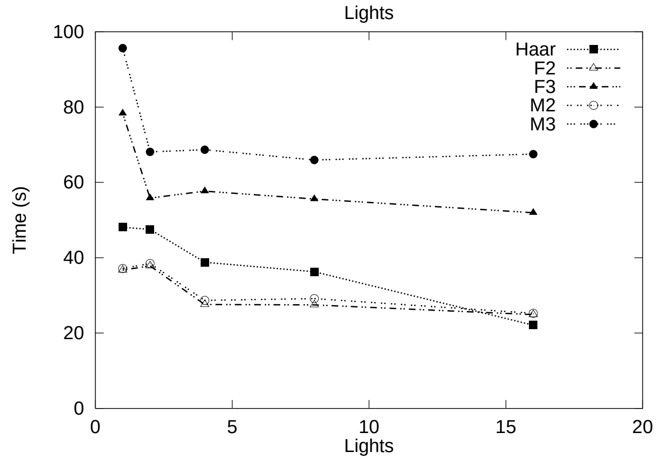


Figure 54: Lights experiment, time to convergence for wavelet radiosity methods.

Wavelet

Figure 54 shows the same situation for the wavelet methods. Overall, note that the wavelet methods are running much faster than progressive or substructuring: 10-30 times faster for 16 lights. We would not expect the number of light sources to affect wavelet radiosity much, as its gather steps are independent of the lights in the scene. Of course, patch brightness does affect the creation of links due to the brightness-weighted oracle we use, so we expect to see some effect due to this. Indeed, the convergence times are largely constant, with the exception of the Haar basis, which surprisingly decreases as the number of light sources increases. We hypothesize that this is due to the larger number of light sources creating a smoother illumination of the scene, and thus necessitating fewer links. (The higher order methods use from 20,000 to 30,000 links, whereas Haar drops from 90,000 to 35,000 links as we go from 1 light to 16.)

4.2. Other Experimental Results

In this section we look more closely at certain aspects of our results, such as visibility, over-refinement, the effects of reflectance and epsilon, and meshing.

4.2.1. Components of Solution Time

Visibility calculations can be one of the most time-consuming parts of a radiosity algorithm. **Figure 55** shows the fraction of total runtime that the progressive and wavelet methods spend tracing rays for the complex experiment. The first observation is that the Haar wavelet method is dominated by visibility calculations, spending between 60% and 80% of its time on them. It creates many more links than the higher order methods, thus requiring more visibility tests than them, and its fractional visibility estimate requires 16 rays cast per transfer to the progressive method's 1; as a result the progressive method spends much less of its time on visibility (40%). The higher order wavelets also spend more time calculating transfer coefficients for each link, so as the basis order increases, visibility calculations become a much smaller part of the total computation time—for the F3 basis, they drop to under 20%. Noticeably, the fraction of time spent ray-tracing stays reasonably constant as scene size increases.

We claimed in Section 2.6 that our visibility cost was $O(k)$, with a low constant. **Figure 56** shows the time taken to cast a ray for the various complex scenes; the simplest requires around 30 s per ray cast, and the most complex (with 10 times as many polygons) takes around 50 s.

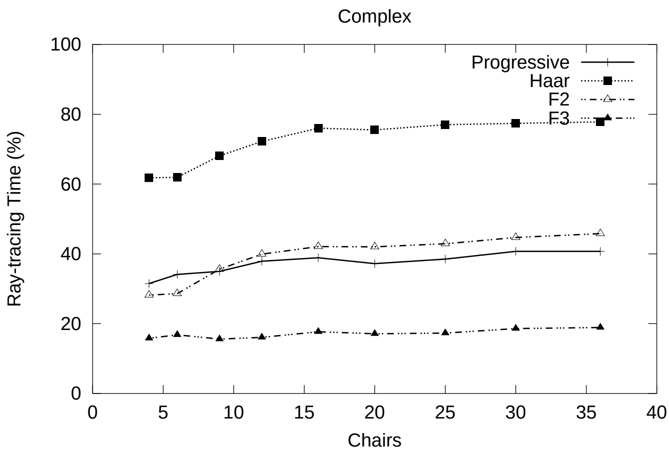


Figure 55: Complex experiment, ray tracing time as a percentage of total run time.

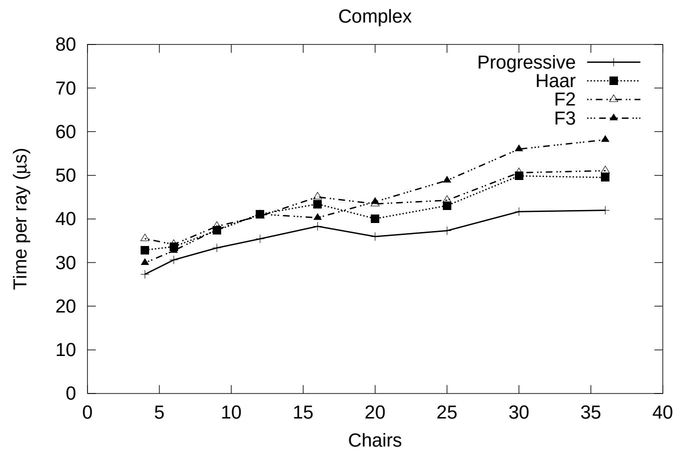


Figure 56: Complex experiment, time taken to cast a ray.

These results were collected by analysing the various methods with a PC-sampling profiler, which produced a list of estimates of the runtime spent in each of the method's routines. The time spent in all ray-tracing-related routines was summed to produce the graphs above. We can roughly group the remaining routines into three more categories: the results for a selection of the methods run on the complex scene with 30 chairs can be seen in **Table 5**. The 'Quadrature' category includes the calculation of form factors, and in the case of wavelet radiosity, transfer coefficients. The 'Solve' category includes time spent shooting or gathering radiosity. 'Ray-tracing' covers all visibility calculations, as above, and all other routines were classified as 'Other'. Notably, the F3 method spends more time on quadrature than solution, presumably due to the constant refinement of links causing recalculation of transfer coefficients at ever finer scales.

Method	Quadrature	Solve	Ray-Tracing	Other
Substructuring	15	29	46	10
Haar	3	14	76	7
F3	39	28	20	13

Table 5. Complex scene with 30 chairs, percentage of time spent on various tasks.

4.2.2. Analysing Visibility

In **Figure 57** we see the number of rays hierarchical radiosity casts as solution time for the various complex scenes. The curves are roughly linear, and show that Haar issues ray-cast calls at a roughly constant rate, independent of scene complexity. As the scene complexity increases the lines extend further, and also shift to the right, signifying the slightly slower rate at which rays are cast. Where the lines tail off is where the algorithm has resolved shadows to its satisfaction, and is working on other parts of the solution, or in a gather stage. **Figure 58** shows how this picture changes as the order of the basis increases: the slopes decrease, and the algorithm spends relatively less time refining and more time solving. This graph also provides a snapshot of the relative speeds of the different bases; the M3 basis is noticeably slower than the F3 basis at issuing ray casts, for instance, whereas the difference between the M2 and F2 bases is not so pronounced. Finally, in **Figure 59** we see the average number of rays cast per link for scenes of varying complexity. This will have a maximum of 16 when all links require a visibility test upon formation. All the curves follow the pattern of a high initial-rays-cast per link, which then tails off once shadows have been satisfactorily resolved.

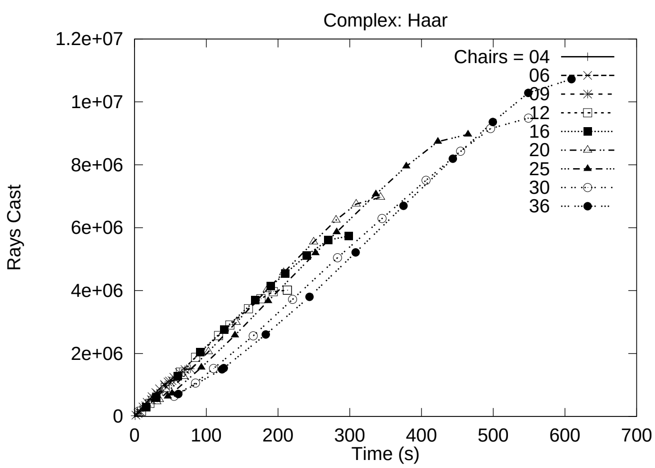


Figure 57: Complex experiment, rays cast over time for hierarchical radiosity.

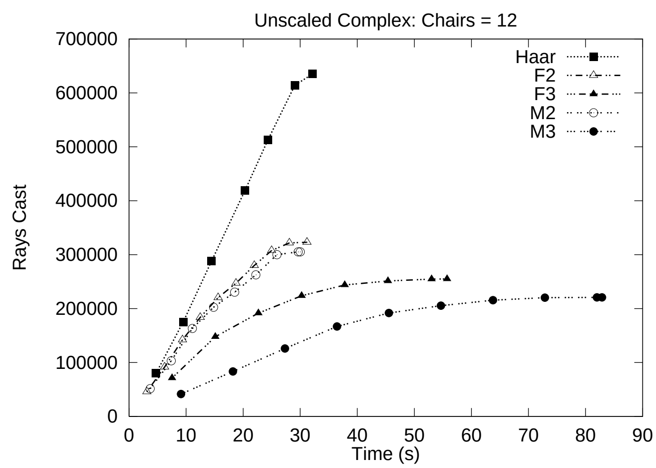


Figure 58: Complex experiment, rays vs. time to convergence.

In **Figure 60** we see a comparison between substructuring and hierarchical radiosity; the graph shows how many rays each method casts to achieve a particular error level for a mid-complexity scene. While the progressive curve is smooth, the hierarchical curve has a prominent ‘knee’, beyond which casting more rays gives only a small improvement in error. In almost all cases the progressive method requires fewer rays cast to achieve a given error level. For scenes where the ray-tracing or visibility method employed is not efficient, substructuring radiosity will be favoured. These graphs also suggest that a better occlusion-testing method is needed for wavelet radiosity; a method that exploits the multi-resolution nature of the algorithm would seem to be the most promising approach. (Links between different levels of the element hierarchy need visibility estimates across variously-sized sub-spaces of the scene.)

4.2.3. Wavelet and Progressive Solution Curves

In this section we examine the character of the time/error curves for progressive and hierarchical radiosity. The hierarchical curves have an initial linking phase during which radiosities remain unchanged, followed by a rapid drop to a low level of

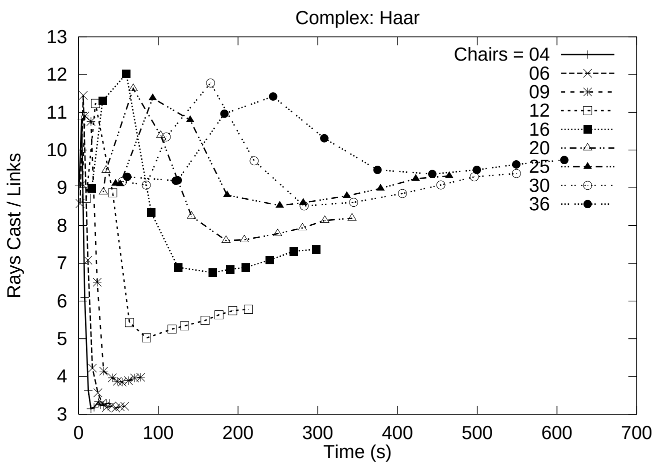


Figure 59: Complex experiment, rays cast per link.

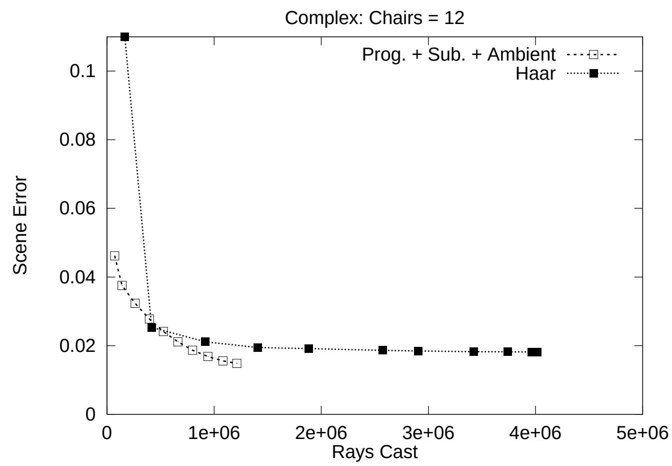


Figure 60: Complex experiment, rays needed for a given error level.

error; the error decreases only slowly after this. By contrast the progressive error curves decrease slowly but smoothly over the entire course of a run towards some minimum value. **Figure 61** shows these curves for the complex scene with 36 chairs; the sharp drop in error for the two wavelet methods marks the end of the initial linking phase.

Figure 62 shows how these curves change with complexity. For both the high and low complexity scenes, the hierarchical method reaches a lower error faster, but after that the error decreases only slowly, towards a lower error bound controlled by epsilon. The progressive method's error decreases more slowly, and for low complexity scenes it never reaches the hierarchical error. For the more complex scene, substructuring is again beaten in the short term, but eventually catches up with hierarchical radiosity before that algorithm terminates, and then passes it. For the hierarchical method to compete with this, epsilon would have to be decreased, resulting in a longer solution time, and greater memory consumption.

Given the long tails of the hierarchical time/error curves, it obviously could be worthwhile to terminate the algorithm before all links have been fully subdivided. A promising approach may be to terminate the algorithm after a certain number of links have been created, or utilise the decreasing-epsilon approach described in [14]. It may also be worthwhile to increase epsilon at lower levels of the element hierarchy.

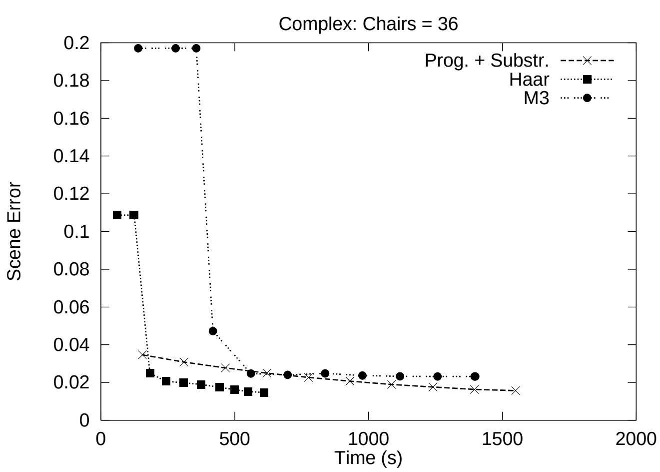


Figure 61: Complex experiment, time/error curves for substructuring radiosity, and wavelet radiosity using the Haar and M3 bases.

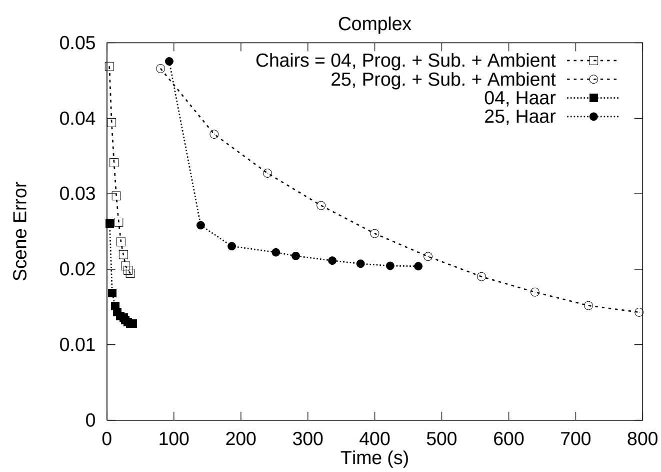


Figure 62: Complex experiment, progressive and hierarchical radiosity for two complexities.

4.2.4. Over-Refinement

A common problem with the substructuring version of progressive radiosity in well-lit or highly reflective scenes is illustrated in **Figure 63**. The two solution lines to the left of the graph show the usual situation for progressive radiosity; the vanilla version is outperformed by the substructuring version. These are solutions from a low-reflectance scene; when the reflectance is increased, as on the right hand side of the graph, we see that the vanilla algorithm quickly reaches an error level that the substructuring version can't match for quite a while. In such a situation the substructuring algorithm spends time refining the mesh to receive radiosity from the initial patches shot, even though later shots from other patches will make this refinement unnecessary. **Figure 64** shows how this problem worsens for higher reflectances; while substructuring is faster or as fast as progressive radiosity for reflectances below 0.4, its convergence time soars as reflectance increases.

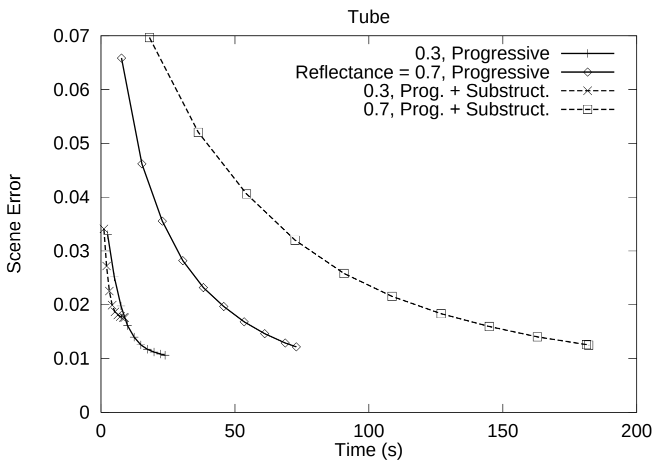


Figure 63: Tube experiment, error at convergence for progressive radiosity methods at two reflectance levels.

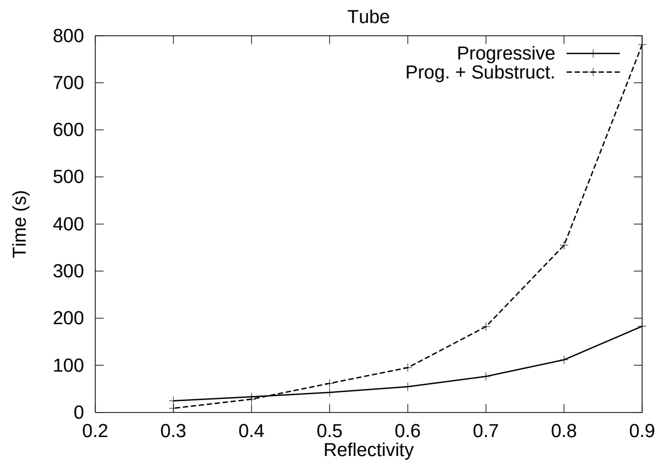


Figure 64: Tube experiment, time to convergence for progressive radiosity methods. Illustrates over-refinement in progressive radiosity with substructuring.

While substructuring brings out shadow boundaries well, it tends to overrefine the mesh during the initial few shots. We can see this effect at work in **Figure 65**; in the first picture, only the right hand part of the light source has been shot, and the algorithm has meshed finely around this initial bright spot. In the second picture we see that after the algorithm has run further, the entire area in front of the light source is white, and in fact this area could have been meshed relatively coarsely.

4.2.5. Reflectance

In this section we explore in more detail the effects of the reflectance experiments on certain aspects of the progressive and hierarchical radiosity methods. In particular we examine a serious problem with hierarchical radiosity in some situations where the element hierarchy is deep.

Figure 66 shows an interesting result; in the box scene the number of elements required to produce a solution for a given reflectance level seems to increase almost linearly for substructuring radiosity. **Figure 67** shows the same situation for the tube scene. Thus with the termination criteria we have used, a useful rule-of-thumb for these scenes is that the number of elements in the mesh increases approximately linearly with scene reflectance.

Figure 68 shows the time/error curve for hierarchical radiosity at a high reflectance level. At first glance the curve seems wrong; the error is slowly increasing as the solution progresses, rather than decreasing. To see why this is so, we need to look at the centre image in **Figure 69**, which shows a high-reflectance solution, using the form-factor approach outlined in Section 2.1.2. The radiosity in a thin strip along the edge where each pair of patches meets is overly bright. (Comparing the image with the reference solution shown in **Figure 15**, we see that the real solution is darker at the corners, not brighter.) This artifact is even more apparent in the left-hand image, where point-to-point form-factors were used with no singularity correc-

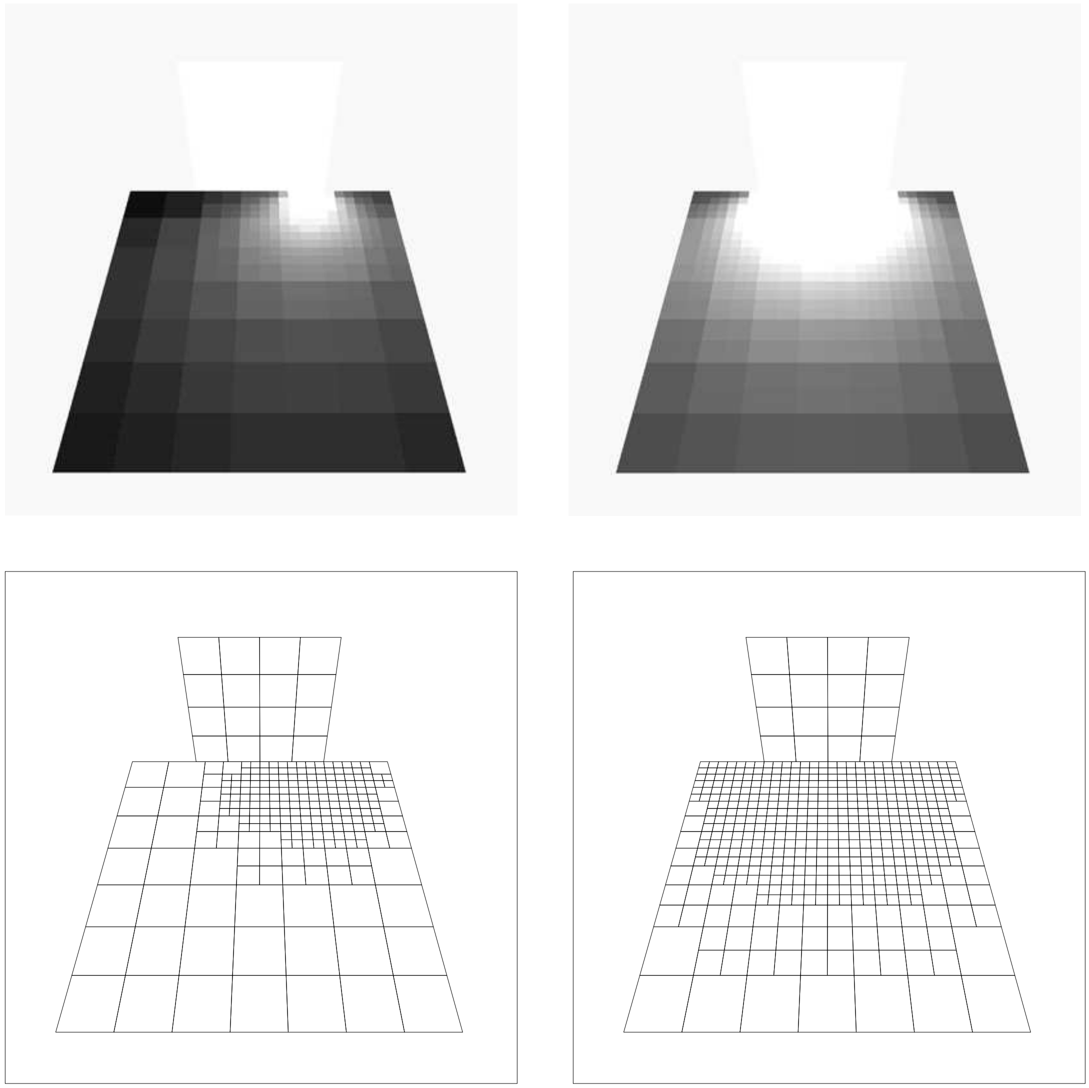


Figure 65: Over-refinement in Progressive Radiosity. On the left, the first shots from emitter patches cause heavy subdivison, because most of the receiver is still unilluminated. On the right, shots from the rest of the emitter have illuminated the rest of the receiver, making some of the earlier refinement redundant.

tion. Experience has shown that as the link hierarchy deepens, these ‘bright strips’ become more noticeable; hence the increasing error over the reference solution as the solution progresses, and links are pushed deeper into the element hierarchy.

In hierarchical radiosity, the effective form-factor from a patch to any leaf element is the sum of all links from that patch to the element itself and its parent patches. As the hierarchy deepens, there will be more such links summed. This suggests that a possible cause of the problem is systematic error being introduced when these partial form-factors are summed. If our form-factor estimates consistently over-estimate the form-factor to elements in the corners of the box, we would expect this error to accumulate as the hierarchy deepened. If this were the case the problem could be ameliorated by using more accurate form-factors. However, using a highly-accurate form-factor, as in the right-hand image in **Figure 69**, brings only a slight improvement; the artifact is still noticeable.

The error is in fact systematic, not because of the errors in the form-factor integrals accumulating, but because the *projection* error in the radiosity transfers accumulates. This projection error occurs when we project the radiosity function across a receiving element into the basis being used to represent the radiosity of that patch. **Figure 70** shows an example of the problem. The top part of the diagram illustrates how the radiosity across the element of the horizontal receiving patch closest to the vertical

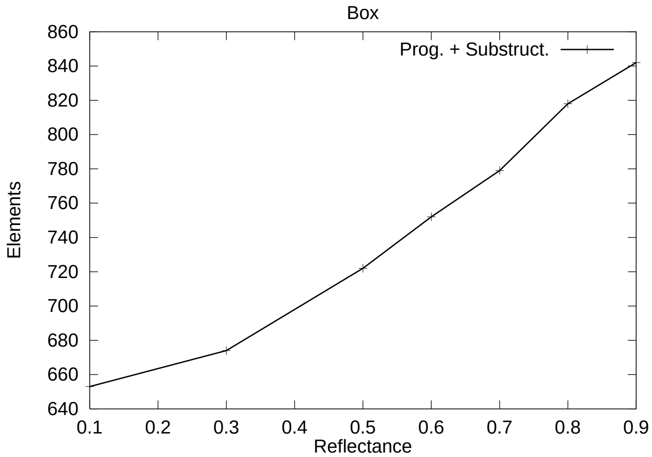


Figure 66: Box experiment, elements at convergence time increase approximately linearly with reflectance.

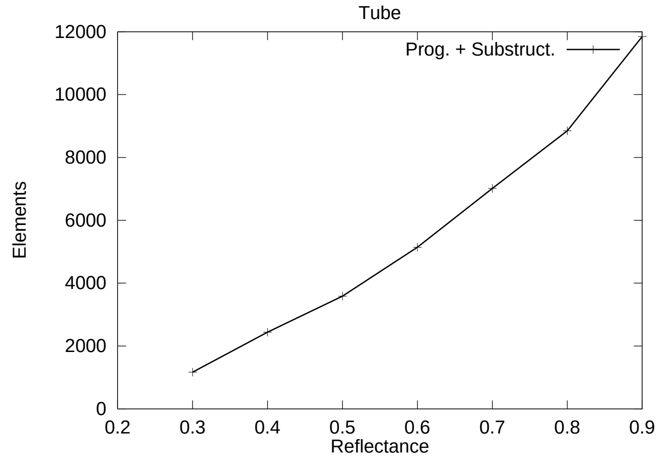


Figure 67: Tube experiment, elements at convergence time increase linearly with reflectance.

source patch would typically be composed from contributions of sub-elements of the source patch. (We assume in this example that the source has constant radiosity.) In this case there are three levels to the link hierarchy. The bottom part of the diagram shows both the approximate radiosity functions due to the source elements, and the actual radiosity functions. From these we can see that the projection into the Haar basis consistently overestimates the radiosity function close to the corner for source elements that aren't adjacent to the receiving element. (For all such elements the radiosity distribution should be zero at the left-hand side of the receiving element, so the constant approximation to the function must always over-estimate the actual function at this point.) When these distributions are summed to get the overall distribution, the result is an over-estimate of the radiosity for the receiving element, as we see on the right of the diagram. The same problem also occurs with higher order bases. It is diminished considerably because these bases fit the radiosity function better than the Haar basis, most particularly the left-hand zero mentioned previously in the case of the multiwavelets. However there is still some error, and it accumulates the same way as for the Haar basis.

Close examination of the images in **Figure 69** reveals another less obvious artifact. The floor of the box should be tinted green on the right, and red on the left, and interpolate smoothly between the two extremes in between. Instead there is a noticeable discontinuity between the left 'red' half of the floor, and the right, 'green' half. This is caused by the low resolution of the links formed for the secondary illumination between the walls of the box. (As opposed to the primary illumination carried by the links between the light source and the walls.) Although a suitable epsilon was chosen to produce a well-meshed result for first-

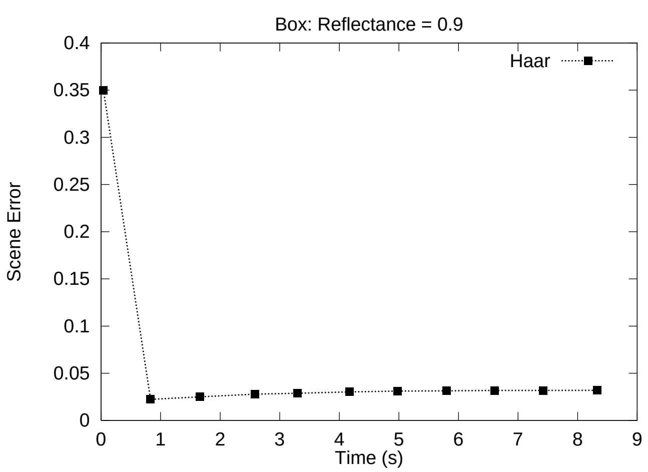


Figure 68: Box experiment, singularity error increases a bit at high levels of subdivision for hierarchical radiosity during a run.

Figure 69: Singularity problems in the box experiment with reflectance = 0.9 causing over-bright regions near abutting surfaces. From left to right, the solution was calculated with: (1) point-to-point form-factors only, (2) point-to-point form-factors with polygon-to-point correction (as used in the experiment itself), and (3) high-accuracy form-factors calculated by sampling the polygon-to-point form-factor in a 10 by 10 grid over each receiving patch. Exaggerated versions of these images appear in **Figure 96** on page 85.

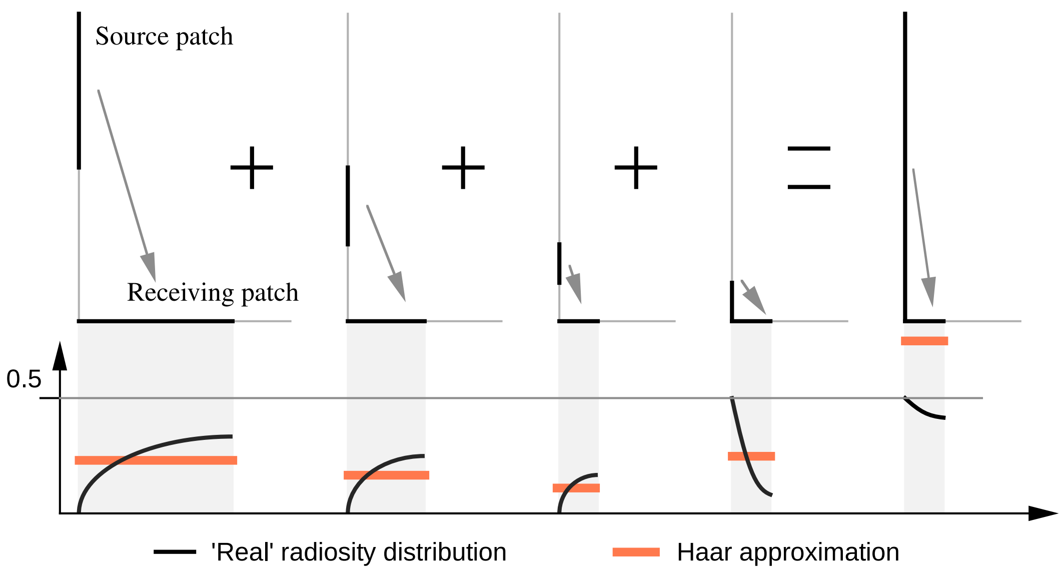


Figure 70: Accumulated error close to the singularity in wavelet radiosity. The bottom of the diagram shows the actual form factors between the subsections of the source and the receiver, as well as the Haar approximations to them. The resulting sum overestimates the form-factor between the entire source patch and that part of the receiving patch that is closest to it. Exaggerated for clarity.

order illumination, this did not assure sufficient meshing for the secondary illumination in this case. The problem here is in the brightness metric used to weight link subdivision in our simulation; it is the hue of the floor that changes from left to right, rather than its brightness. This could either be addressed by using separate links for each of the colour components, or using a better metric to measure the significance of energy transferred across a link.

The tube scene illustrates another potential problem with hierarchical radiosity. At high reflectances the results can become highly sensitive to the setting of the threshold for subdivision, and can act in non-intuitive ways. **Figure 71** shows scenes generated with large and small epsilons. Though we would expect the scene on the right to be better, it is demonstrably worse than that on the left. This occurs because the number of links generated by the second solution increases sharply over the first, and the accumulated error in the form-factors prevents as accurate a solution being reached.

Figure 71: Epsilon instability in the tube experiment with reflectance = 0.9, and using hierarchical radiosity. On the left is the solution for epsilon = 0.02, and on the right the solution for epsilon = 0.01.

4.2.6. Varying Epsilon for Wavelet Radiosity

It is useful to know the behaviour of the wavelet algorithm as we vary epsilon, the parameter that controls its accuracy. (Recall that epsilon controls the refinement of links, and thus elements. As epsilon is decreased, a finer mesh is created, and a more accurate solution is possible.) **Figure 72**, **Figure 73**, and **Figure 74** show its effects on convergence time, error at convergence, and memory use at convergence, for the complex scene with 9 chairs. The memory and time curves are both approximately proportional to 1/epsilon. If we disregard the last point in both graphs, the relationship is approximately linear, so time is Θ(1/epsilon), and memory use is also Θ(1/epsilon). (We assume that the last point is sublinear because the subdivision limit, `edge_min`, starts to reduce the number of elements, and thus links, created at around this value of epsilon.)

Notably, there is a 'sweet spot' around a 1/epsilon value of 100 where the time and memory consumed are not extreme, and the error close to the minimum. Above this setting, decreasing epsilon results in minimal extra accuracy for a lot of extra time and space. In a related plot, **Figure 75** demonstrates a ‘memory wall’—at a certain error point, the memory used by hierarchical radiosity starts increasing rapidly for very little gain in accuracy.

The relationship between the number of links and 1/epsilon is shown in **Figure 76**. For all the bases the curves are for the most part linear. The last point of the Haar curve drops below linearity, but we suspect this also is due to the `edge_min` limit. This linear behaviour can be informally explained as follows: for any point on a surface in the scene, the form factors can sum to at most one. If we force all links to be subdivided such that their associated form factor is less than epsilon, then there can be at most 1/epsilon links. **Figure 77** is a close up of the behaviour of the F2 basis. For higher values of 1/epsilon, the curve is linear. As 1/epsilon decreases, and the form factors to top-level patches drop below epsilon, the curve becomes nearly constant, and eventually approaches the horizontal line $links_{min} = k^2$. The curves for the other bases display the same behaviour.

We can make use of the apparent linear relationship between 1/epsilon and memory use and time when setting epsilon for a solution: for instance, if we wished to halve the amount of memory used by the algorithm, we double epsilon, although this will of course result in a solution with greater error.

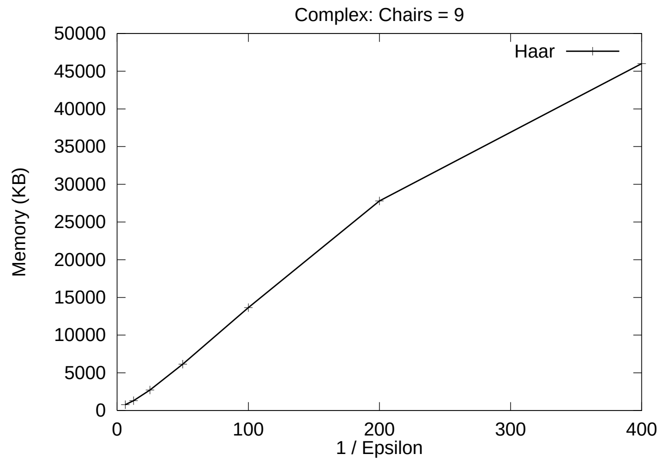


Figure 72: Complex experiment, effect of varying epsilon on memory for wavelet radiosity.

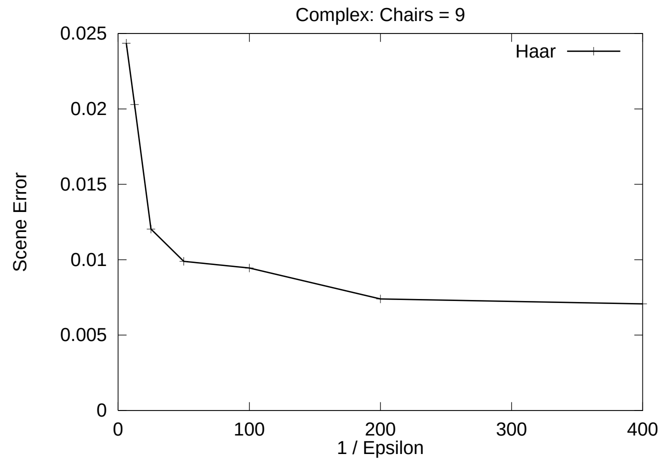


Figure 73: Complex experiment, effect of varying epsilon on error at convergence for wavelet radiosity.

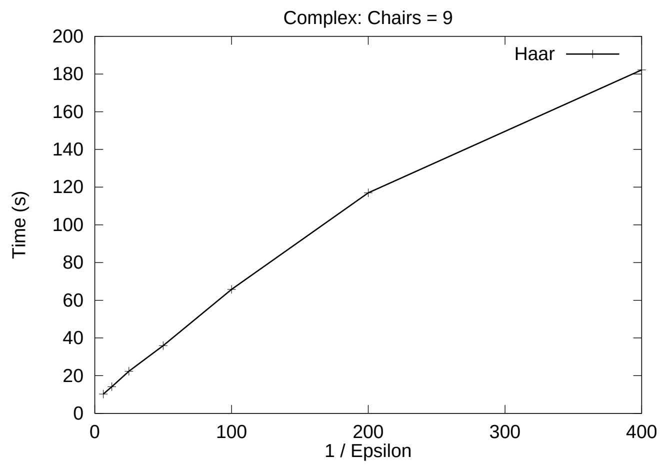


Figure 74: Complex experiment, effect of varying epsilon on convergence time for wavelet radiosity.

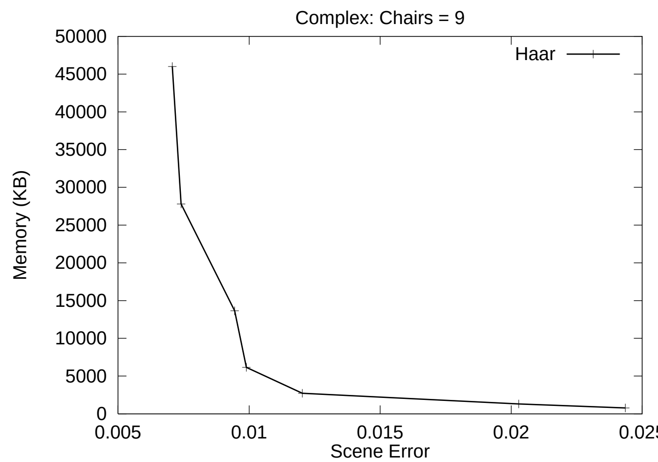


Figure 75: Complex experiment, memory use increases dramatically below an error level of 0.01.

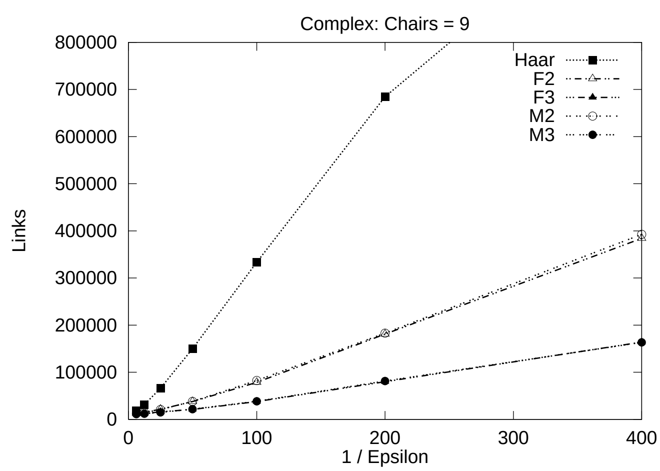


Figure 76: Complex experiment, effect of varying epsilon on links created by wavelet radiosity.

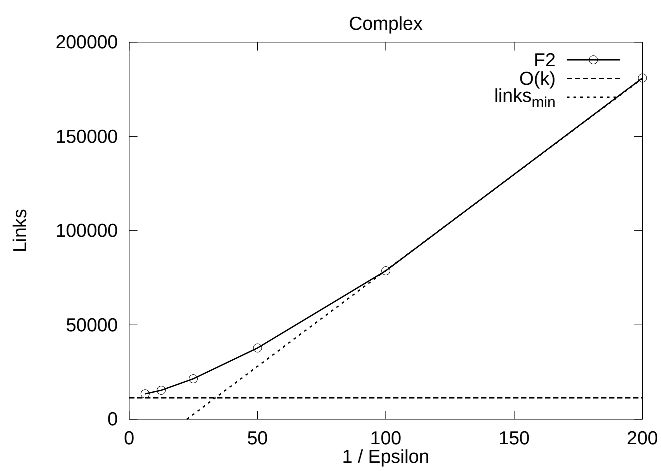


Figure 77: Complex experiment, measured and theoretical behaviour of link creation for the F2 basis.

4.2.7. Meshing

As we have seen, a good mesh can reduce error dramatically. The various methods differ in how they use a mesh and how the user controls refinement, but for all methods we can determine the number of elements in the mesh at convergence. In **Figure 78** we have plotted the scene error different methods achieve with a given number of elements in the mesh. For matrix and progressive radiosity this was done by varying the maximum edge length of the mesh directly, and for substructuring and wavelet radiosity by varying the refinement epsilon. Most notably, in the 5000 element range the higher-order wavelet methods get a smaller error for the same number of elements than either progressive or hierarchical radiosity. However, their elements are more heavyweight in terms of memory and computation time than the others'. Because of this, for a given time hierarchical radiosity can generate a solution with many more elements than the higher order wavelet methods, and thus find a more accurate solution in this time-frame. Progressive radiosity with substructuring utilises the number of elements in its mesh relatively poorly compared to the other methods, but because it subdivides its mesh for very little cost (recall that the substructuring does not effect the number of patches shooting light, and that there are no link data structures to be maintained), it still achieves comparable or better error in a given time period.

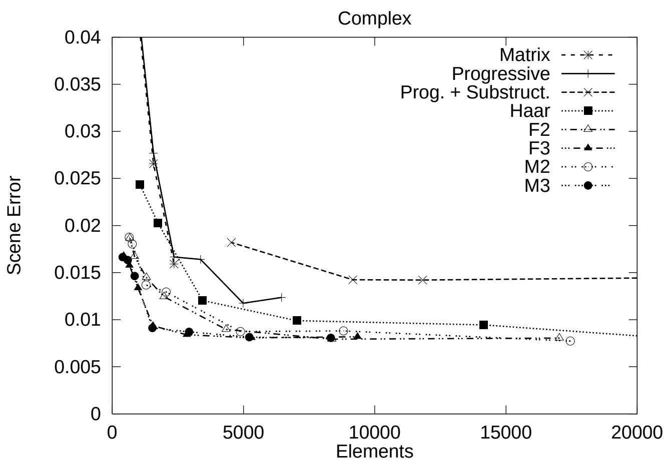


Figure 78: Complex experiment with 9 chairs, mesh density vs. error at convergence time for matrix, progressive, and wavelet radiosity

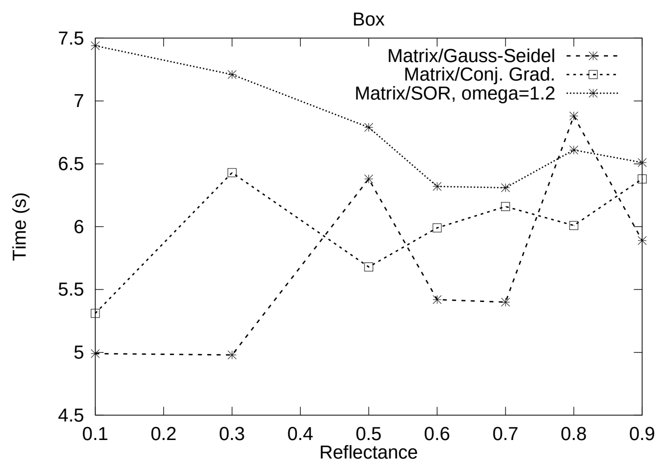


Figure 79: Box experiment, conjugate-gradient and Gauss-Seidel perform equally well.

4.3. Testing Theory

4.3.1. Linear System Solution

Matrix radiosity uses immense amounts of memory, so it is impractical for all but simple scenes, but it is in many ways the easiest algorithm to analyse because it is non-adaptive. Analysing this algorithm can shed light on the other, more practical, techniques.

All the radiosity methods at their core involve solving a system of linear equations. With this in mind, it is interesting to look at how standard techniques for solving linear systems operate in the context of radiosity matrices. **Figure 79** shows the time taken to solve the system for matrix radiosity using the Gauss-Seidel, successive overrelaxation (SOR), and conjugate-gradient solution methods for the box experiment. All methods produce almost identical errors, so there is no need to equalise the times. There is no significant difference between the time taken by Gauss-Seidel and the conjugate gradient methods. For this particular experiment, SOR was slower than Gauss-Seidel, but that is not typically the case.

Figure 80 shows the same kind of graph for the tube experiment, with much more interesting results. Here we see that both Gauss-Seidel and SOR with ω=1.1 become quite slow on scenes with high reflectance (recall that Gauss-Seidel is equivalent to

SOR with ω=1.0). The total solution time, including both form factor computation and solving, among other operations, ranges from about 100 seconds for low reflectance to about 200 seconds for high reflectance. SOR with larger values of ω and conjugate gradients do not slow down as much for high reflectance scenes. Just comparing the SOR curves, it appears empirically that smaller values of ω work best for low reflectance scenes and larger values of ω work best for high reflectance scenes. For comparison, the time for progressive radiosity (Southwell's relaxation method) to simulate the same scene, with the same mesh (but producing a higher error) ranged from 24 seconds for ρ=0.3 to 183 seconds for ρ=0.9, as shown in **Figures 39** and **40**. Thus, solving the more reflective scene took 9 times longer than the less reflective one! For matrix radiosity, the time to compute form factors clearly dominates for low reflectance scenes, but for high reflectance scenes, the time spent by the solver can be half of the total. Progressive radiosity is several times faster than matrix radiosity on scenes with low reflectance, but it becomes as slow as matrix radiosity for high reflectance.

In light of these results we would recommend using the conjugate gradient method for matrix radiosity; the cases in which it makes a considerable difference are perhaps rare, but there is no penalty to using it in other situations. Successive overrelaxation is next best (if ω is chosen carefully), and Gauss-Seidel is the poorest of the three.

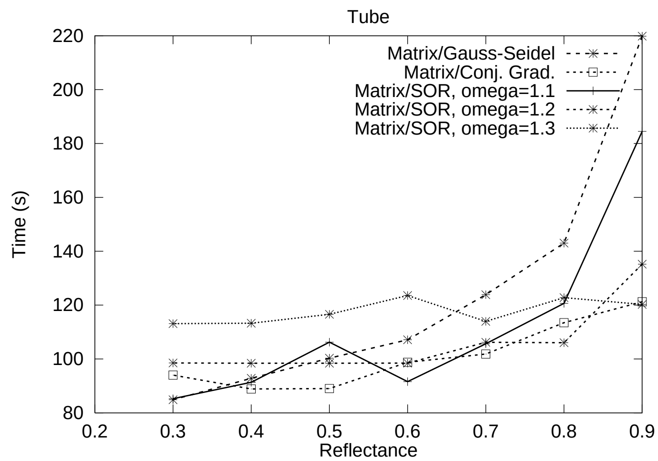


Figure 80: Tube experiment, conjugate-gradient outperforms Gauss-Seidel as reflectance increases.

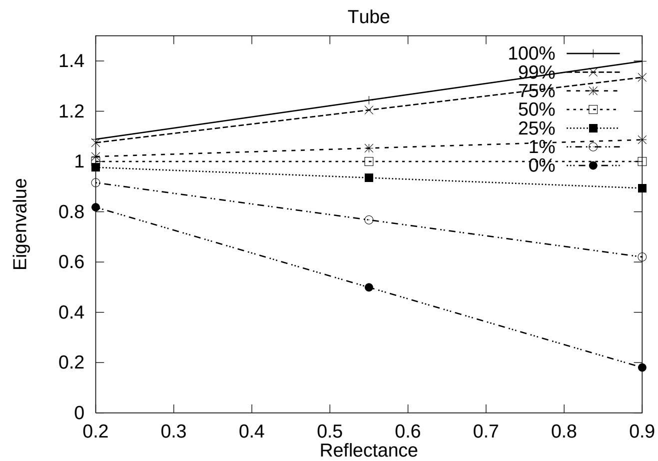


Figure 81: Tube experiment, eigenvalue percentiles of the 260×260 matrix A for the blue channel.

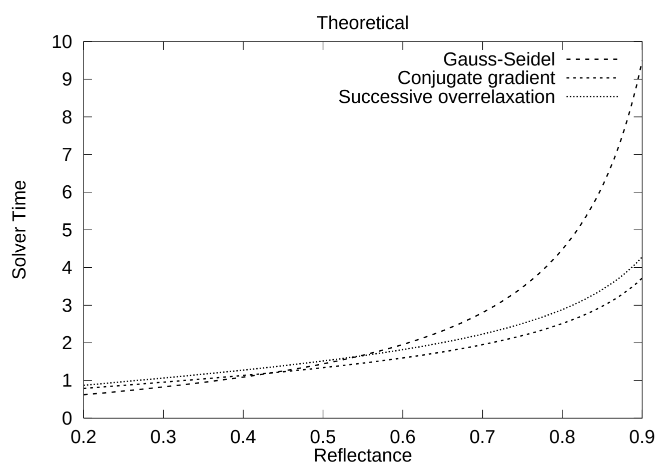


Figure 82: Theoretical performance of the Gauss-Seidel, SOR, and conjugate gradient solution methods.

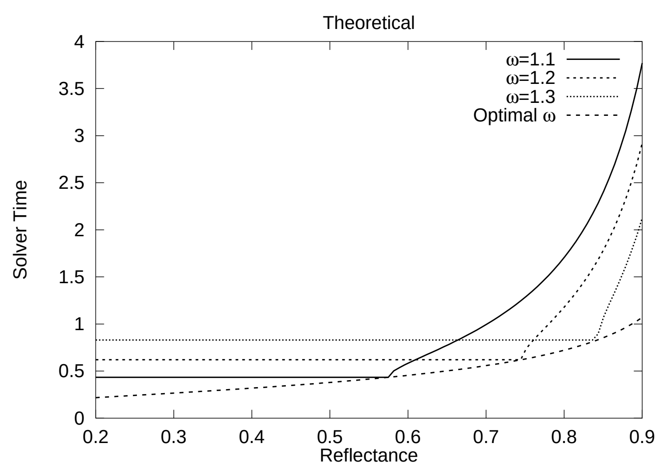


Figure 83: Theoretical performance of successive overrelaxation for various values of ω.

In related work, Baranoski et al. found the Chebyshev iterative solution technique to be even faster than conjugate gradients, and found both of these techniques to be faster than progressive or Gauss-Seidel when reflectances were high [4]. Faster variants of the progressive method that "overshoot" the light have been derived [62]. It may be worthwhile to introduce conjugate gradient-like techniques to progressive or wavelet radiosity, similar to the manner in which successive overrelaxation was applied to progressive radiosity [21].

Eigenvalue Analysis

Variation in solution time as a function of reflectance can be explained in terms of the eigenvalues of the radiosity matrix and their effect on the convergence of system solution methods. Recall that the radiosity equations are written $Ab = e$, and that $A = I - K$ where K is reflectances times form factors: $K_{ij} = \rho_i F_{ij}$. The matrix A is diagonally dominant [13] and positive definite (all eigenvalues real and positive), and the matrix K has real eigenvalues with absolute value less than 1 [27].

The Jacobi iterative solution method has an error of $O(|\lambda_{max}(K)|^i)$ after i iterations [18] and Gauss-Seidel has an error of $O(|\lambda^2{}_{max}(K)|^i)$ [63], where $\lambda_{max}(K)$ is the largest-magnitude eigenvalue of K (here we are assuming that F_{ii}=0, which is true for polygonal scenes). The convergence of successive overrelaxation is highly dependent on the choice of the parameter ω. The optimal value of ω is a function of $\lambda_{max}(K)$, but fortunately we can predict this largest eigenvalue fairly easily for radiosity problems. The optimal value of ω is $\omega_b = 2/((1 + \sqrt{1 - \lambda^2_{max}(K)}))$ [63]. For this value of ω, the error of successive overrelaxation is $O((\omega_b - 1)^i)$. Finally, the conjugate gradient method has an error of $O(((\sqrt{c(A)} - 1)/(\sqrt{c(A)} + 1))^i)$ after i iterations [18], where $c(A) = |\lambda_{max}(A)|/|\lambda_{min}(A)|$ is the condition number of the matrix A.

The number of iterations required to achieve a given error will therefore be $O(-1/(\log\lambda_{max}(K)))$ for Jacobi or Gauss-Seidel iteration, $O(-1/(\log(\omega_b - 1)))$ for successive overrelaxation with $\omega=\omega_b$, and $O(-1/\log((\sqrt{c(A)} - 1)/(\sqrt{c(A)} + 1)))$ for conjugate gradients. Jacobi and Gauss-Seidel have the same asymptotic convergence, but the latter is twice as fast as the former. Empirically, the eigenvalues of the matrix K have absolute value between 0 and ρ, and A's eigenvalues appear to lie between $1 - \rho$ and $1 + \rho / 2$ (the eigenvalue scatterplots of Baranoski et al. show a similar range [4]). In **Figure 81**, for example, we see how the eigenvalues change as a function of reflectance for the tube scene. When ρ_{blue}=0.9, the eigenvalues of A range from λ_{min} = 0.18 to λ_{max} = 1.40, with λ_{median} = 1.0. Note, however, that eigenvalue ranges are only predictable if the form factors are accurate.

For the tube scene, we can therefore predict the number of iterations for Jacobi, Gauss-Seidel, or successive overrelaxation by substituting $\lambda_{max}(K) = \rho$ into the formulas above. The cost of solving will be proportional to the number of iterations. If one does not use the optimal value of ω in SOR, the convergence behaviour is still predictable [63, p. 173]. **Figure 83** shows solver time as a function of ω and ρ, based on the assumption that λ_{max}=ρ. Low values of ω (as in Gauss-Seidel) require many iterations to converge when the reflectance is high. Use of the optimal parameter value, $\omega=\omega_b$, yields good convergence throughout the range. Finally, we can predict the number of iterations for conjugate gradients by substituting $c(A) = (1 + \rho/2)/(1 - \rho)$ into the equations above. Most surfaces found in the real world have a reflectance below 0.9[1], so the condition numbers of radiosity matrices are quite small, relative to those of many matrices encountered in other scientific computing problems. Since the conjugate gradient method converges very quickly for matrices with low condition number, often requiring far fewer than n iterations, the method appears very well suited to radiosity matrices. In our experience, 5 to 10 iterations suffice.

Plots of theoretically predicted solver time based on the formulas above are shown in **Figure 82**. Gauss-Seidel is quite slow for high reflectance scenes, while SOR and conjugate gradient are much less affected. Note that the relative scales on these curves are arbitrary, and depend on the time per iteration for each method. When scaled a bit, the SOR complexity curve becomes nearly identical to the conjugate gradient curve. When this figure is compared to **Figure 80**, we see that the empirical data match the theoretical predictions quite well.

4.3.2. Progressive Convergence

Cohen has observed that the complexity of progressive radiosity, although worst-case $O(k^2)$, is often $O(k)$ in practice [14]. We have found this to be true for some scenes and untrue for others. In **Figure 84**, for instance, the time taken by the progressive solution increases linearly with the initial patches in the scene. For this case we used an unscaled version of the complex scene referred to in Section 3.1: the room was held at a fixed size, and the chairs in the room were packed in more densely as their

1. Very high reflectances are rare because dust and other natural process tend to darken them. Constant cleaning is required to maintain reflectances above 0.9 or so. Personal communication, Greg Ward, 1989.

number increased. **Figure 85** shows the same graph for the usual, scaled complex scene where this is not the case; the curves are approximately quadratic. In this case the chairs were evenly spaced, and the room scaled up to fit them.

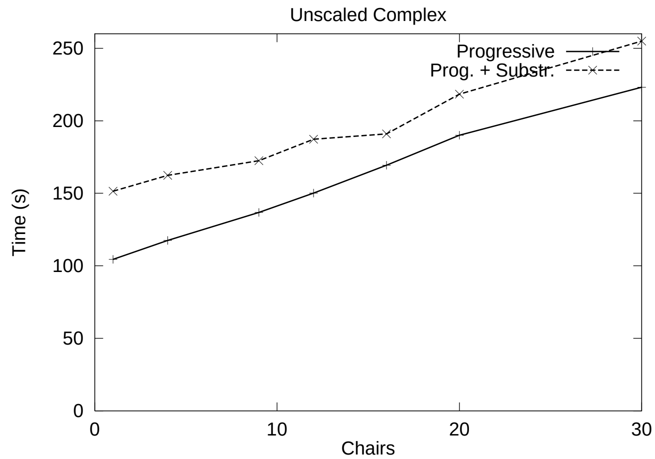


Figure 84: Complex experiment, time to convergence for progressive radiosity is linear for unscaled scene.

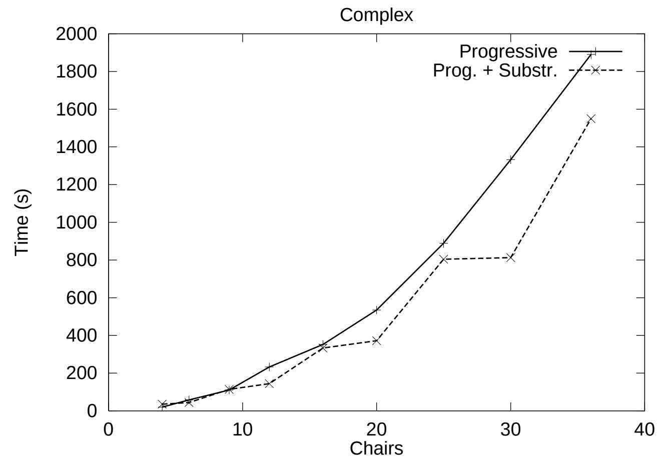


Figure 85: Complex experiment, time to convergence for progressive radiosity is super-linear for scaled scene.

The dominant costs of progressive radiosity are tracing rays to determine visibility and computation of each form factor. Theoretically the asymptotic cost of the former is $O(snv)$ and the cost of the latter is $O(sn)$, where s is the number of shooting steps. **Figure 56** shows that for progressive radiosity, in practice, the time spent tracing each ray is approximately constant ($v = \Theta(1)$), so the total cost of visibility in progressive radiosity is well characterised as $\Theta(sn)$. Therefore we predict the total cost of progressive radiosity to be $\Theta(sn)$. Next, we need to predict s and n as a function of k for our scaled and unscaled scenes.

In the scaled scene, the room grows as the number of chairs increases such that the total wall and floor area is proportional to the number of chairs. Because the chairs remain a fixed size, the total surface area of the chairs also grows linearly with scene complexity. Thus, the total surface area of all objects in the scene is directly proportional to the number of chairs. Initial meshing subdivides to a constant edge length of `edge_len` or `edge_len2` for progressive and substructuring methods, respectively. Therefore, the number of elements, n, created by initial meshing will be proportional to the number of chairs. **Figure 85** shows empirically that substructuring spends approximately the same amount of time on these complex scenes as the vanilla algorithm, suggesting that the fraction of elements that are created adaptively in substructuring is small relative to the total number of patches. In our time estimate we will therefore ignore all but the initial elements. We observe in **Figure 87** that in the scaled scene, the number of shooting steps (to achieve a reasonably low error) is very close to proportional to the number of chairs, so $s = \Theta(k)$. Since the number of polygons, k, and s and n are all proportional to the number of chairs, we would thus predict that the time for progressive radiosity on the scaled scene is $\Theta(k^2)$, and this is empirically confirmed in **Figure 85**.

In the unscaled scene, the room does not scale up, so the total surface area consists of a large constant term for the wall area, plus a small linear term for the chairs. The initial number of elements, n, is likewise linear as a function of k. **Figure 86** shows that the number of shooting steps for the unscaled scene is approximately constant, independent of the number of chairs. This is very different from the scaled scene, where the number of shooting steps is linear in k. In both scenes, we suspect that the lights, walls, and floor shoot, but not the chairs. (For the two scenes we found that the number of shooting steps, s, ranges from 1/5 to 4/7 of the number of patches in the source mesh.) Since the number of elements in the walls and floor grows linearly in the scaled scene, but is constant in the unscaled scene, the number of patches likely to shoot is $\Theta(k)$ and $\Theta(1)$, respectively. We would thus predict a total cost of $\Theta(sn) = \Theta(k)$ for the unscaled scene, and this is also empirically confirmed in **Figure 84**.

In summary, the contrasts between **Figure 84** and **Figure 85** illustrate that the asymptotic time cost is highly dependent on the manner in which a scene scales.

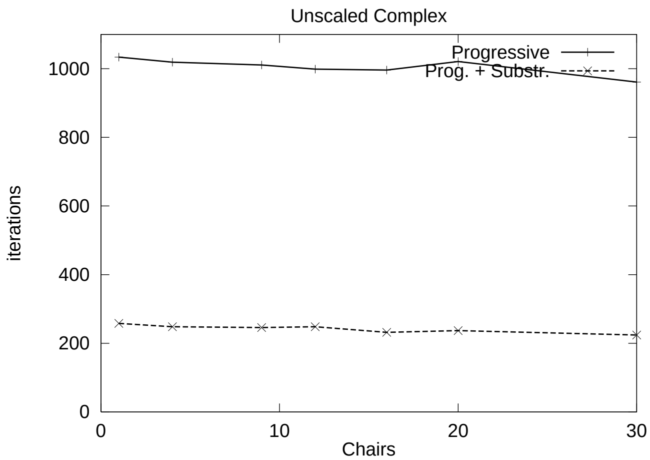


Figure 86: Complex experiment, number of iterations until convergence for progressive radiosity is constant for unscaled scene.

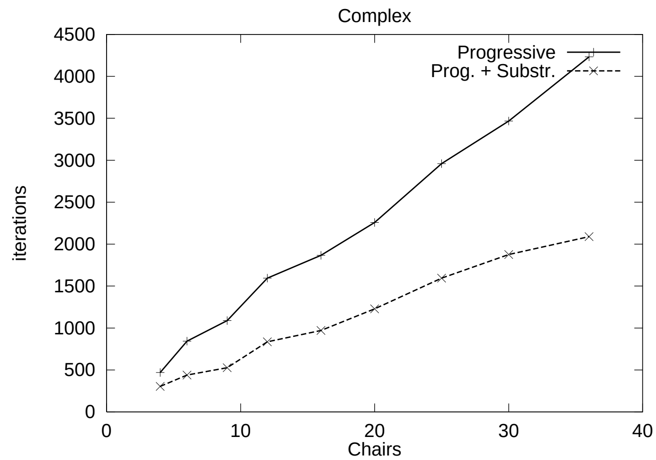


Figure 87: Complex experiment, number of iterations until convergence for progressive radiosity is linear for scaled scene.

4.3.3. Ambient Correction Term

A smaller problem concerns the ambient term that is commonly added into a progressive refinement solution to make the image seem more realistic. Cohen recommended the ambient approximation, and found it to be more accurate early in the solution process, when little light has been shot [12]. We found that instead this term tends to overestimate the radiosity in the scene, as can be seen from **Figure 88**. This charts the solution process for both normal progressive radiosity, and progressive radiosity with the ambient term added; the error of the solution with the ambient term added is always higher than the error of the solution without this term.

The ambient term overestimates the amount of radiosity left in the scene because it assumes all patches are inter-visible. Thus occlusion, and co-planarity of patches belonging to the same base polygon, contribute to the overestimation of the ambient term. The contrasts between **Figure 84** and **Figure 85** illustrate the great effect that the choice of mesh has on speed.

4.3.4. Complexity of Wavelet Radiosity

In this section we look to compare various aspects of the performance of wavelet radiosity in our experiments to complexity estimates in the literature. We also attempt to find new empirical relationships from our results.

One of the main theoretical benefits of higher-order wavelets is that they can fit smooth radiosity functions with a smaller number of basis functions and thus converge faster. We saw this in **Figure 24** for two parallel elements, as in [22].

Figure 89 shows how the number of elements vs. solution time varies for the different bases. We would expect these plots to be linear; we are plotting time against *n*, for a fixed *k*, and theoretically the solution time is $O(k^2v + nv)$ [25]. In fact the curves are slightly sub-linear; we hypothesize that this is due to the effects of visibility, and the `edge_min` limit. Obviously, as the order of the basis increases, the overhead in creating elements and links grows, thus Haar has the steepest slope, and M3 the flattest. The difference between the slopes of the flatlet and multiwavelet bases is minor at order 2, but more significant at order 3. This difference is due to the extra time overhead of the quadrature needed to calculate transfer coefficients for the multiwavelet bases.

In a typical scene many polygons are not inter-visible; indeed we might expect that each polygon only ‘sees’ a roughly constant number of other polygons. It is reasonable to ask whether this occurs in our complex scene, and we can do this by looking at the visual *density* of the scene, which we define as the number of initial links created between the scene polygons divided by the total number of possible links between them. The latter is $k(k-1)$, the former is some fraction of that based on the amount

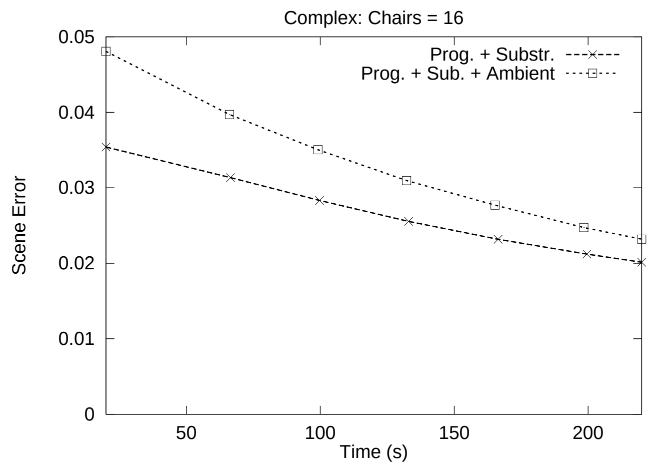


Figure 88: Complex experiment, error vs. time for progressive radiosity with and without the ambient term.

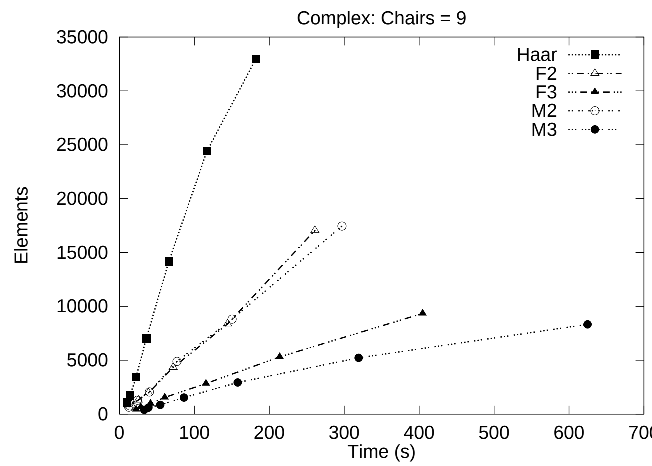


Figure 89: Complex scene, elements at convergence time for a constant number of input polygons.

of occlusion in the scene. Surprisingly, both the unscaled and scaled versions of the complex scene have an almost constant density of around 0.17, independent of scene complexity. (Of course this does not change the fundamentally quadratic nature of the k^2 term.)

The total links in a scene comprise the initial links created between the scene polygons, and the extra links created during the course of the algorithm during refinement. The relationship between the number of initial polygons k and the number of extra links is plotted in **Figure 90**. The higher order bases show a clearly linear relationship between the two. The relationship is less clear for the Haar basis, but it appears sub-linear. We suspect that at around 400 polygons, some of the Haar basis elements become small enough that the subdivision limit `edge_min` prevents further subdivision, which would account for the sub-linear nature of the curve. The same relationship for initial links is shown in **Figure 91** for clarity: this is of course just k^2density.

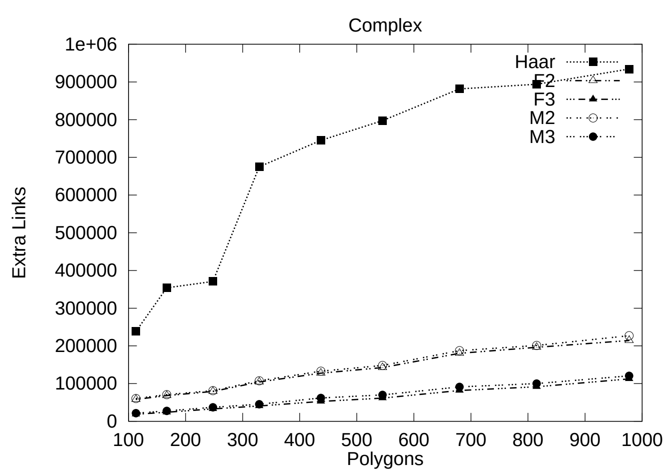


Figure 90: Complex experiment, number of links created after initial linking phase vs. number of polygons (k).

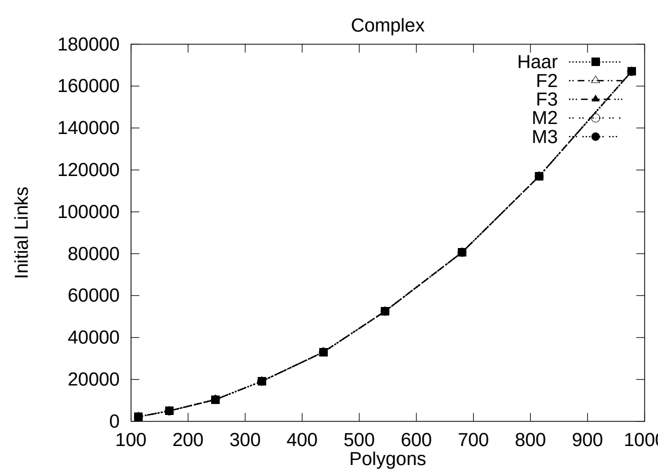


Figure 91: Complex experiment, number of links created in the initial linking phase vs. number of polygons (k). (Note that the five curves lie on top of each other.)

From these graphs we can deduce that the overall number of links for higher-order wavelets (and probably Haar as well) is k^2density + βk. The Haar basis creates many more links than do the second-order bases, which in turn create more links than the third-order bases. For example, in the complex scene with 36 chairs, the number of initial links as a percentage of the total links at convergence was 15% for Haar, 43% for F2 and M2, and 59% for F3 and M3. So for the Haar basis, the number of links created is dominated by the second, linear term of this relationship, and as the basis order increases, the number of links starts to be dominated by the first, quadratic term. This effect at least partly explains the orders of the memory and time complexity curves for the different wavelet bases seen in Section 4.1.4.

Figure 92 shows the wavelet order's effect on mesh density for given error. A Galerkin method using a basis of order *M* - 1 should have an error of $O(h^M)$, where *h* is the sidelength of an element [42]. If this formula is to hold, we would expect the log plots shown to be linear, with a slope of -*M*/2. However, rather than slopes of -1/2, -1 and -3/2, fitting lines to the plots gives slopes of -0.80, -0.63, and -0.60 for the Haar, M2 and M3 bases respectively[1]. The theoretical predictions assume no occlusion, and evenly divided meshes. We believe that the disagreement between our results and theory is due to the poor visibility handling and inadequate meshing mentioned in previous sections. Thus, as we saw in Section 4.1.4, the Haar method is the most efficient, whereas the other bases have roughly equal efficiency.

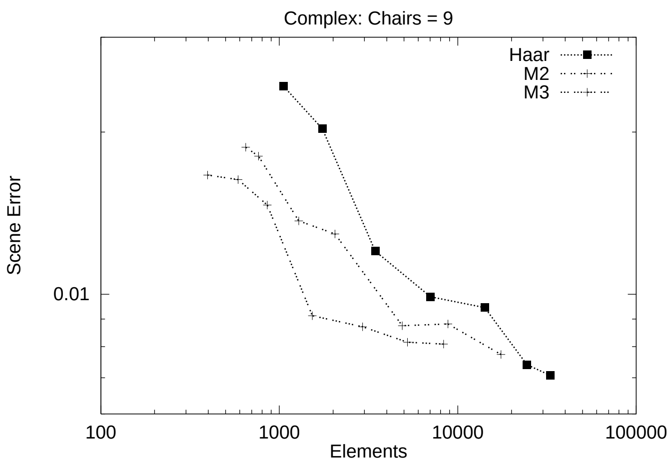


Figure 92: Complex experiment, mesh density at convergence time vs. error for different wavelet bases.

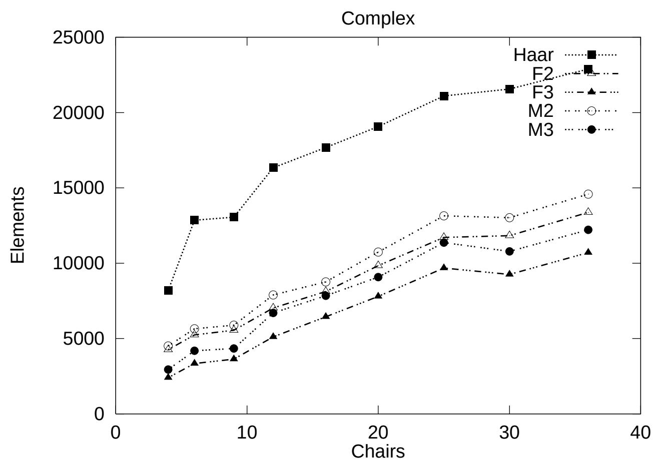


Figure 93: Complex experiment, elements at convergence time for wavelet radiosity.

Finally, **Figure 93** shows how the number of elements required by a wavelet radiosity solution varies with scene complexity. For all the bases, the curves are close to linear from 12 to 25 chairs; if it were not for the other data points, we might conclude that time is linear in scene complexity, i.e., time = $O(k)$.

Close examination of the Haar method's meshes explains the large jumps in element number early in its curve. **Figure 94** shows these meshes for the rear wall of the two least complex scenes. Suppose the oracle recommends refinement for a certain wall element, whose size is a fixed fraction of the room's dimensions. The room grows as *k* increases, but `edge_min`, the refinement termination criterion, is fixed. In some cases such an element could be larger than `edge_min` for a large scene, but smaller than `edge_min` for a small scene, permitting refinement in the large scene but preventing it in the small scene.

In a scene with 4 chairs, fine subdivision on the walls near the lights is prevented, while for scenes of 6 or more chairs, this refinement takes place. Similarly, in scenes of 9 or fewer chairs, the shadows under the chairs are refined less than in more complex scenes, because of the `edge_min` constraint. These two observed effects lead to the large leap in element numbers in the Haar curve at 6 and 12 chairs. We suspect that similar effects are responsible for the downward dip in all the curves at 30 and 36 chairs.

4.4. Hardware Used

The experiments were carried out on a range of SGI workstations. In particular, the experiments were distributed as follows:

1. The 70% confidence interval for these fits is around 0.05 for all bases.

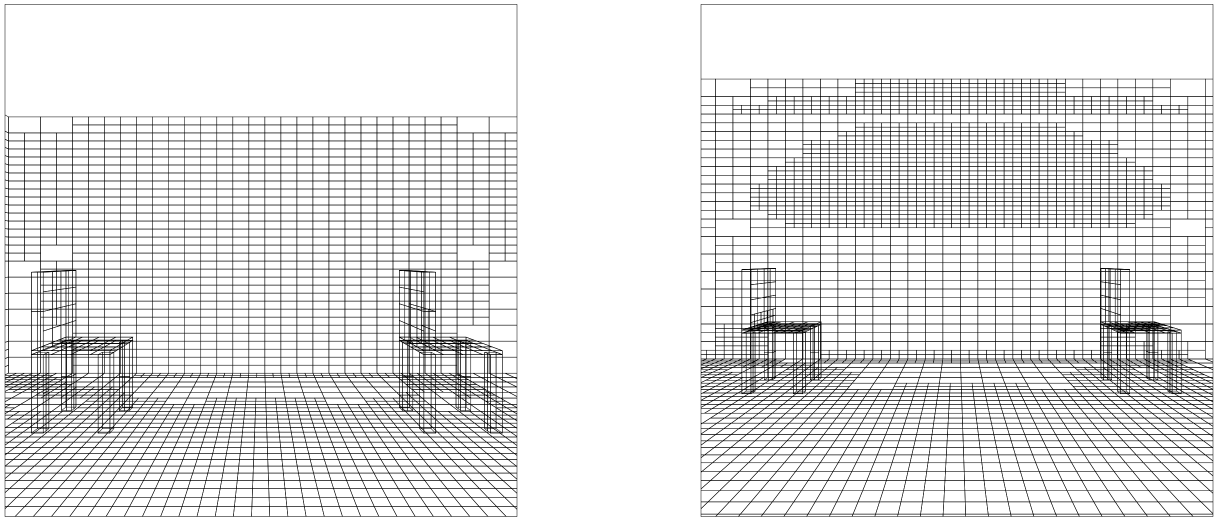


Figure 94: Meshing on the rear wall of the complex scene using wavelet radiosity with the Haar basis: 4 chairs (left), and 6 chairs (right).

- Parallel, Perpendicular, Blocker

  SGI Indy, 150MHz R4400, 32 Mbytes RAM, 16K/16K/1MB instruction/data/L2 cache.
- Box, Tube

  SGI Indigo 2, 250MHz R4400, 128 Mbytes RAM, 16K/16K/2MB instruction/data/L2 cache.
- Complex, Lights

  SGI Indigo 2, 195MHz R10000, 128 Mbytes RAM, 32K/32K/1Mb instruction/data/L2 cache.

5. Conclusions

We end this report by presenting our major conclusions, and then discussing other aspects of our results. We finish with proposals for future work. It should be noted that our conclusions are based on our experimental scenes, and the three basic radiosity methods as we have implemented them. While we have attempted to devise the tests, to choose algorithms, and to implement them in an unbiased way, one should use caution in generalizing our conclusions. We would expect the use of clustering and other advanced techniques to alter these conclusions.

5.1. Primary Conclusions

Our main conclusions are:

- **For simple scenes, substructuring is usually best.** For the simplest scenes we tested, substructuring mostly outperformed the other radiosity methods: It was fast, had low error, and resolved shadows well.
- **For moderate complexity scenes, wavelet radiosity using the Haar basis is often best.** For our 'complex' and lights scenes, Haar did the best or near best of the methods we tested (**Figure 45**). Also, for the complex scene, its time cost appeared to grow linearly, while substructuring's cost appeared to be quadratic (**Figure**s **53** and **54**). For a given amount of time, Haar often gave the most accurate results. However...
- **Wavelet methods use an immense amount of memory, so for complex scenes they become impractical.** Empirically, the memory requirements for wavelet methods for this scene size grow quadratically with scene complexity in scaled scenes (**Figures 51** and **52**). Also, the higher-order methods require many times more memory than Haar. Recall that wavelet methods of order M use $O(M^2)$ memory per element and $O(M^4)$ memory per link. For our complex scene with thirty-six chairs, for example, the second-order methods (F2 and M2) required three times as much memory as Haar, and the third-order methods (F3 and M3) took twelve times as much as Haar. Haar itself took fifty-five times as much memory as substructuring. (See table below.) Furthermore, in contrast to time, memory is a finite resource. When the scene is too complex for wavelet methods, substructuring becomes a better choice.

	Substructuring	**Haar**	**F2 & M2**	**F3 & M3**	**Matrix**
Memory for complex, 36 chairs	0.8 MB	44 MB	150 MB	550 MB	1000 MB (estimate)

- **Higher-order wavelet methods as implemented look poor on scenes with occlusion.** With either the fractional visibility method or the visibility-in-quadrature method, shadows appear extremely blocky due to discontinuities between elements (**Figure 48**). The former approach appears to suffer due to piecewise-constant approximation of visibility, and the latter samples visibility on too coarse a grid. On unoccluded scenes such as the parallel and perpendicular scenes, however, higher-order methods often gave the most accurate solutions (**Figures 26** and **29**). Higher-order wavelets currently handle visibility poorly, preventing them from achieving their potential accuracy. Further work is needed on this.
- **For hard shadows, substructuring is recommended.** Objectively, on the blocker scene we observed that substructuring is the most accurate method when a shadow contains a umbra (**Figure 32**). Because wavelet methods typically refine less than substructuring near shadow boundaries, their shadows appear blurrier (**Figure 47**). This blurring problem is most evident on hard shadows. Although our experiments indicate hierarchical radiosity can be better for very soft shadows, our visual system is subjectively more sensitive to errors in hard shadows. Since our objective error measure does not take this into account, sometimes substructuring produces more realistic looking scenes even though its measured error is the same as that for hierarchical radiosity.
- **Progressive radiosity without substructuring and matrix radiosity are not recommended.** Except in certain rare circumstances (tube scene, **Figures 39** and **41**), they are inferior to substructuring and hierarchical radiosity. Matrix radiosity is even more memory intensive than higher-order wavelets. We estimate that to render the complex scene with thirty-six

chairs, and the same mesh as was used for progressive radiosity, matrix radiosity would take over a gigabyte of memory. This is twice that used by the F3 and M3 bases, in spite of using a coarser mesh. Progressive radiosity was typically poorer than progressive with substructuring, in the sense that it had a higher error for a given amount of time (**Figure 45**).

In an attempt to summarize the relative merits of the algorithms we implemented, below is a graph suggesting the overall subjective ‘goodness’ of the three leading methods based on our experiments. (The relative positions and heights of these curves are only meant to be suggestive.) In general, for the simplest scenes, substructuring is typically best, but for moderately complex scenes, Haar is best. Higher order wavelets look unacceptable for scenes with occlusion, so they are rarely the best choice. The extreme memory requirements of the wavelet methods cause them to page for problems past a certain size. The higher-order methods thrash sooner, and Haar later. Beyond that size, these methods become impractically slow, and substructuring is recommended. Higher order wavelets are only recommended for scenes with little occlusion. We rate matrix radiosity (not graphed) as the poorest of the methods.

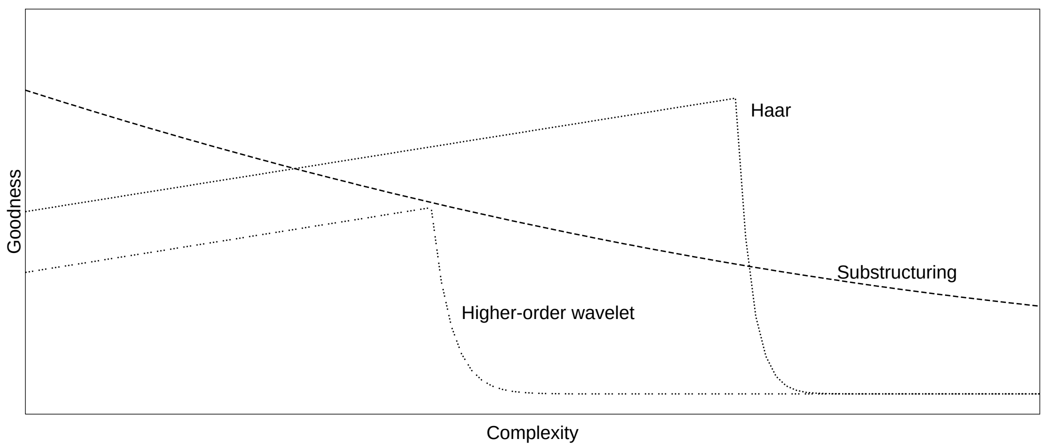


Figure 95: Subjective, overall “goodness” as a function of scene complexity.

We expect that improvements to the wavelet methods will at some stage change our conclusions about their current utility. Certainly the wavelet methods have more long-term promise than progressive radiosity with substructuring.

5.2. Empirical Complexity Results

We observed strong effects of complexity on time and memory use in our experiments. We summarise them here and present some explanations.

5.2.1. Progressive Methods

Running both progressive methods on our complex scene, we found that:

- **Convergence time is $\Theta(k^2)$ for a scaled scene, and $\Theta(k)$ for an unscaled scene** (**Figure 84**, **Figure 85**).

  Figure 86 and **Figure 87** show that the number of iterations (shooting steps) required for the progressive methods to converge is close to constant for the unscaled scene, and increase linearly with complexity for the scaled scene. Each shooting step is $O(nv)$, where n is the number of patches or elements in the receiving mesh. Thus, if we assume that v is $O(1)$ on average[1] we would expect to see the $\Theta(k^2)$ and $\Theta(k)$ time costs above.

1. This is often the case when testing occlusion in medium or low complexity scenes, where early-ray termination is used.

If we assume that almost all radiosity shot comes from the light source, walls, and floor, and very little from the chairs themselves in the complex scene, we can explain the behaviour of the iterations required for convergence as follows. For the unscaled scene, the room stays the same size as more chairs are added, and thus the number of patches in the mesh covering the walls and floor remains constant. If these are the major sources of radiosity being shot, then the number of shooting steps will also be constant. In contrast, in the scaled scene the surface area of the walls and floor scales with the number of chairs, and thus the number of patches in the mesh covering them will increase linearly with complexity, and so will the number of shooting steps.

- **Memory use for progressive radiosity is $\Theta(k)$** with a low constant (**Figure 52**). Since progressive radiosity stores neither a matrix or links, but allocates most of its memory for the mesh of elements, its memory use is $\Theta(n)$. As shown above, $n = \Theta(k)$ for both scaled and unscaled complex scenes, so memory use of both types of progressive radiosity is expected to be $\Theta(k)$.

  The above conclusions hold for our complex scene with the meshing and termination criteria we chose, but might not generalize to other scenes, where detailed mesh refinement might make the number of elements grow faster than $\Theta(k)$.

5.2.2. Wavelet Methods

For all bases, we observed the following behaviour as the number of polygons, k, was varied in the complex scene:

- **#elements = $\Theta(k)$** (**Figure 93**). The figure shows that n grows approximately linearly for Haar and the higher order methods, although for some scenes the number of elements was depressed due to the effect of the `edge_min` parameter. This was discussed in Section 4.3.4.
- **density = $\Theta(1)$**. As discussed in section Section 4.3.4, the fraction of unoccluded links was constant in both the scaled and unscaled scenes.
- **#links = k^2density + βk** (**Figure 90**). The number of initial links is density.k^2 and the number of extra links created adaptively is $\Theta(k)$, as discussed in Section 4.3.4. Thus, the number of links is $O(k^2)$ asymptotically, but we observe that for the scenes tested, the extra links dominated for Haar and the initial links dominated for the higher order methods.
- **memory = $\Theta(k^p)$ for $p \in [1,2]$ depending on basis order** (**Figure 51**). The predominant use for memory in wavelet methods is to store links. Their number grows as k^2density + βk; the two terms correspond to initial links and extra links. For the Haar basis, the extra links dominate, and we observe $\Theta(k)$ memory use. For higher order methods, the initial links dominate, and we observe faster growth:$\Theta(k^{1.28})$ for second order methods and $\Theta(k^{1.62})$ for third order methods.
- **time = $\Theta(k^p)$ for $p \in [1,2]$ depending on basis order (Figure 46**, **Figure 50**). Theoretically, we would predict a time cost of $O(links \times v + n)$. When visibility has constant cost or negligible cost, and the link term dominates the element term, we would expect to observe similar behaviour for time as for memory. This is indeed the case: we see $\Theta(k)$ time cost for Haar, $\Theta(k^1)$ for the second order methods, and $\Theta(k^{1.8})$ for third order methods.

As the meshing parameters were varied for the complex scene, we observed:

- **#links = Θ(1/epsilon) (Figure 76)**. As shown in Section 4.2.6, for small enough epsilon, the number of links generated by the wavelet methods is proportional to 1/epsilon. As epsilon is increased, and the number of links decreases to the minimum, k^2density, the number of links approaches a constant.
- **time, memory= Θ(1/epsilon) (Figure 72, Figure 74)**. We also observed that the time cost was linear in 1/epsilon, which suggests that it is more dependent on the number of links in the scene than the number of elements. Similarly with memory.

5.3. Problems Observed and Proposed Solutions

Now we identify specific problems with the radiosity algorithms, and suggest some solutions.

5.3.1. Progressive Methods

Ambient Term. When the ambient term is added to a progressive solution, its accuracy decreases (**Figure 88**). Experiments with adding fractional parts of this term may produce better results.

Meshing. Light sources must be finely meshed in order to produce soft shadows. However, in doing so we currently also force other, non-shadow-casting surfaces in the scene to be more finely meshed, and thus drastically increase running time. It would be better to mesh the light sources to a greater extent than other objects in the scene, and high-reflectance objects more heavily than low-reflectance ones. This is where wavelet radiosity has an advantage, with its ability to effectively subdivide the source mesh based on the current illumination in the scene; we saw that advantage starting to manifest in the last two scenes of the complex experiment.

Overrefinement. Overrefinement is a problem for high reflectance scenes and lights that touch another surface at a reflex corner. This slows substructuring radiosity down, and uses more memory than necessary (Figures **40** and **63**). The root cause is premature refinement. The problem could perhaps be solved using a multigrid approach. If the scene is first illuminated with a coarse mesh and with substructuring turned off, then, when the substructuring method is run using that mesh as a starting point, patches that are shot early on will not overrefine areas where the final radiosity gradient is smooth.

5.3.2. Wavelet Methods In General

Singularities. A significant source of error near kernel singularities is accumulated projection error (**Figure 69**). One approach to this problem is to force links to the leaves of the receiving patch near the singularity. Then all links from the source patch will go directly to these leaf elements, and there will be no summing of projections at different scales. This approach has the drawback that it will increase the total number of links needed for the transfer, and thus solution time. It may also be that for the piece-wise constant basis functions, the area-to-point form factor can be employed to replace the links from the source to multiple levels of the receiver's hierarchy with a single link, which would also decrease the accumulated error.

Link formation. When surfaces are close, we found with hierarchical radiosity that the number of (point-to-point) links needed to accurately represent the radiosity transfer was large enough to make it uncompetitive with simpler methods that employed the area-to-point form factor estimate. This suggests that more intelligent oracles employing asymmetrical link formation (where the level of links at the source is less than at the receiver) could improve performance. Oracles that do a posteriori error estimation could also help [8].

5.3.3. Higher-Order Wavelets

Higher order wavelets are considerably more complex to implement than standard hierarchical radiosity, which begs the question: Is the cost of implementing them justified by better performance? For our implementation and the experiments presented in this report, the answer must be no, especially considering that our scenes were only of moderate complexity. In particular, we found that:

- The memory use of higher-order wavelets is prohibitive.
- For non-trivial scenes (i.e., our complex scene), higher-order wavelets are always less accurate for a given time than Haar.
- Shadows appear unacceptably blocky.

We suspect three causes: the incomplete utilisation of high order methods, the lack of post process smoothing, and storage of superfluous coefficients. The potential of higher-order wavelets is not being realised because the poor approximation of visibility handicaps their accuracy. Coefficients are wasted in situations where the radiosity function across an element is smooth enough that it could be equally well approximated by a lower order basis.

At a minimum, the following points must be addressed before higher-order wavelet radiosity is viable:

Higher-order visibility. Better methods than either fractional visibility or visibility-in-quadrature must be used. The visibility approximation employed should be at least C^0 between adjacent elements, instead of being piecewise-constant, as in the fractional visibility method, or piecewise-linear or quadratic, as in visibility-in-quadrature. One approach would be to find a higher-order-than-constant estimate of the variation of visibility across the receiving patch, and then use this to weight the transferred radiosity after quadrature has taken place, in a more sophisticated version of fractional visibility. Slusallek et al. have produced visually pleasing results using a similar approach, in which the visibility is calculated at a higher resolution than the elements; this is essentially a version of Zatz's shadow masks [51, 64]. While this solves the continuity problems with wavelet shadows, the separation of visibility and geometric terms in the integration for transfer coefficients is an approximation which must occasionally lead to errors. (Note that this is also true of fractional visibility.) An alternative would be to use visibility-in-quadrature (which does not make this approximation) with some form of discontinuity meshing; this would avoid visibility discontinuities falling across the support of the wavelets, and thus greatly reduce ringing behaviour and inter-element discontinuities. It is an open question as to which approach will yield the best combination of accuracy and speed.

Continuity. For display, one of the major problems with the higher order wavelets used in this report is the lack of any continuity constraints on neighbouring elements. Even in the absence of occlusion, this leads to shading discontinuities, which the eye is adept at picking up. Because these discontinuities are exaggerated by our visual system, our global error measure under-represented the problem.

To address this we either need to find different, non-tree wavelet bases, or to find a way of enforcing continuity between the elements as a post-process. (Recall that we post-processed Haar's solution to make it C^0 . For the former approach, bases similar to the Hermite family of splines look promising, although the greater amount of subdivision they require, and the greater expense of reconstructing the radiosity function from their basis coefficients, may well count against them. One avenue for the latter approach is to use bases similar to the multiwavelets, but re-parameterised so that the basis coefficients correspond directly to radiosity samples at the edges of the element. This has the advantage that we can then enforce continuity between neighbouring elements by averaging their radiosity values at shared vertices. Also, for second-order multiwavelets the mesh can be displayed directly from the basis coefficients, without an intermediate reconstruction step.

Memory Use. For the higher-order methods to be practical, their memory use must be reduced. Promising results have been produced for hierarchical radiosity with clustering, which removes the k^2 dependence of the number of links, and thus should similarly reduce memory costs [52, 50]. Hopefully these results will be extended to wavelet radiosity with higher-order bases. Another possibility is generalised multiwavelet radiosity, in which the order of the bases of different elements changes to match the variation in radiosity across them, rather than being the same for all elements. For elements where the radiosity function is constant, the Haar basis could be used, and where the radiosity function has greater variation, an appropriate higher-order multiwavelet could be used[1]. This would avoid storing extra basis or transfer coefficients when they are unneeded.

For applications where view-independent illumination is not needed, it is highly likely that importance-based radiosity will help reduce memory costs to a reasonable level.

Ease of Implementation. In order to promote the wider use of wavelet radiosity, the 'entry cost' for implementing it has to be lowered. There are currently no clear, straightforward guides to implementing wavelet radiosity, as exist for the progressive method, and few freely-available wavelet radiosity implementations. (We are aware of only one other than our own, which is described in [3].) We hope to alleviate this both by our description of our implementation in this report and a later report, and by making the implementation itself available for others to use. We see no reason why higher-order wavelet methods shouldn't be as easy to implement as progressive radiosity with substructuring, so long as there is a clear example to follow.

1. It can be shown that calculating the link coefficients for transfers between multiwavelets of different orders requires no more than two matrix multiplications—the same as for transfers between bases of the same order.

5.4. Miscellaneous

5.4.1. Fast Visibility for Wavelet Methods

We found that progressive radiosity with substructuring was the most efficient method in terms of rays cast per time, followed by progressive radiosity and then hierarchical radiosity. This was in spite of the 'triage' optimisation used for hierarchical radiosity, where totally occluded and unoccluded links did not have their visibility recalculated when they were refined. The large number of rays the method casts suggests that it would be profitable to investigate alternative visibility methods. A truly multi-resolution method would be preferable, because different links can cover widely variable parts of scene space. Possibilities include storing objects in a multi-resolution format [26], and testing against an appropriate level of complexity depending on some measure of the cross-sectional area a link covers, or using a multi-resolution volumetric data-structure to represent occlusion, and traversing it at the appropriate level of resolution.

5.4.2. High Reflectance Scenes

Of all our experiments, the tube scene gave the most anomalous results. High reflectance coupled with the large number of abutting patches in the scene acted to retard the adaptive methods so that they performed worse than both matrix and progressive radiosity, which was never the case for other scenes. Substructuring deteriorated in terms of convergence time, and wavelet radiosity in terms of error at convergence. Given these problems, it might be instructive to look into ways for the adaptive methods to detect situations like these, and either revert to non-adaptive behaviour, or address their problems in some other manner. It is an open question how common such situations are, but we suspect that similar effects may be seen in white hallways in architectural scenes.

The tube and box experiments in particular highlight how touching surfaces are a problem for wavelet methods; many links are created in these cases, slowing them down markedly. Their approximations at corners are poor, and often get worse when refined. The tube scene in particular, with its high radiosity gradient down the length of the tube, would have suited an anisotropic mesh, where unequal subdivision in u and v is possible; this would have reduced the number of links and/or elements required. For the wavelet methods, the quadtree data-structure could be replaced by a deeper k-d tree for rectangular elements, as these use easily-separable tensor bases.

5.4.3. Performance in Complex Scenes

The hierarchical and substructuring methods were equal in performance for the complex scenes until the number of chairs passed thirty. We hypothesize that this was due to the walls of the room in the complex scenes scaling up as the number of chairs increased. It would appear that for hierarchical radiosity to outperform substructuring, the scene must be dominated by large, smooth surfaces. For high-density scenes, consisting of many objects with small surface areas in close proximity, we suspect that substructuring radiosity would do much better than hierarchical radiosity, at least in the absence of clustering. In opposite situations, it is hierarchical radiosity that does well.

5.4.4. Setting Parameters

It is easier to control progressive radiosity than wavelet radiosity. The algorithms are controlled by a number of parameters governing mesh density, and termination. For progressive radiosity, convergence time is reasonably stable as geometry changes, and due to the smooth decrease in scene error over time, changes in the termination ratio produce predictable changes in scene quality. In contrast, wavelet radiosity's time to convergence can be quite variable as scene geometry changes, and the decrease in error over time during a solution run is much less smooth. We have also found that varying epsilon does not produce easily-predictable changes in the error of the resulting solution.

In substructuring radiosity, epsilon, which controls sensitivity to gradients, and the edge-length parameter, which controls how fine the source mesh is, are interdependent. If we set epsilon low, but use a coarse shooting mesh, we get a small number of superimposed sharp shadows cast by point light sources. On the other hand, if epsilon is high, and there is a fine shooting mesh, light sources with a small area don't cast sufficiently sharp shadows. Coupling these parameters in some manner could make the algorithm easier to use.

5.4.5. Matrix Radiosity

As expected, matrix radiosity proved practical only for simple scenes. For even moderately complex scenes its memory use was too demanding, and we found we could not use it on our complex scenes with the available hardware. In our experience the majority of the time taken in a matrix solution is due to form-factor and visibility calculation; thus the sparse matrix representation did not help as much as might otherwise be expected.

We presented new theoretical predictions for the speed of linear system solution techniques for radiosity problems. We conjecture that the eigenvalues of radiosity matrices lie between $1 - \rho$ and $1 + \rho / 2$, where ρ is the maximum reflectance in a scene. Both successive relaxation and conjugate gradients were found to converge quickly for any reflectance, while Gauss-Seidel and progressive methods (Southwell's relaxation method) converge very slowly for high reflectance scenes. Because of the low condition number of radiosity matrices, the conjugate gradient method appears very well suited to solving matrix radiosity problems. Further work is needed, however, to explain and analyse the success of the conjugate gradient method on non-symmetric matrices, as discussed in Section 2.1.1. The conjugate gradient method may be a superior alternative to Jacobi iteration for wavelet radiosity methods, although it is not immediately obvious how it would be applied.

Although matrix radiosity is generally dismissed as impractical because of its $O(n^2)$ memory consumption, a simple variation on the technique could reduce its memory consumption to $O(n)$, allowing it to be run on scenes of much higher complexity. One need not store the entire matrix of form factors, but rather one could recompute rows of the matrix only as needed. This is sufficient, since the basic operation of most iterative solvers is the multiplication of a matrix with a vector. When coupled with a fast linear system solver, such as conjugate gradients, the number of times that each form factor would be recomputed could probably be kept to a small constant.

5.5. Future Work on Radiosity Comparisons

Now we have our testbed in place, and have completed our initial investigation of radiosity methods, there are a number of avenues for future work. These include:

- Investigating performance on a range of real-world scenes, from architectural models to CGI scenes.
- Comparing the radiosity approaches investigated here to Ward's RADIANCE system [61].
- Producing results for the standard library of scene files due to Rushmeier et. al. [37].
- Evaluating the effect a final-gather pass has on the accuracy of the simulation.
- Considering view-dependent methods, and using image-based error metrics.
- Considering perceptually-based error measures.

To assist others in the research community with the investigation of these areas, and for evaluation of our experiments, we have placed our radiosity renderer and experimental system on the internet at the following URL: `http://www.cs.cmu.edu/~radiosity/`. Comments or suggestions can be sent to the authors at `{ajw|ph}@cs.cmu.edu`.

References

[1] Bradley Alpert. Sparse representation of smooth linear operators. Technical Report DCS/RR-814, CS Dept, Yale U, Aug 1990.

[2] Ian Ashdown. *Radiosity: A Programmer's Perspective*. John Wiley & Sons, New York, 1994.

[3] Philippe Bakaert and Yves D. Willems. Hirad: A hierarchical higher order radiosity code. In *Proc. 12th Spring Conference on Computer Graphics, Budmerice, Slovakia*, pages 213–228, June 1996.

[4] Gladimir V. Baranoski, Randall Bramley, and Peter Shirley. Iterative methods for fast radiosity solutions. Technical Report TR 429, CS Dept, Indiana University, Bloomington, IN, April 1995. http://swarm.cs.wustl.edu/~hart/mirror/indiana-biblo.html.

[5] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[6] Daniel R. Baum, Stephen Mann, Kevin P. Smith, and James M. Winget. Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):51–60, July 1991.

[7] Daniel R. Baum, Holly E. Rushmeier, and James M. Winget. Improving radiosity solutions through the use of analytically determined form-factors. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 325–334, July 1989.

[8] Philippe Bekaert and Yves Willems. Error Control for Radiosity. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, pages 153–164, New York, NY, 1996. Springer-Verlag/Wien.

[9] Per H. Christensen, Eric J. Stollnitz, David H. Salesin, and Tony D. DeRose. Wavelet radiance. In *Fifth Eurographics Workshop on Rendering*, pages 287–302, Darmstadt, Germany, June 1994. http://www.cs.washington.edu/research/projects/grail2/www/pub/abstracts.html#WaveletRadiance.

[10] John G. Cleary and Geoff Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2):65–83, July 1988.

[11] Michael Cohen, Donald P. Greenberg, Dave S. Immel, and Philip J. Brock. An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics and Applications*, 6(3):26–35, March 1986.

[12] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):75–84, Aug. 1988.

[13] Michael F. Cohen and Donald P. Greenberg. The hemi-cube: A radiosity solution for complex environments. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):31–40, July 1985.

[14] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press, Boston, 1993.

[15] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, pages 16–26, April 1986.

[16] Reid Gershbein. Integration Methods for Galerkin Radiosity Couplings. In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering) in Dublin, Ireland, June 12-14, 1995)*, pages 264–273, New

York, NY, 1995. Springer-Verlag. http://www-graphics.stanford.edu/~rsg/Quadrature/index.html.

[17] Reid Gershbein, Peter Schröder, and Pat Hanrahan. Textures and radiosity: Controlling emission and reflection with texture maps. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 51–58. ACM SIGGRAPH, ACM Press, July 1994. http://www-graphics.stanford.edu/papers/texture.

[18] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 1989.

[19] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison Wesley Publishing, Menlo Park, CA, 1992.

[20] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):213–222, July 1984.

[21] Steven Gortler, Michael F. Cohen, and Philipp Slusallek. Radiosity and relaxation methods: Progressive refinement is southwell relaxation. Technical Report CS-TR-408-93, Computer Science Dept., Princeton University, Feb. 1993. http://ncstrl.cs.princeton.edu/Dienst/UI/2.0/Describe/PRINCETONCS/TR-408-93.

[22] Steven J. Gortler, Peter Schröder, Michael F. Cohen, and Pat Hanrahan. Wavelet radiosity. *Computer Graphics (SIGGRAPH '93 Proceedings)*, August 1993. http://www-graphics.stanford.edu/papers/wavrad.

[23] Eric A. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 7(11):3–5, November 1987.

[24] Roy Hall. *Illumination and Color in Computer Generated Imagery*. Springer Verlag, New York, 1989.

[25] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):197–206, July 1991. http://www-graphics.stanford.edu/papers/rad.

[26] Paul Heckbert and Michael Garland. Multiresolution modeling for fast rendering. In *Proceedings of Graphics Interface '94*, pages 43–50, Banff, Alberta, Canada, May 1994. Canadian Information Processing Society. http://www.cs.cmu.edu/~ph/multirend.ps.Z.

[27] Paul S. Heckbert. *Simulating Global Illumination Using Adaptive Meshing*. PhD thesis, CS Division, UC Berkeley, June 1991. http://cs-tr.cs.berkeley.edu/TR/UCB:CSD-91-636.

[28] Paul S. Heckbert. Discontinuity meshing for radiosity. In *Third Eurographics Workshop on Rendering*, pages 203–216, Bristol, UK, May 1992. http://www.cs.cmu.edu/~ph/discon.ps.gz.

[29] Paul S. Heckbert. Radiosity in flatland. *Computer Graphics Forum*, 11(3):181–192, 464, Sept. 1992. http://www.cs.cmu.edu/~ph/flat.ps.gz.

[30] Lightscape Technologies Inc. http://www.lightscape.com.

[31] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. *Computer Graphics (SIGGRAPH '93 Proceedings)*, August 1993. http://www.cs.huji.ac.il/~danix/papers/s93paper.ps.gz.

[32] Bruce Naylor and William Thibault. Application of BSP trees to ray-tracing and CGS evaluation. Technical Report GIT-ICS 86/03, Georgia Institute of Tech., School of Information and Computer Science, February 1986.

[33] Laszlo Neumann. New Efficient Algorithms with Positive Definite Radiosity Matrix. *Fifth Eurographics Workshop*

on Rendering, pages 219–237, June 1994.

[34] Yves Nievergelt. Making any radiosity matrix symmetric positive definite. *J. of the Illuminating Engineering Society*, 26(1):165–172, 1997.

[35] Tomoyuki Nishita and Eihachiro Nakamae. Continuous tone representation of 3-D objects taking account of shadows and interreflection. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):23–30, July 1985.

[36] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, Cambridge, 1992.

[37] Holly Rushmeier, Greg Ward, Christine Piatko, Phil Sanders, and Bert Rust. Comparing real and synthetic images: Some ideas about metrics. In *Rendering Techniques '95*, pages 82–91. Springer-Verlag, 1995. http://radsite.lbl.gov/mgf/compare.html.

[38] Holly E. Rushmeier, Charles Patterson, and Aravindan Veerasamy. Geometric simplification for indirect illumination calculations. In *Proc. Graphics Interface '93*, pages 227–236, Toronto, Ontario, May 1993. Canadian Inf. Proc. Soc.

[39] Holly E. Rushmeier and Kenneth E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):293–302, July 1987.

[40] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[41] Peter Schröder. Numerical integration for radiosity in the presence of singularities. In *Fourth Eurographics Workshop on Rendering*, Paris, June 1993.

[42] Peter Schroder. Wavelet Radiosity: Wavelet Methods for Integral Equations. In *ACM SIGGRAPH '96 Course Notes - Wavelets in Computer Graphics*, pages 143–165. 1996. http://csvax.cs.caltech.edu/~ps/waveletcourse/.

[43] Peter Schroder, Steven J. Gortler, Michael F. Cohen, and Pat Hanrahan. Wavelet projections for radiosity. *Computer Graphics Forum*, 13(2):141–151, June 1994.

[44] Peter Schröder and Pat Hanrahan. On the form factor between two polygons. *Computer Graphics (SIGGRAPH '93 Proceedings)*, August 1993. http://csvax.cs.caltech.edu/~ps/formfactor/ffpaper.ps.gz.

[45] F. S. Shaw. *An Introduction to Relaxation Methods*. Dover, 1953.

[46] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Computer Science Tech. Report 94-125, Carnegie Mellon University, Pittsburgh, PA, 1994. http://www.cs.cmu.edu/~quake/papers.html.

[47] Peter Shirley. Test scenes for comparing global illumination algorithms, 1994. http://radsite.lbl.gov/mgf/scene/erw5.tar.Z.

[48] François Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann, San Francisco, 1994.

[49] François X. Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. A global illumination solution for general reflectance distributions. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):187–196, July 1991.

[50] Francois Sillion. Clustering and Volume Scattering for Hierarchical Radiosity Calculations. In *Fifth Eurographics Workshop on Rendering*, pages 105–117, Darmstadt, Germany, June 1994. ftp://w3imagis.imag.fr/Publications/fxs/S94.ps.gz.

[51] Philipp Slusallek, Michael Schroder, Marc Stamminger, and Hans-Peter Seidel. Smart Links and Efficient Reconstruction for Wavelet Radiosity. In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 240–251, New York, NY, 1995. Springer-Verlag.

[52] Brian Smits, James Arvo, and Donald Greenberg. A clustering algorithm for radiosity in complex environments. In *SIGGRAPH '94 Proc.*, pages 435–442, July 1994. http://csvax.cs.caltech.edu/~arvo/papers/Clustering.ps.

[53] Brian E. Smits, James R. Arvo, and David H. Salesin. An importance-driven radiosity algorithm. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(4), July 1992.

[54] Ephraim M. Sparrow. On the calculation of radiant interchange between surfaces. In *Modern Developments in Heat Transfer*, New York, 1963. Academic Press.

[55] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers, San Francisco, CA, 1996. http://www.amath.washington.edu/~stoll/pub.html.

[56] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, Box 157, Wellesley MA 02181, 1986.

[57] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 61–69, July 1991.

[58] Roy Troutman and Nelson Max. Radiosity algorithms using higher-order finite elements. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 209–212, August 1993.

[59] John R. Wallace, Kells A. Elmquist, and Eric A. Haines. A ray tracing algorithm for progressive radiosity. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 315–324, July 1989.

[60] Gregory J. Ward. The RADIANCE lighting simulation and rendering system. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 459–472. ACM SIGGRAPH, ACM Press, July 1994. http://radsite.lbl.gov/radiance/papers/sg94.1/paper.html

[61] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):85–92, Aug. 1988. http://radsite.lbl.gov/radiance/papers/sg88/paper.html.

[62] Wei Xu and Donald S. Fussell. Constructing Solvers for Radiosity Equation Systems. *Fifth Eurographics Workshop on Rendering*, pages 207–217, June 1994.

[63] David M. Young. *Iterative solution of large linear systems*. Academic Press, New York, NY, 1971.

[64] Harold R. Zatz. Galerkin radiosity: A higher-order solution method for global illumination. *Computer Graphics (SIGGRAPH '93 Proceedings)*, August 1993.

Appendix A

A.1. Wavelet Coefficients

In this appendix we give a brief introduction to how the push, pull, and inter-element radiosity transfer operations can be carried out in wavelet radiosity [22]. We then list the coefficients required for these operations.

To 'push' radiosity from a patch to one of its children requires a single linear transformation, and can be accomplished by a matrix multiply. Generally, if b is a vector holding the basis coefficients of the radiosity function for the parent patch, and b_i holds the coefficients for the ith child patch, then $b_i = P_i b$. However, for wavelet radiosity over rectangular patches we can use the tensor product of 1D bases to cover a 2D surface, so that $f(x,y) = g(x)g(y)$. Thus we store our basis coefficients in an $n \times n$ matrix, B, where n is the order of the basis, and our push is accomplished by a tensor matrix multiplication as follows:

$$B_i = P_i B P_i^T \tag{12}$$

Similarly, for a 'pull' operation from all the children of a patch to the parent patch, we have:

$$B = \sum_i L_i B_i L_i^T \tag{13}$$

To calculate the transfer of radiosity between two patches, the source patch having coefficients B_s, and the destination B_d, we need a transfer matrix T. For the general, non-tensor case, this can be calculated from a matrix of kernel samples from the source to the destination patches, K, and two other matrices, as follows:

$$T = RKS \tag{14}$$

Here S transforms the source coefficients to quadrature samples on the source patch, K transforms these to the equivalent samples on the receiving patch, and R projects these into the destination basis. We then have:

$$b_d = T b_s \tag{15}$$

For our tensor-product bases, K and T are four-dimensional matrices, and we calculate K as follows:

$$K_{ijkl} = \kappa(map_s(x_i, x_j), map_d(x_k, x_l)) , \tag{16}$$

where κ is defined in Equation 2, and the *map* functions map from the parameter space ([-1, 1],[-1, 1]) to a point on the source or destination patches.

Calculating T from K is not quite as straightforward as a pair of tensor matrix multiplications, because, while S and R both represent 1D linear transformations, K represents a full, 2D linear transformation[1]. Instead, we find that:

$$T_{ijkl} = \sum_s \sum_t S_{is} S_{jt} \left(\sum_u \sum_v R_{ku} R_{lv} K_{stuv} \right). \tag{17}$$

To take advantage of existing matrix-multiply routines, this can be calculated in two stages:

$$T'_{ij} = R K_{ij} R^T \text{, and then} \tag{18}$$

1. In fact, we could store R and S as full, 2D linear transformations, so that calculating the transfer coefficients *could* be achieved by a couple of simple matrix multiplications; these would be 4×4 matrix multiplies for the M2 basis, for example. While this would be simpler conceptually, and for coding, it would take more arithmetic operations than using the (more involved) tensor multiplications, so we recommend the above approach.

$$T_{ij} = \sum_s \sum_t S_{is} S_{jt} T'_{st} . \tag{19}$$

Note that T_{ij}, T'_{st}, and R are $n \times n$ matrices, and S_{is} and S_{jt} are scalars. R and S are given below for the multiwavelet bases. For the flatlets, they are both the identity matrix, and thus $T = K$. Finally, from T the transfer of radiosity can be calculated as follows:

$$B_d = \sum T_{ij} B_{sij} \tag{20}$$

The coefficients we used for wavelet radiosity are listed below. The first part of the name denotes the basis, and the second part the purpose: `P` for push matrix, `L` for pull matrix, `R` for kernel-to-destination matrix, `S` for source-to-kernel matrix, and `x` for a quadrature position matrix.

```
=== Tensor Push/Pull matrices ================

M2_P0 = [1.0, -0.5]
        [0.0, 0.5]
M2_P1 = [1.0, 0.5]
        [0.0, 0.5]
M2_L0 = [0.5, 0.0]
        [-0.75, 0.25]
M2_L1 = [0.5, 0.0]
        [0.75, 0.25]

F2_P0 = [1.25, -0.25]
        [0.75, 0.25]
F2_P1 = [0.25, 0.75]
        [-0.25, 1.25]
F2_L0 = [0.5625, 0.3125]
        [-0.0625, 0.1875]
F2_L1 = [0.1875, -0.0625]
        [0.3125, 0.5625]

M3_P0 = [1.0, -0.5, 0.25]
        [0.0, 0.5, -0.5]
        [0.0, 0.0, 0.25]
M3_P1 = [1.0, 0.5, 0.25]
        [0.0, 0.5, 0.5]
        [0.0, 0.0, 0.25]
M3_L0 = [0.5, 0.3125, 0.125]
        [-0.75, 0.25, -0.25]
        [0.0, -0.9375, 0.125]
M3_L1 = [0.5, -0.3125, 0.125]
        [0.75, 0.25, 0.25]
        [0.0, 0.9375, 0.125]

F3_P0 = [1.375, -0.5, 0.125]
        [0.625, 0.5, -0.125]
        [0.125, 1.0, -0.125]
F3_P1 = [-0.125, 1.0, 0.125]
        [-0.125, 0.5, 0.625]
        [0.125, -0.5, 1.375]
F3_L0 = [0.583333, 0.291667, 0.125]
        [-0.0833333, 0.25, 0.333333]
        [0.0, -0.0416667, 0.0416667]
F3_L1 = [0.0416667, -0.0416667, 0.0]
        [0.333333, 0.25, -0.0833333]
        [0.125, 0.291667, 0.583333]
```

```
=== Tensor transfer matrices ===================

M2_R =  [0.5, 0.5]
        [-0.866025, 0.866025]
M2_S =  [1.0, -0.57735]
        [1.0, 0.57735]
M2_x =  [-0.57735]
        [0.57735]

M3_R =  [0.0, 1.0, 0.0]
        [-0.645497, 0.0, 0.645497]
        [0.833333, -1.66667, 0.833333]
M3_S =  [0.555556, -0.430331, 0.333333]
        [0.888889, 0.0, 0.0]
        [0.555556, 0.430331, 0.333333]
M3_x =  [-0.774597]
        [0.0]
        [0.774597]
```

A.2. An Example Run File

The following is the run file output for progressive radiosity with substructuring during the blocker experiment. This file gives snapshots of error, memory use, and other statistics during the run of a radiosity method. The last line represents the state at convergence. Relevant parameters are listed first, followed by a line that names the fields for the subsequent data lines, and then a single data line for each checkpoint.

```
avar blocker_height = 1.3      // separation of blocker and receiver is 1.3
renderer rad_2.0
method hprog                   // use progressive radiosity with substructuring
sub 7                          // 1/edge_len = 7 (granularity of shooting mesh)
esub 7                         // 1/edge_len2 = 7 (initial granularity of receiving mesh)
error 0.5                      // t_ratio = 0.5 (termination criterion)
scene blocker
quads 3                        // there are 3 initial triangular/rectangular elements
srcPatches 230                 // after meshing, there are 230 patches.
ID time stage elements shootErr  resErr   rays mem     iterations err       mean
--------------------------------------------------------------------------------
----
0  0    1     230      1         1        0    17.0703 0          1.0422    -1.56597
1  0.19 5     242      1         0.51884  854  23.3516 4          0.711783  0.840823
2  0.4  2     305      1         0.678834 1974 28.0273 8          0.616771  0.721792
3  0.58 2     335      1         0.645864 2899 30.2539 11         0.5133    0.601377
4  0.76 2     350      1         0.778721 3888 31.3672 14         0.405155  0.481565
5  0.93 5     383      1         0.653611 4618 33.8164 16         0.335625  0.402092
6  1.12 2     401      1         0.752922 5755 35.1523 19         0.229482  0.282337
7  1.3  4     467      1         0.607409 6694 40.0508 21         0.131116  0.163727
8  1.51 2     530      1         0.667729 7752 44.7266 23         0.0989429 0.124096
9  1.72 2     584      1         0.674312 8912 48.7344 25         0.0354268 0.0450835
10 1.79 8     614      0.0968708 0        8952 50.9609 26         0.0217024 0.00579821
```

The fields listed above correspond to the following:

`time` is the time since the method was started
`stage` is an ID corresponding to which section of the algorithm is currently being executed.
`elements` is the current number of (receiving) mesh elements

`shootErr` is the amount of power last shot as a fraction of the original shot power
`rays` is the number of rays cast so far
`mem` is the amount of memory used, in kilobytes.
`resErr` measures the radiosity delta between successive shots.
`err` is the solution error, measured as in Section 3.2.3.
`mean` is the average deviation from the reference solution; it shows whether the current solution underestimates or overestimates the total scene radiosity.

A.3. An Example Mesh Output File

This is a (truncated) example of a mesh checkpoint file for the progressive radiosity method during the blocker experiment:

```
object "blocker"
    camera +params 0.697115 [[1 0 0 0]
          [0 0.85185 0.523786 0]
          [0 -0.523786 0.85185 0]
          [0 0 0 1]]
          [[1 0 0 0]
          [0 1 0 0]
          [0 0 1 0]
          [-0.00985222 0.201923 0 1]]
    object "light"
        colour [0 0 0]
        avar emittance light [5 20 50] [1 1 1]
        points [[-0.3 1 -0.3] [0.3 1 -0.3] [0.3 1 0.3] [-0.3 1 0.3] ...]
        poly +clrs 441 +grid 0 1 2 3 0 1 2 3 [1 0 1] 20 20 [
                0 23 24 4 0 23 24 4 [20 20 20]
                4 24 25 5 4 24 25 5 [20 20 20]
                ...]
    end
    colour [0.5 0.5 0.5]
    object "wall"
        points [[-1 -1 1] [1 -1 1] [1 -1 -1] [-1 -1 -1] ...]
        poly +clrs 4225 +grid 0 1 2 3 0 1 2 3 [1 0 1] 64 64 [
                0 67 68 4 0 67 68 4 [0.127858 0.127858 0.127858]
                4 68 69 5 4 68 69 5 [0.130446 0.130446 0.130446]
                ...]
    end
    colour [0.5 0.25 0.5]
    object "wall"
        points [[-0.2 0.3 0.2] [0.2 0.3 0.2] [0.2 0.3 -0.2] [-0.2 0.3 -0.2] ...]
        poly +clrs 196 +grid 0 1 2 3 0 1 2 3 [1 0 1] 13 13 [
                0 16 17 4 0 16 17 4 [1.57537 0.787684 1.57537]
                4 17 18 5 4 17 18 5 [1.61802 0.809011 1.61802]
                ...]
    end
end
```

The key directives in this file are the ‘points’ command, which defines a list of points in world space, and the ‘poly’ command, which defines a polygon based on indices into the current points list. The poly command takes optional parameters, such as a list of mesh patches (the ‘+grid’ option above) or quadtree nodes. The other directives act to group and position these base polygons.

A.4. Plotting Examples

To produce the graphs used in this report, we implemented a scripted front-end capable of extracting data from our run files, according to a number of command line options, and producing data series and plot scripts. These were then fed to the `gnuplot` plotting program to produce either on-screen output, or eps files. The usage for this command was as follows:

```
xplot [-data <dataname>] [file.{mif|ppm|eps}] <experiment> <scene> <method> <field1>[/
    start][/end] <field2>[/start][/end]
```

There was a similar command, `explot`, that produced the equalised plots. The table below gives some example plot commands, and the figure numbers of the graphs they produced in the report.

Command	**Figure**
`xplot abut all m2/m3/m4 last err`	**27**
`explot abut all m4/m5/m6/m7/m8 time err/0/0.04`	**29**
`xplot scomplex scn.3 m3/m4 rays err`	**60**
`xplot log scomplex all m4/m5/m6/m7/m8 quads/last newLinks`	**92**

The xplot program pulls the parameters `field1` and `field2` out of the run file (see **A.2**). ‘abut’ is a pseudonym for perpendicular, and ‘scomplex’ refers to the scaled complex scene.

A.5. Code Snippet for Area-to-Point Form Factors

The code snippet below calculates the area-to-point form factor of Equation 5, which we reproduce here:

$$F_{ij} \approx \sum_k \frac{\hat{n}_j \cdot (\underline{r}_{ijk} \times \underline{r}_{ij(k+1)})}{2\pi \|\underline{r}_{ijk} \times \underline{r}_{ij(k+1)}\|}\theta_{k,\,k+1}$$

This is essentially evaluating a contour integral: the `EdgeArea` call below evaluates this integral along one edge of the source polygon. Its arguments `p` and `q` are the vectors from the point in question to the vertices at either end of this edge (they correspond to $\underline{r}_{ijk}$ and $\underline{r}_{ij(k+1)}$ respectively), and `n` is the surface normal at the point ($\hat{n}_j$).

```
Real EdgeArea(const Vector &p, const Vector &q, const Vector &n)
// Numerically stable edge-area calculator. (Including degenerate cases.)
{
      const Real epsilon = 1e-6;
      Real sinFactor, qdotp, projArea;

      qdotp = dot(p, q);
      sinFactor = sqrlen(p) * sqrlen(q) - sqr(qdotp);
      projArea = -dot(cross(p, q), n);

      // If SinFactor > 0 we may apply the formula without problem

      if (sinFactor > 0)
      {
            sinFactor = sqrt(sinFactor);
            return(projArea * (kHalfPi - atan(qdotp / sinFactor)) /
                  (2 * kPi * sinFactor));
      }
```

```
     // If not, we have three remaining cases...

     else if (qdotp > epsilon)       // CASE 1: Zero-length edge}
          return(0);
     else if (qdotp < -epsilon)      // CASE 2: Viewpoint lies on the edge }
          return(0.5);
     else
          return(0.125);             // CASE 3: Viewpoint lies at a vertex }
}

Real FormFactor(Poly p, Point v, Vector n)
// n is the surface normal at v.
{
     return(sum over all edges of p: EdgeArea(edge.v1 - v, edge.v2 - v, n));
}
```

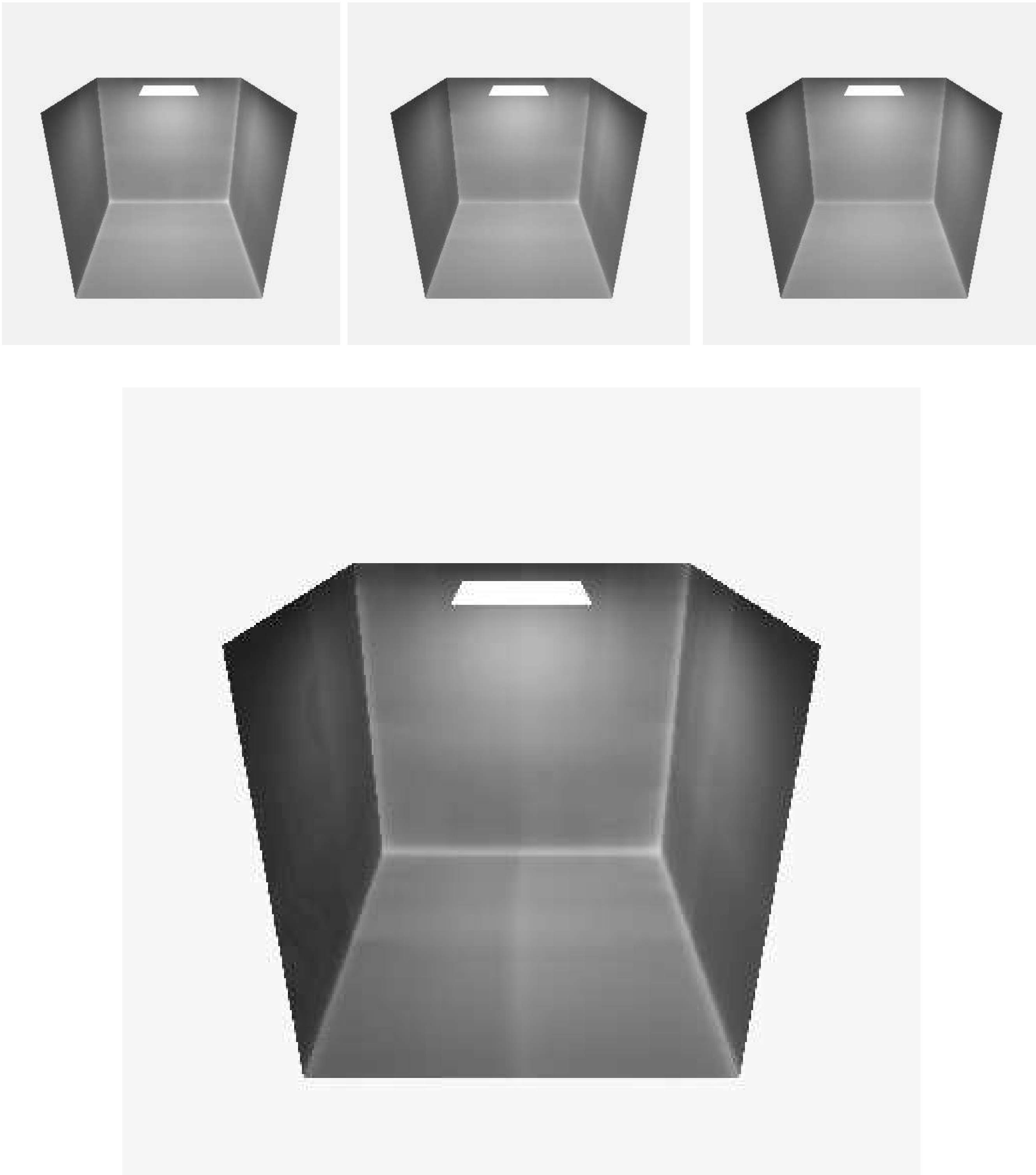


Figure 96: The images in **Figure 69** are reproduced here: this page should be viewed in colour to properly see the colour discontinuity. Below is the same image as the left-hand image above, but enhanced (by increasing the saturation and contrast) and enlarged so as to make the edge and colour discontinuities clearer.