LBL-37858

The RADIANCE Lighting Simulation and Rendering System

Gregory J. Ward

Lighting Group
Building Technologies Program
Lawrence Berkeley Laboratory
(e-mail: GJWard@Lbl.Gov)

ABSTRACT

This paper describes a physically-based rendering system tailored to the demands of lighting design and architecture. The simulation uses a light-backwards ray-tracing method with extensions to efficiently solve the rendering equation under most conditions. This includes specular, diffuse and directional-diffuse reflection and transmission in any combination to any level in any environment, including complicated, curved geometries. The simulation blends deterministic and stochastic ray-tracing techniques to achieve the best balance between speed and accuracy in its local and global illumination methods. Some of the more interesting techniques are outlined, with references to more detailed descriptions elsewhere. Finally, examples are given of successful applications of this free software by others.

CR Categories: I.3.3 [Computer Graphics]: Picture/image generation - *Display algorithms*; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - *Shading*.
Additional Keywords and Phrases: lighting simulation, Monte Carlo, physically-based rendering, radiosity, ray-tracing.

1. Introduction

Despite voluminous research in global illumination and radiosity over the past decade, few practical applications have surfaced in the fields that stand the most to benefit: architecture and lighting design. Most designers who use rendering software employ it in a purely illustrative fashion to show geometry and style, not to predict lighting or true appearance. The designers cannot be blamed for this; rendering systems that promote flash over content have been the mainstay of the graphics industry for years, and the shortcuts employed are well-understood by the software community and well-supported by the hardware manufacturers.

Why has radiosity not yet taken off in the rendering market? Perhaps not enough time has passed since its introduction to the graphics community a decade ago [8]. After all, it took ray-tracing nearly that long to become a mainstream, commercial rendering technique. Another possibility is that the method is too compute-intensive for most applications, or that it simply does not fulfill enough people's needs. For example, most radiosity systems are not well automated, and do not permit general reflectance models or curved surfaces. If we are unable to garner support even from the principal beneficiaries, designers, what does that say of our chances with the rest of the user community?

Acceptance of physically-based rendering is bound to improve†, but researchers must first demonstrate the real-life applicability of their techniques. There have been few notable successes in applying radiosity to the needs of practicing designers [6]. While much research has focused on improving efficiency of the basic radiosity method, problems associated with more realistic, complicated geometries have only recently gotten the attention they deserve [2,19,22]. For whatever reason, it appears that radiosity has yet to fulfill its promise, and it is time to reexamine this technique in light of real-world applications and other alternatives for solving the rendering equation [10].

There are three closely related challenges to physically-based rendering for architecture and lighting design: accuracy, generality and practicality. The first challenge is that the calculation must be accurate; it must compute absolute values in physical units with reasonable certainty. Although recent research in global illumination has studied sources of calculation error [1,20], few researchers bother to compute in physical lighting units, and even fewer have compared their results to physical experiments [15]. No matter how good the theory is, accuracy claims for simulation must ultimately be backed up with comparisons to what is being simulated. The second challenge is that a rendering program must be general. It is not necessary to simulate every physical lighting phenomenon, but it is important to do enough that the unsolvable rendering problems are either unimportant or truly exceptional. The third challenge for any rendering system is that it be practical. This includes a broad spectrum of requirements, from being reliable (i.e. debugged and tested) to being application-friendly, to producing good results in a reasonable time. All three of the above challenges must be met if a physically-based rendering package is to succeed, and all three must be treated with equal importance.

Radiance is the name of a rendering system developed by the author over the past nine years at the Lawrence Berkeley Laboratory (LBL) in California and the Ecole Polytechnique Federale de Lausanne (EPFL) in Switzerland. It began as a study in ray-tracing algorithms, and after demonstrating its potential for saving energy through better lighting design, acquired funding from the U.S. Department of Energy and later from the Swiss government. The first free software release was in 1989, and since then it has found an increasing number of users in the research and design community. Although it has never been a commercial product, *Radiance* has benefited enor-

†The term "physically-based rendering" is used throughout the paper to refer to rendering techniques based on physical principles of light behavior for local and global illumination. The term "simulation" is more general, referring to any algorithm that mimics a physical process.

mously from the existence of an enthusiastic, active and growing user base, which has provided invaluable debugging help and stress-testing of the software. In fact, most of the enhancements made to the system were the outcome of real or perceived user requirements. This is in contrast to the much of the research community, which tends to respond to intriguing problems before it responds to critical ones. Nine years of user-stimulated software evolution gives us the confidence to claim we have a rendering system that goes a long way towards satisfying the needs of the design community. Further evidence has been provided by the two or three design companies who have abandoned their own in-house software (some of which cost over a million dollars to develop) in favor of *Radiance.*

In this paper, we describe the *Radiance* system design goals, followed with the principal techniques used to meet these goals. We follow with examples of how users have applied *Radiance* to specific problems, followed by conclusions and ideas for future directions.

2. System Design Goals

The original goals for the *Radiance* system were modest, or so we thought. The idea was to produce an accurate tool for lighting simulation and visualization based on ray-tracing. Although the initial results were promising, we soon learned that there was much more to getting the simulation right than plugging proper values and units into a standard ray-tracing algorithm. We needed to overcome some basic shortcomings. The main shortcoming of conventional ray-tracing is that diffuse interreflection between surfaces is approximated by a uniform "ambient" term. For many scenes, this is a poor approximation, even if the ambient term is assigned correctly. Other difficulties arise in treating light distribution from large sources such as windows, skylights, and large fixtures. Finally, reflections of lights from mirrors and other secondary sources are problematic. These problems, which we will cover in some detail later, arose from the consideration of our system design goals, given below.

The principal design goals of *Radiance* were to:

1. Ensure accurate calculation of luminance
2. Model both electric light and daylight
3. Support a variety of reflectance models
4. Support complicated geometry
5. Take unmodified input from CAD systems

These goals reflect many years of experience in architectural lighting simulation; some of them are physically-motivated, others are user-motivated. All of them must be met before a lighting simulation tool can be of significant value to a designer.

2.1. Ensure Accurate Calculation of Luminance

Accuracy is one of the key challenges in physically-based rendering, and luminance (or the more general "spectral radiance") is probably the most versatile unit in lighting. Photometric units such as luminance are measured in terms of visible radiation, and radiometric units such as radiance are measured in terms of power (energy/time). Luminance represents the quantity of visible radiation passing through a point in a given direction, measured in lumens/steradian/meter^2 in SI units. Radiance is the radiometric equivalent of luminance, measured in watts/steradian/meter^2. Spectral radiance simply adds a dependence on wavelength to this. Luminance and spectral radiance are most closely related to a pixel, which is what the eye actually "sees." From this single unit, all other lighting metrics can be derived. Illuminance, for example, is the integral of luminance over a projected hemisphere (lumens/meter^2 or "lux" in SI units). Luminous intensity and luminous flux follow similar derivations. By computing the most basic lighting unit, our simulation will adapt more readily to new applications.

To assure that a simulation delivers on its promise, it is essential that the program undergo periodic validation. In our case, this means comparing luminance values predicted by *Radiance* to measurements of physical models. An initial validation was completed in 1989 by Grynberg [9], and subsequent validations by ourselves and others confirm that the values are getting better and not worse [14].

2.2. Model Both Electric Light and Daylight

In order to be general, a lighting calculation must include all significant sources of illumination. Daylight simulation is of particular interest to architects, since the design of the building facade and to a lesser degree the interior depends on daylight considerations.

Initially, *Radiance* was designed to model electric light in interior spaces. With the addition of algorithms for modeling diffuse interreflection [25], the software became more accurate and capable of simulating daylight (both sun and sky contributions) for building interiors and exteriors. The role of daylight simulation in *Radiance* was given new importance when the software was chosen by the International Energy Agency (IEA) for its daylight modeling task* [4].

2.3. Support a Variety of Reflectance Models

Luminance is a directional quantity, and its value is strongly determined by a material's reflectance/transmittance distribution function. If luminance is calculated using a Lambertian (i.e. diffuse) assumption, specular highlights and reflections are ignored and the result can easily be wrong by a hundred times or more. We cannot afford to lose directional information if we hope to use our simulation to evaluate visual performance, visual comfort and aesthetics.

A global illumination program is only as general as its local illumination model. The standard model of ambient plus diffuse plus Phong specular is not good enough for realistic image synthesis. *Radiance* includes the ability to model arbitrary reflectance and transmittance functions, and we have also taken empirical measurements of materials and modeled them successfully in our system [29].

2.4. Support Complicated Geometry

A lighting simulation of an empty room is not very interesting, nor is it very informative. The contents of a room must be included if light transfer is to be calculated correctly. Also, it is difficult for humans to evaluate aesthetics based on visualizations of empty spaces. Furniture, shadows and other details provide the visual cues a person needs to understand the lighting of a space. Modeling exteriors is even more challenging, often requiring hundreds of thousands of surfaces.

Although we leave the definition of "complicated geometry" somewhat loose, including it as a goal means that we shall not limit the geometric modeling capability of our simulation in any fundamental way. To be practical, data structure size should grow linearly (at worst) with geometric complexity, and there should be no built-in limit as to the number of surfaces. To be accurate, we shall support a variety of surface primitives, also ensuring our models are as memory-efficient as possible. To be general, we shall provide N-sided polygons and a mechanism for interpolating surface normals, so any reasonable shape may be represented. Finally, computation time should have a sublinear relationship to the number of surfaces so that the user does not pay an unreasonable price for accurate modeling.

*The IEA is a consortium of researchers from developed nations cooperatively seeking alternative energy sources and ways of improving energy efficiency in their countries.

2.5. Take Unmodified Input from CAD Systems

If we are to model complicated geometry, we must have a practical means to enter these models into our simulation. The creation of a complicated geometric model is probably the most difficult task facing the user. It is imperative that the user be allowed every means to simplify this task, including advanced CAD systems and input devices. If our simulation limits this process in any way, its value is diminished.

Therefore, to the greatest degree possible, we must accept input geometry from any CAD environment. This is perhaps the most difficult of the goals we have outlined, as the detail and quality of CAD models varies widely. Many CAD systems and users produce only 2D or wireframe models, which are next to useless for simulation. Other CAD systems, capable of producing true 3D geometric models, cannot label the component surfaces and associate the material information necessary for an accurate lighting simulation. These systems require a certain degree of user intervention and post-processing to complete the model. Even the most advanced CAD systems, which produce accurate 3D models with associated surface data, do not break surfaces into meshes suitable for a radiosity calculation. The missing information must either be added by the user, inferred from the model, or the need for it must be eliminated. In our case, we eliminate this need by using something other than a radiosity (i.e. finite element) algorithm.

CAD translators have been written for *AutoCAD, GDS, ArchiCAD, DesignWorkshop, StrataStudio, Wavefront,* and *Architrion,* among others. None of these translators requires special intervention by the user to reorient surface normals, eliminate T-vertices, or mesh surfaces. The only requirement is that surfaces must somehow be associated with a layer or identifier that indicates their material type.

3. Approach

We have outlined the goals for our rendering system and linked them back to the three key challenges of accuracy, generality and practicality. Let us now explore some of the techniques we have found helpful in meeting these goals and challenges.

We start with a basic description of the problem we are solving and how we go about solving it in section 3.1, followed by specific solution techniques in sections 3.2 to 3.5. Sections 3.6 to 3.9 present some important optimizations, and section 3.10 describes the overall implementation and use of the system.

3.1. Hybrid Deterministic/Stochastic Ray Tracing

Essentially, *Radiance* uses ray-tracing in a recursive evaluation of the following integral equation at each surface point:

$$L_r(\theta_r,\phi_r) = L_e(\theta_r,\phi_r) + \int_0^{2\pi}\int_0^{\pi} L_i(\theta_i,\phi_i)\,\rho_{bd}(\theta_i,\phi_i;\theta_r,\phi_r)\,|\cos\theta_i|\,\sin\theta_i\,d\theta_i\,d\phi_i \quad (1)$$

where:

θ is the polar angle measured from the surface normal

ϕ is the azimuthal angle measured about the surface normal

$L_e(\theta_r,\phi_r)$ is the emitted radiance (watts/steradian/meter^2 in SI units)

$L_r(\theta_r,\phi_r)$ is the reflected radiance

$L_i(\theta_i,\phi_i)$ is the incident radiance

$\rho_{bd}(\theta_i,\phi_i;\theta_r,\phi_r)$ is the bidirectional reflectance-transmittance distribution function (steradian^{-1})

This equation is essentially Kajiya's rendering equation [10] with the notion of energy transfer between two points replaced by energy passing through a point in a specific direction (i.e. the definition of radiance). This formula has been documented many times, going back before the standard definition of ρ_{bd} [16]. Its generality and simplicity provide the best foundation for building a lighting simulation.

This formulation of the rendering problem is a natural for ray tracing because it gives outgoing radiance in terms of incoming radiance over the projected sphere, without any explicit mention of the model geometry. The only thing to consider at any one time is the light interaction with a specific surface point, and how best to compute this integral from spawned ray values. Thus, no restrictions are placed on the number or shape of surfaces or surface elements, and discretization (meshing) of the scene is unnecessary and even irrelevant.

Although it is possible to approximate a solution to this equation using uniform stochastic sampling (i.e. Monte Carlo), the convergence under most conditions is so slow that such a solution is impractical. For example, a simple outdoor scene with a ground plane, a brick and the sun would take days to compute using naive Monte Carlo simply because the sun is so small (0.5° of arc) in comparison to the rest of the sky. It would take many thousands of samples per pixel to properly integrate light coming from such a concentrated source.

The key to fast convergence is in deciding *what* to sample by removing those parts of the integral we can compute deterministically and gauging the importance of the rest so as to maximize the payback from our ray calculations. In the case of the outdoor scene just described, we would want to consider the sun as an important contribution to be sampled separately, thus removing the biggest source of variance from our integral. Instead of relying on random samples over the hemisphere, we send a single sample ray towards the sun, and if it arrives unobstructed, we use a deterministic calculation of the total solar contribution based on the known size and luminosity of the sun as a whole. We are making the assumption that the sun is not partially occluded, but such an assumption would only be in error within the penumbra of a solar shadow region, and we know these regions to represent a very small portion of our scene.

Light sources cause peaks in the incident radiance distribution, $L_i(\theta_i,\phi_i)$. Directional reflection and transmission cause peaks in the scattering function, ρ_{bd}. This will occur for reflective materials near the mirror angle, and in the refracted direction of dielectric surfaces (e.g. glass). Removing such peak reflection and transmission angles by sending separate samples reduces the variance of our integral at a comparatively modest cost. This approach was introduced at the same time as ray-tracing by Whitted [31]. Further improvements were made by adding stochastic sampling to the deterministic source and specular calculations by Cook in the first real linking of stochastic and deterministic techniques [5]. *Radiance* employs a tightly coupled source and specular calculation, described in [29].

3.2. Cached Indirect Irradiances for Diffuse Interreflection

No matter how successful we are at removing the specular reflections and direct illumination from the integral (1), the cost of determining the remaining diffuse indirect contributions is too great to recalculate at every pixel because this requires tracing hundreds of rays to reduce the variance to tolerable levels. Therefore, most ray-tracing calculations ignore diffuse interreflection between surfaces, using a constant "ambient" term to replace the missing energy.

Part of the reason a constant ambient value has been accepted for so long (other than the cost of replacing it) is that diffuse interreflection changes only gradually over surfaces. Thus, the contrast-sensitive eye usually does not object to the loss of subtle shading that accompanies an ambient approximation. However, the inaccuracies that result *are* a problem if one

wants to know light levels or see the effects of daylight or indirect lighting systems.

Since indirect lighting changes gradually over surfaces, it should be possible to spread out this influence over many pixels to obtain a result that is smooth and accurate at a modest sampling cost. This is exactly what we have done in *Radiance*. The original method for computing and using cached irradiance values [25] has been enhanced using gradient information [28].

The basic idea is to perform a full evaluation of Equation (1) for indirect diffuse contributions only as needed, caching and interpolating these values over each surface. Direct and specular components are still computed on a per-pixel basis, but hemispherical sampling occurs less frequently. This gives us a good estimate of the indirect diffuse contribution when we need it by sending more samples than we would be able to afford for a pixel-independent calculation. The approach is effectively similar to finite element methods that subdivide surfaces into patches, calculate accurate illumination at one point on each patch and interpolate the results. However, an explicit mesh is not used in our method, and we are free to adjust the density of our calculation points in response to the illumination environment. Furthermore, since we compute these view-independent values only as needed, separate form factor and solution stages do not have to complete over the entire scene prior to rendering. This can amount to tremendous savings in large architectural models where only a portion is viewed at any one time.

Figure 1 looks down on a diffuse sphere in a room with indirect lighting only. A blue dot has been placed at the position of each indirect irradiance calculation. Notice that the values are irregularly spaced and denser underneath the sphere, on the sphere and near the walls at the edges of the image. Thus, the spacing of points adapts to changing illumination to maintain constant accuracy with the fewest samples.

To compute the indirect irradiance at a point in our scene, we send a few hundred rays that are uniformly distributed over the projected hemisphere. If any of our rays hits a light source, we disregard it since the direct contribution is computed separately. This sampling process is applied recursively for multiple reflections, and it does not grow exponentially because each level has its own cache of indirect values.

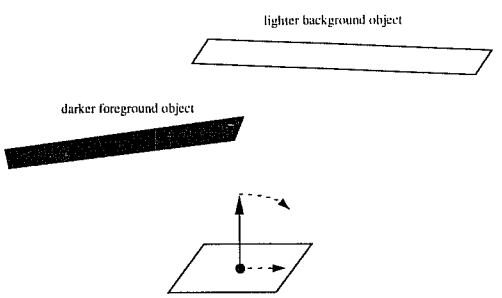


Figure 2: Irradiance gradients due to bright and dark objects in the environment.

Our hemisphere samples not only tell us the total indirect illumination, they also give us more detailed information about the locations and brightnesses of surfaces visible from the evaluation point. This information may be used to predict how irradiance will change as a function of point location and surface orientation, effectively telling us the first derivative (gradient) of the irradiance function. For example, we may have a bright reflecting surface behind and to the right of a darker surface as shown in Figure 2. Moving our evaluation point to the right would yield an increase in the computed irradiance (i.e. the translational gradient is positive in this direction), and our samples can tell us this. A clockwise rotation of the surface element would also cause an increase in the irradiance value (i.e. the rotational gradient is positive in this direction), and our hemisphere samples contain this information as well. Formalizing these observations, we have developed a numerical approximation to the irradiance gradient based on hemisphere samples. Unfortunately, its derivation does not fit easily into a general paper, so we refer the reader to the original research [28].

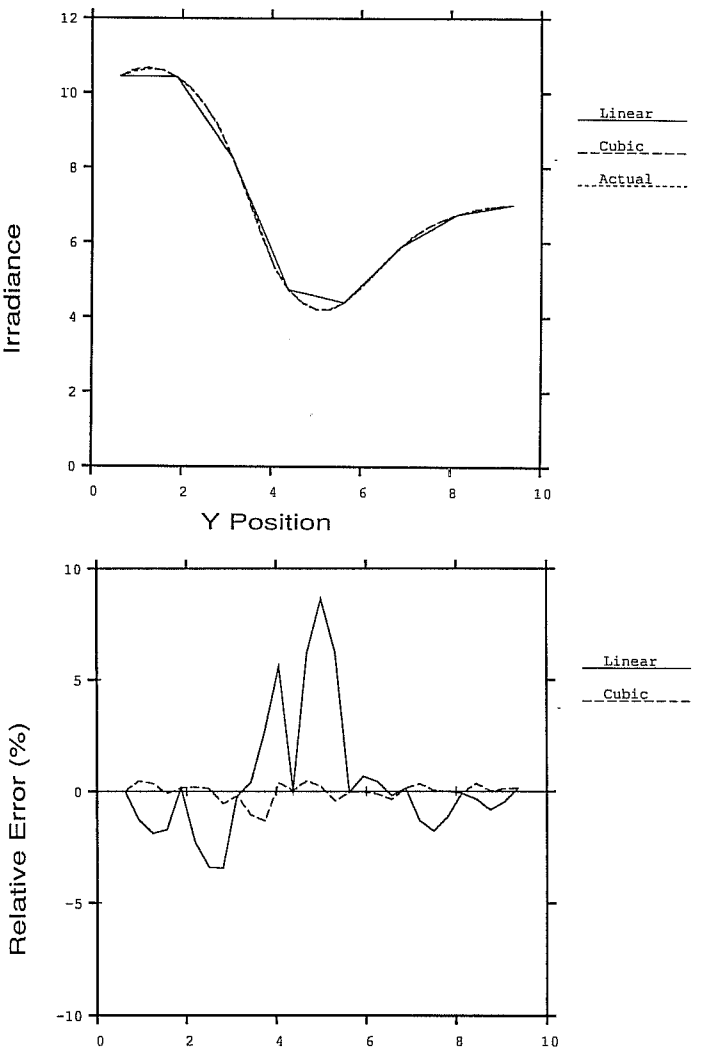


Figure 3a,b. *Plots showing the superiority of gradient interpolation for indirect irradiance values. The reference curve is an exact calculation of the irradiance along the red line in Figure 1. The linear interpolation is equivalent to Gouraud shading between evenly spaced points, as in radiosity rendering. The Hermite cubic interpolation uses the gradient values computed by Radiance, and is not only smoother but demonstrably more accurate than a linear interpolation.*

Knowing the gradient in addition to the value of a function, we can use a higher order interpolation method to get a better irradiance estimate between the calculated points. In effect, we will obtain a smoother and more accurate result without having to do any additional sampling, and with very little overhead. (Evaluating the gradient formulas costs almost nothing compared to computing the hemisphere samples.)

Figure 3a shows the irradiance function across the floor of Figure 1, along the red line. The exact curve is shown overlaid with a linearly interpolated value between regularly spaced cal-

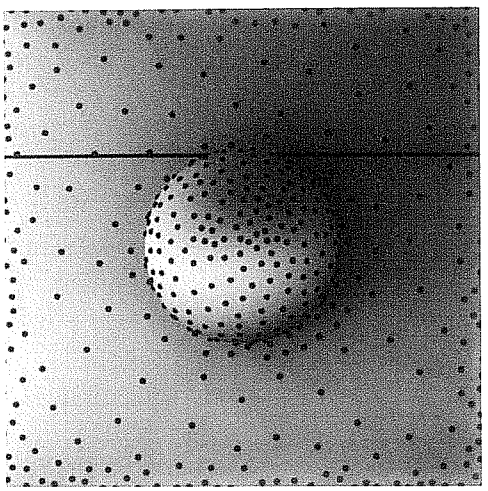


Figure 1. Blue dots show calculation points for diffuse interreflection. (See Figure 3 regarding red line.)

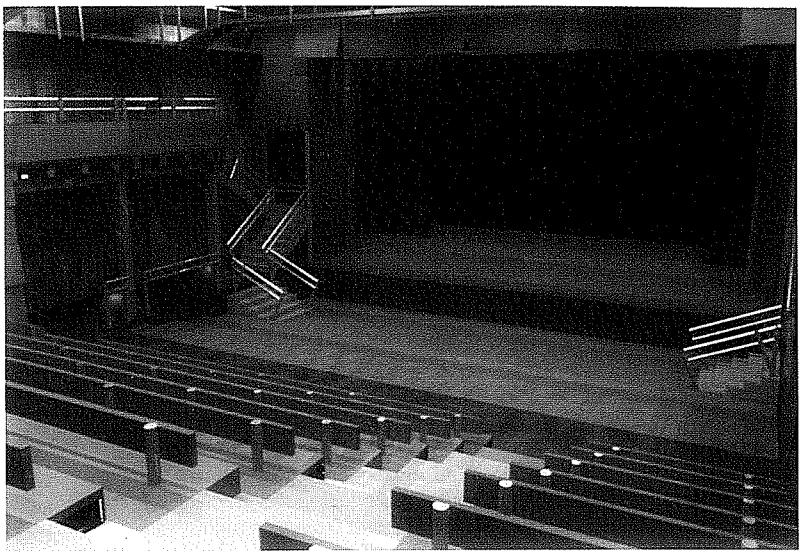


Figure 5. A theater house lighting simulation. The striated shadows on the floor are cast by the catwalks above.

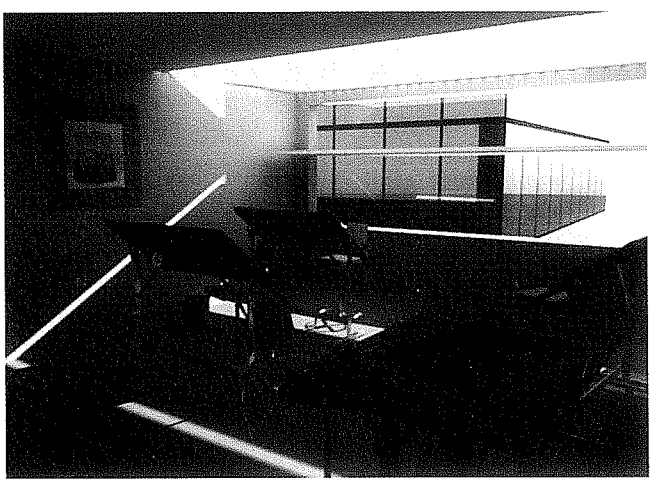


Figure 9a. A drafting office with a mirror light shelf slicing the window and redirecting light upwards.

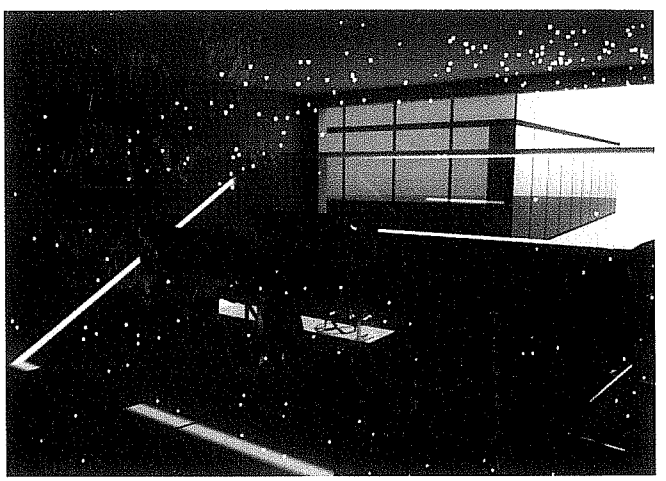


Figure 9b. A Monte Carlo simulation rendered in the same time as 9a, showing the resulting noise.

Figure 9c. A visualization of the illuminance levels on the room surfaces in the drafting office.

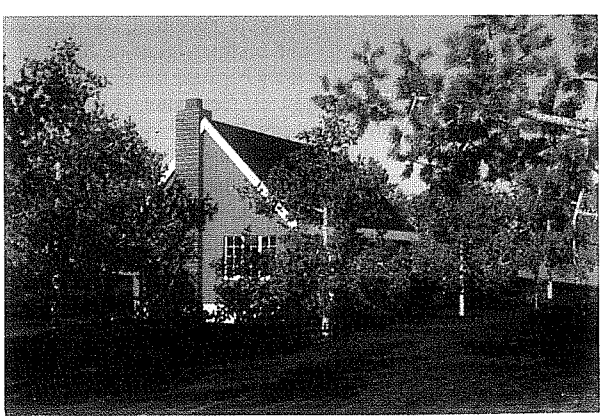


Figure 11. A cabin in a forest using hierarchical octrees to model over a million pine needles.

culation points, and a Hermite cubic interpolation using computed gradients. The cubic interpolation is difficult to separate from the exact curve. Figure 3b shows the relative error for these two interpolation methods, clearly demonstrating the advantage of using gradient information.

Caching indirect irradiances has four important advantages over radiosity methods. First, no meshing is required, since a separate octree data structure is used to hold the calculated values. This lifts restrictions on geometric shapes and complexity, and greatly simplifies user input and scene analysis. Second, we only have to compute those irradiances affecting the portion of the scene being viewed. This speeds rendering time under any circumstance, since our *view-independent* values may be reused in subsequent images (unlike values computed with importance-driven radiosity [20]). Third, the density of irradiance calculations is reduced at each level of interreflection, maintaining constant accuracy while reducing the time required to compute additional bounces. Fourth, the technique adapts to illumination by spacing values more closely in regions where there may be large gradients, without actually using the gradient as a criterion. This eliminates errors that result from using initial samples to decide sampling density [12], and improves accuracy overall. The gradient *is* used to improve value interpolation, yielding a smoother and more accurate result without the Mach-bands that can degrade conventional radiosity images.

3.3. Adaptive Sampling of Light Sources

Although sending one sample ray to each light source is quite reasonable for outdoor scenes, such an approach is impractical for indoor scenes that may have over a hundred light sources. Most rays in a typical calculation are in fact shadow rays. It is therefore worth our while to rule out light sources that are unimportant and avoid testing them for visibility.

The method we use in *Radiance* for reducing the number of shadow rays is described in [26]. A prioritized list of potential source contributions is created at each evaluation of Equation (1). The largest potential contributors (contribution being a function of source output, proximity and ρ_{bd}) are tested for shadows first, and we stop testing when the remainder of the source list is below some fraction of the unoccluded contributions. The remaining source contributions are then added based on statistical estimates of how likely each of them is to be visible.

Figure 4 shows a simple example of how this works. The left column represents our sorted list of potential light source contributions for a specific sample point. We proceed down our list, checking the visibility of each source by tracing shadow rays, and summing together the unobstructed contributions. After each test, we check to see if the remainder of our potential contributions has fallen below some specified fraction of our accumulated total. If we set our accuracy goal to 10%, we can stop testing after four light sources because the remainder of the list is less than 10% of our known direct value. We could either add all of the remainder in or throw it away and our value would still be within 10% of the correct answer. But we can do better than that; we can make an educated guess at the visibility of the remaining sources using statistics. Taking the history of obstructed versus unobstructed shadow rays from previous tests of each light source, we multiply this probability of hitting an untested source by the ratio of successful shadow tests at this point over all successful shadow tests (2/(.9+.55+.65+.95) == 0.65 in this example), and arrive at a reasonable estimate of the remainder. (If any computed multiplier is greater than 1, 1 is used instead.) Our total estimate of the direct contribution at this point is then the sum of the tested light sources and our statistical estimate of the remainder, or 1616 in this example.

We have found this method to be very successful in reducing the number of shadow test rays required, and it is possible to place absolute bounds on the error of the approximation. Most importantly, this type of adaptive shadow testing emphasizes contrast as the primary decision criterion. Contrast is defined as the difference between the luminance at a point and the background luminance divided by the background luminance. If a shadow boundary is below the visible contrast threshold, then an error in its calculation is undetectable by the viewer. Thus, this method produces no *visible* artifacts in its tradeoff of speed for accuracy. Accuracy is still lost in a controlled way, but the resulting image is subjectively flawless, due to the eye's relative insensitivity to absolute light levels.

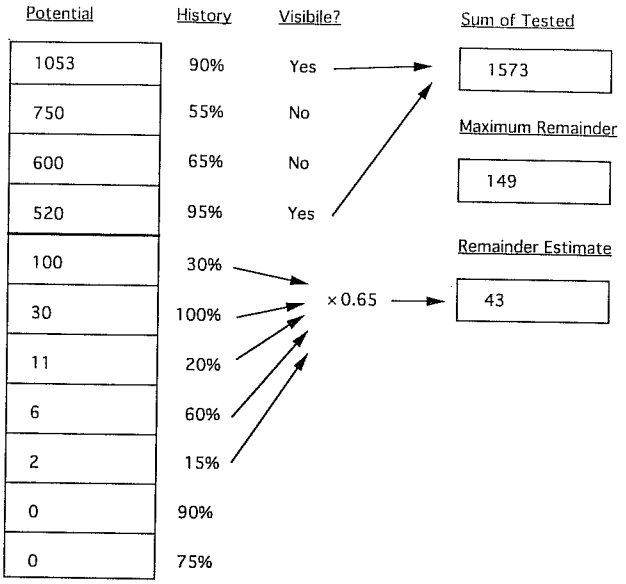


Figure 4: Adaptive shadow testing algorithm, explained in Section 3.4.

Figure 5 shows a theater lighting simulation generated by *Radiance* in 1989. This image contains slightly over a hundred light sources, and originally took about 4 days to render on a Sun-4/260. (The equivalent of about 5 Vax-11/780's.) Using our adaptive shadow testing algorithm reduced the rendering time to 2 days for the same image†. The time savings for scenes with more light sources can be better than 70%, especially if the light sources have narrow output distributions, such as the spotlights popular in overlighted retail applications.

A different problem associated with ray-per-source shadow testing is *inadequate sampling* of large or nearby sources, which threatens simulation accuracy. For example, a single ray cannot adequately sample a fluorescent desk lamp for a point directly beneath it. The simplest approach for sources that are large relative to their distance is to send multiple sample rays. Unfortunately, breaking a source into pieces and sending many rays to it is inefficient for distant points in the room. Again, an adaptive sampling technique is the most practical solution.

In our adaptive technique, we send multiple rays to a light source if its extent is large relative to its distance. We recursively divide such sources into smaller pieces until each piece satisfies some size/distance criterion. Figure 6a shows a long, skinny light source that has been broken into halves repeatedly until each source is small enough to keep penumbra and solid

†The theater model was later rendered in [2] using automatic meshing and progressive radiosity. Meshing the scene caused it to take up about 100 Mbytes of memory, and rendering took over 18 hours on an SGI R3000 workstation for the direct component alone, compared to 5 hours in 11 Mbytes using *Radiance* on the same computer.

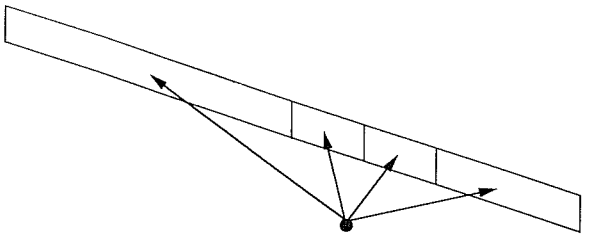


Figure 6a: Linear light source is adaptively split to minimize falloff and visibility errors.

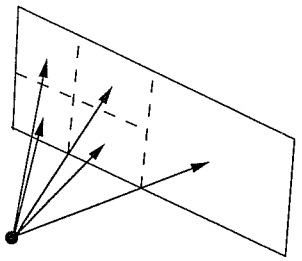


Figure 6b: Area light source is subdivided in two dimensions rather than one.

angle errors in check. Figure 6b shows a similar subdivision of a large rectangular source. A point far away from either source will not result in subdivision, sending only a single ray to some (randomly) chosen location on the source to determine visibility.

3.4. Automatic Preprocessing of "Virtual" Light Sources

Thus far we have accounted for direct contributions from known light sources, specular reflections and transmission, and diffuse interreflections. However, there are still transfers from specular surfaces that will not be handled efficiently by our calculation. A mirror surface may reflect sunlight onto a diffuse or semispecular surface, for example. Although the diffuse interreflection calculation could in principle include such an effect, we are returning to the original problem of insufficient sampling of an intense light source. A small source reflected specularly is still too small to find in a practical number of naive Monte Carlo samples. We have to know where to look.

We therefore introduce "virtual" light sources that do not exist in reality, but are used during the calculation to direct shadow rays in the appropriate directions to find reflected or otherwise transferred light sources. This works for any planar surface, and has been implemented for mirrors as well as prismatic glazings (used in daylighting systems [4]). For example, a planar mirror might result in a virtual sun in the mirror direction from the real sun. When a shadow ray is sent towards the virtual sun, it will be reflected off the mirror to intersect the real sun. An example is shown in Figure 7a. This approach is essentially the same as the "virtual worlds" idea put forth by Rushmeier [18] and exploited by Wallace [24], but it is only carried out for light sources and not for all contributing surfaces. Thus, multiple transfers between specular surfaces can be made practical with this method using intelligent optimization techniques.

The first optimization we apply is to limit the scope of a virtual light source to its affected volume. Given a specific source and a specific specular surface, the influence is usually limited to a certain projected volume. Points that fall outside this volume are not affected and thus it is not necessary to consider the source everywhere. Furthermore, multiple reflections of the source are possible only within this volume. We can thus avoid creating virtual-virtual sources in cases where the volume of one virtual source fails to intersect the second reflecting surface, as shown in Figure 7b. The same holds for thrice redirected sources and so on, and the likelihood that virtual source volumes intersect becomes less likely each time, provided that the reflecting surfaces do not occupy a majority of the space.

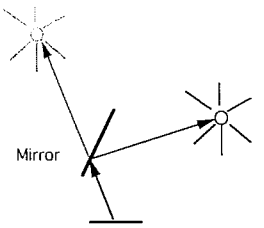


Figure 7a: Virtual source caused by mirror reflection.

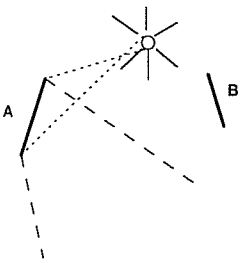


Figure 7b: Source reflection in mirror **A** cannot intersect mirror **B**, so no virtual-virtual source is created.

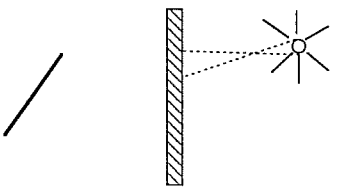


Figure 7c: Source rays cannot reach mirror surface, so no virtual source is created.

To minimize the creation of useless virtual light sources, we check very carefully to confirm that the light in fact has some free path between the source and the reflecting surface before creating the virtual source. For example, we might have an intervening surface that prevents all rays from reaching a reflecting surface from a specific light source, such as the situation shown in Figure 7c. We can test for this condition by sending a number of presampling rays between the light source and the reflecting surface, assuming if none of the rays arrives that the reflecting path must be completely obstructed. Conversely, if none of the rays is obstructed, we can save time during shadow testing later by assuming that any ray arriving at the reflecting surface in fact has a free path to the source, and further ray intersection tests are unnecessary. We have found presampling to be very effective in avoiding wasteful testing of completely blocked or unhindered virtual light source paths.

Figure 8 shows a cross-section of an office space with a light shelf having a mirrored top surface. Exterior to this office is a building with a mirrored glass facade. Figure 9a shows the interior of the office with sunlight reflected by the shelf onto the ceiling. Light has also been reflected by the exterior, glazed building. Light shelf systems utilize daylight very effectively and are finding increasing popularity among designers.

To make our calculation more efficient overall, we have made additional use of "secondary" light sources, described in the next section.

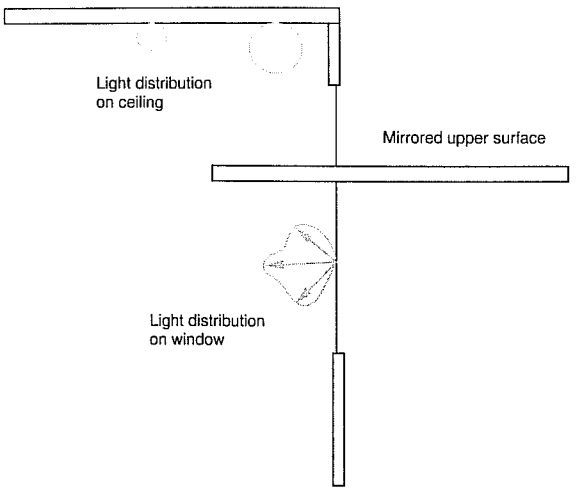


Figure 8: Crossection of office space with mirrored light shelf.

3.5. User-directed Preprocessing of "Secondary" Sources

What happens when daylight enters a space through a skylight or window? If we do not treat such "secondary" emitters specially in our calculation, we will have to rely on the ability of the naive Monte Carlo sampling to find and properly integrate these contributions, which is slow. Especially when a window or skylight is partially obscured by venetian blinds or has a geometrically complex configuration, computing its contribution requires significant effort. Since we know a priori that such openings have an important influence on indoor illumination, we can greatly improve the efficiency of our simulation by removing them from the indirect calculation and treating them instead as part of the direct (i.e. source) component.

Radiance provides a practical means for the user to move such secondary sources into the direct calculation. For example, the user may specify that a certain window is to be treated as a light source, and a separate calculation will collect samples of the transmitted radiation over all points on the window over all directions, a 4-dimensional function. This distribution is then automatically applied to the window, which is treated as a secondary light source in the final calculation. This method was used in Figure 9a not only for the windows, but also for light reflected by the ceiling. Bright solar patches on interior surfaces can make important contributions to interior illumination. Since this was the desired result of our mirrored light shelf design, we knew in advance that treating the ceiling as a secondary light source might improve the efficiency of our calculation. Using secondary light sources in this scene reduced simulation time to approximately one fifth of what it would have been to reach the same accuracy using the default sampling techniques.

Figure 9b shows a Monte Carlo path tracing calculation of the same scene as 9a, and took roughly the same amount of time to compute. The usual optimizations of sending rays to light sources (the sun in this case) and in specular directions were used. Nevertheless, the image is very noisy due to the difficulty of computing interreflection independently at each pixel. Also, locating the sun reflected in the mirrored light shelf is hopeless with naive sampling; thus the ceiling is extremely noisy and the room is not as well lit as it should be.

An important aspect of secondary light sources in *Radiance* is that they have a dual nature. When treated in the direct component calculation, they are merely surfaces with precalculated output distributions. Thus, they can be treated efficiently as light sources and the actual variation that may take place over their extent (e.g. the bright and dark slats of venetian blinds) will not translate into excessive variance in the calculated illumination. However, when viewed directly, they revert to their original form, showing all the appropriate detail. In our office scene example, we can still see through the window despite its treatment as a secondary light source. This is because we treat a ray coming from the eye differently, allowing it to interact with the actual window rather than seeing only a surface with a smoothed output distribution. In fact, only shadow rays see the simplified representation. Specular rays and other sampling will be carried out as if the window was not a light source at all. As is true with the computation of indirect irradiance described in section 3.2, extreme care must be exercised to avoid double-counting of light sources and other inconsistencies in the calculation.

3.6. Hierarchical Octrees for Spatial Subdivision

One of the goals of our simulation is to model very complicated geometries. Ray-tracing is well-suited to calculations in complicated environments, since spatial subdivision structures reduce the number of ray-surface intersection tests to a tiny fraction of the entire scene. In *Radiance*, we use an octree spatial subdivision scheme similar to that proposed by Glassner [7]. Our octree starts with a cube encompassing the entire scene, and recursively subdivides the cube into eight equal subcubes until each voxel (leaf node) intersects or contains less than a certain number of surfaces, or is smaller than a certain size.

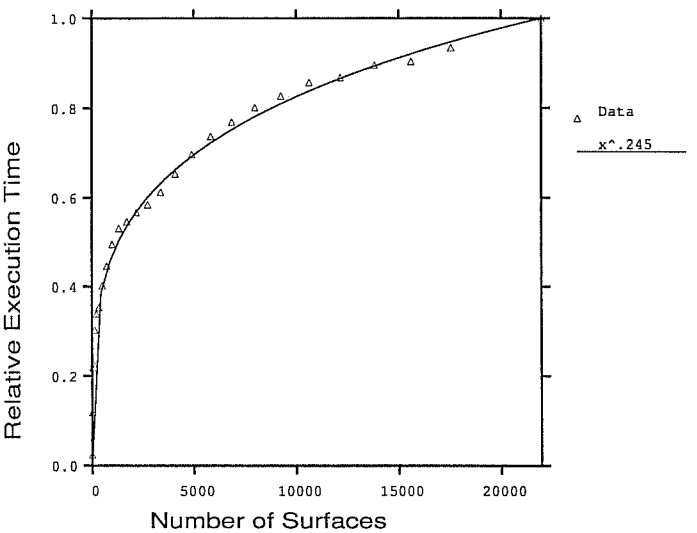


Figure 10. *Plot showing sublinear relationship of intersection time to number of surfaces in a scene. The best fit for γ in this test was 0.245, meaning the ray intersection time grew more slowly than the fourth root of N. The spheres were kept small enough so that a random ray sent from the field's interior had about a 50% chance of hitting something. (I.e. the sphere radii were proportional to $N^{1/3}$.) This guarantees that we are really seeing the cost of complicated geometry, since each ray goes by many surfaces.*

Although it is difficult to prove in general, our empirical tests show that the average cost of ray intersection using this technique grows as a fractional power of the total number of surfaces, i.e. $O(N^{\gamma})$ where $\gamma < \frac{1}{2}$. The time to create the octree grows linearly with the number of surfaces, but it is usually only a tiny fraction of the time spent rendering. Figure 10 shows the relationship between ray intersection time and number of surfaces for a uniformly distributed random field of spheres.

The basic surface primitives supported in *Radiance* are polygons, spheres and cones. Generator programs provide conversion from arbitrary shape definitions (e.g. surfaces of

revolution, prisms, height fields, parametric patches) to these basic types. Additional scene complexity is modeled using hierarchical *instancing*, similar to the method proposed by Snyder [21]. In our application of instancing, objects to be instanced are stored in a separate octree, then this octree is instanced along with other surfaces to create a second, enclosing octree. This process is repeated as many times and in as many layers as desired to produce the combined scene. It is possible to model scenes with a virtually unlimited number of surfaces using this method.

Figure 11 shows a cabin in a forest. We began with a simple spray of 150 needles, which were put into an octree and instanced many times and combined with twigs to form a branch, which was in turn instanced and combined with larger branches and a trunk to form a pine tree. This pine tree was then put in another octree and instanced in different sizes and orientations to make a small stand of trees, which was combined with a terrain and cabin model to make this scene. Thus, four hierarchical octrees were used together to create this scene, which contains over a million surfaces in all. Despite its complexity, the scene still renders in a couple of hours, and the total data structure takes less than 10 Mbytes of RAM.

3.7. Patterns and Textures

Another practical way to add detail to a scene is through the appropriate infusion of surface detail. In *Radiance,* we call a variation in surface color and/or brightness a *pattern,* and a perturbation of the surface normal a *texture.* This is more in keeping with the English definitions of these words, but sometimes at odds with the computer graphics community, which seems to prefer the term "texture" for a color variation and "bump-map" for a perturbation of the surface normal. In any case, we have extended the notion somewhat by allowing patterns and textures to be functions not only of surface position but also of surface normal and ray direction so that a pattern, for example, may also be used to represent a light source output distribution.

Our treatment of patterns and textures was inspired by Perlin's flexible shading language [17], to which we have added the mapping of lookup functions for multi-dimensional data. Using this technique, it is possible to interpret tabulated or image data in any manner desired through the same functional language used for procedural patterns and textures.

Figure 12 shows a scene with many patterns and textures. The textures on the vases and oranges and lemons are procedural, as is the pattern on the bowl. The pattern on the table is scanned, and the picture on the wall is obviously an earlier rendering. Other patterns which are less obvious in this scene are the ones applied to the three light sources, which define their output distributions. The geometry was created with the generator programs included with *Radiance,* which take functional specifications in the same language as the procedural patterns and textures. The star patterns are generated using a *Radiance* filter option that uses the pixel magnitude in deciding how much to spread the image, showing one advantage of using a floating-point picture format [27]. (The main advantage of this format is the ability to adjust exposure after rendering, taking full advantage of tone mapping operators and display calibration [23,30].)

3.8. Parallel Processing

One of the most practical ways to reduce calculation time is with parallel processing. Ray-tracing is a natural for parallel processing, since the calculation of each pixel is relatively independent. However, the caching of indirect irradiance values in *Radiance* means that we benefit from sharing information between pixels that may or may not be neighbors in one or more images. Sharing this information is critical to the efficiency of a parallel computation, and we want to do this in a system-independent way.

We have implemented a coarse-grained, multiple instruction, shared data (MISD) algorithm for *Radiance* rendering†. This technique may be applied to a single image, where multiple processes on one or more machines work on small sections of the image simultaneously, or to a sequence of images, where each process works on a single frame in a long animation. In the latter case, we need only worry about the sharing of indirect irradiance values on multiple active invocations, since dividing the image is not an issue. The method we use is described below.

Indirect irradiance values are written to a shared file whose contents are checked by each process prior to update. If the file has grown, the new values (produced by other processes) are read in before the irradiances computed by this process are written out. File consistency is maintained through the NFS lock manager, thus values may be shared transparently across the network. Irradiance values are written out in blocks big enough to avoid contention problems, but not so big that there is a lot of unnecessary computation caused by insufficient value sharing. We found this method to be much simpler, and about as efficient, as a remote procedure call (RPC) approach.

Since much of the scene information is static throughout the rendering process, it is wasteful to have multiple copies on a multi-processing platform that is capable of sharing memory. As with value sharing, we wished to implement memory sharing in a system-independent fashion. We decided to use the memory sharing properties of the UNIX *fork(2)* call. All systems capable of sharing memory do so during fork on a copy-on-write basis. Thus, a child process need not be concerned that it is sharing its parent's memory, since it will automatically get its own memory the moment it stores something. We can use this feature to our advantage by reading in our entire scene and initializing all the associated data structures before forking a process to run in parallel. So long as we do not alter any of this information during rendering, we will share the associated memory. Duplicate memory may still be required for data that is generated during rendering, but in most cases this represents a minor fraction of our memory requirements.

3.9. Animation

Radiance is often used to create walk-through animations of static environments. Though this is not typically the domain of ray-tracing renderers, we employ some techniques to make the process more efficient. The most important technique is the use of recorded depth information at each pixel to interpolate fully ray-traced frames with a z-buffer algorithm. Our method is similar to the one explained by Chen et al [3], where pixel depths are used to recover an approximate 3-dimensional model of the visible portions of the scene, and a z-buffer is used to make visibility decisions for each intermediate view. This makes it possible to generate 30 very good-looking frames for each second of animation while only having to render about 5 of them. Another technique we use is unique to *Radiance,* which is the sharing of indirect irradiance values. Since these values are view-independent, there is no sense in recomputing them each time, and sharing them during the animation process distributes the cost over so many frames that the incremental cost of simulating diffuse interreflection is negligible.

Finally, it is possible to get interactive frame rates from advanced rendering hardware using illumination maps instead of ray-tracing the frames directly. (An illumination map is a 2-dimensional array of color values that defines the surface shading.) Such maps may be kept separate from the surfaces' own patterns and textures, then combined during rendering. Specular

†Data sharing is of course limited in the case of distributed processors, where each node must have its own local copy of scene data structures.

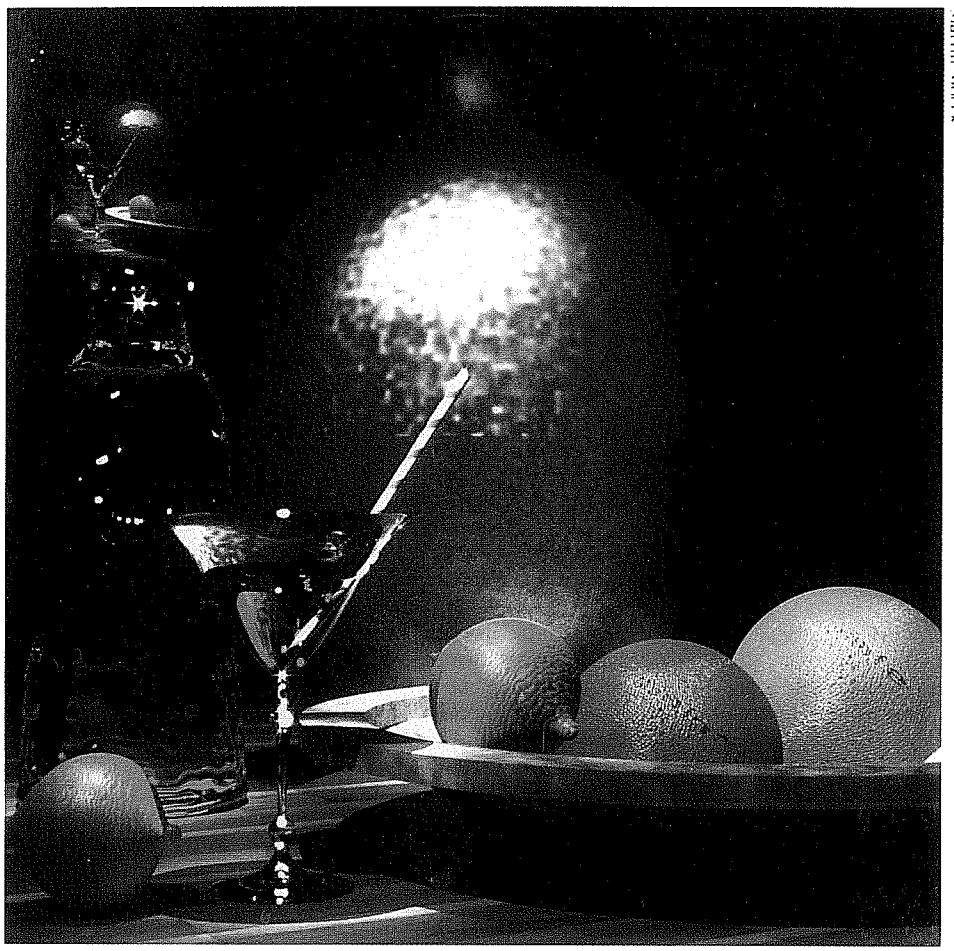
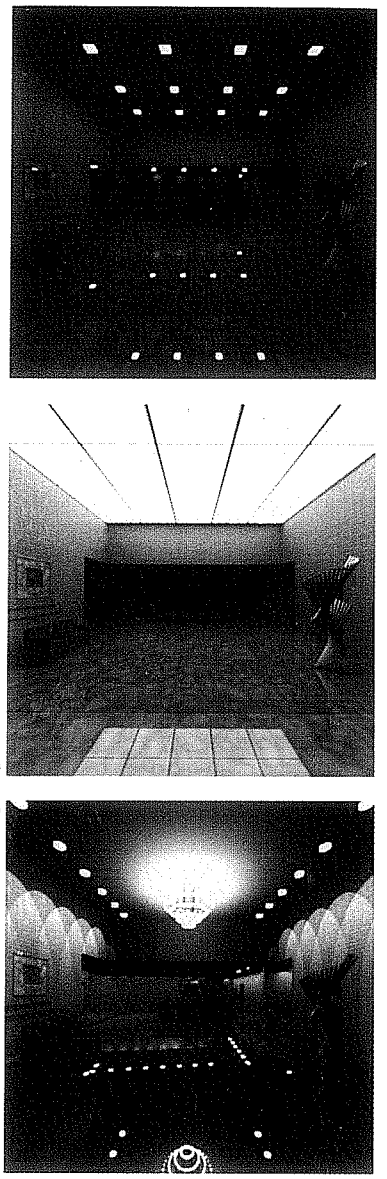
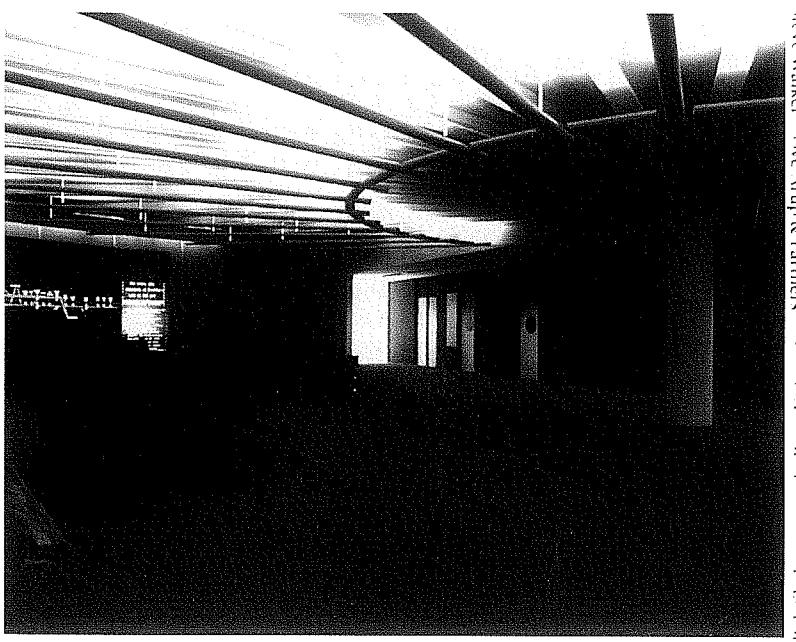


Figure 12. A still life image showing examples of procedural and scanned textures and patterns.



Martin Moeck Siemens Lighting

Figure 13. A comparison of three lighting schemes for a hotel lobby space.



Steve Walker Ove Arup & Partners Rob Shakespeare Indiana University

Figure 14. An indirect lighting system was designed to reduce glare on monitors in the London Underground control center.

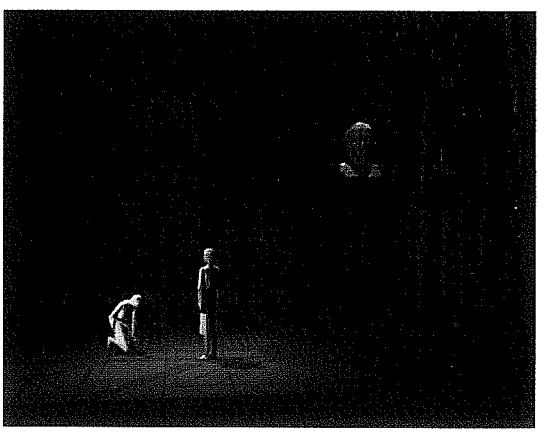


Figure 15. Stage lighting simulation.

RADIANCE File Types			
Data Type	Format	Created by	Used for
Scene Description	ASCII text	text editor, CAD translator	geometry, materials, patterns, textures
Function File	ASCII text	text editor	surface tessellation, patterns, textures, scattering functions, coordinate mappings, data manipulation
Data File	ASCII integers and floats	luminaire data translator, text editor	N-dimensional patterns, textures, scattering functions
Polygonal Font	ASCII integers	Hershey set, font design system, font translator, text editor	text patterns, label generator
Octree	Binary	scene compiler (**oconv**)	fast ray intersection, incremental scene compilation, object instancing
Picture	run-length encoded 4-byte/pixel floating-point	renderer, filter, image translator	interactive display, hard copy, lighting analysis, material pattern, rendering recovery
Ambient File	Binary	renderer, point value program	sharing view-independent indirect irradiance values

Table 1. *All binary types in Radiance are portable between systems, and have a standard information header specifying the format and the originating command(s).*

surfaces will not appear correct since they depend on the viewer's perspective, but this may be a necessary sacrifice when user control of the walk-through is desired. Interactive rendering has long been touted as a principal advantage of radiosity, when in fact complete view-independence is primarily a side-effect of assuming diffuse reflection. *Radiance* calculates the same values using a ray-tracing technique, and storage and rendering may even be more efficient since large polygons need not be subdivided into hundreds of little ones -- an illumination map works just as well or better.

3.10. Implementation Issues

Radiance is a collection of C programs designed to work in concert, communicating via the standard data types listed in Table 1. The system may be compiled directly on most UNIX platforms, including SGI, Sun, HP, DEC, Apple (A/UX), and IBM (RS/6000). Portability is maintained over 60,000+ lines of code using the Kernighan and Ritchie standard [11] and conservative programming practices that do not rely on system-specific behaviors or libraries. (In addition to UNIX support, there is a fairly complete Amiga port by Per Bojsen, and a limited MS-DOS port by Karl Grau.)

A typical rendering session might begin with the user creating or modifying a geometric model of the space using a CAD program. (The user spends about 90% of the time on geometric modeling.) The CAD model is then translated into a *Radiance* scene description file, using either a stand-alone program or a function within the CAD system itself. The user might then create or modify the materials, patterns and textures associated with this model, and add some objects from a library of predefined light sources and furnishings. The completed model would then be compiled by **oconv** into an octree file, which would be passed to the interactive renderer, **rview,** to verify the desired view and calculation parameters. Finally, a batch rendering would be started with **rpict,** and after a few minutes or a few hours, the raw picture would be filtered (i.e. anti-aliased via image reduction) by **pfilt** using a suitable exposure level and target resolution. This finished picture may be displayed with **ximage,** translated to another format, printed, or further analyzed using one of the many *Radiance* image utilities. This illustrates the basic sequence of:

model → convert → render → filter → display

all of which may be put in a single pipelined command if desired.

As *Radiance* has evolved over the years, it has become increasingly sophisticated, with nearly 100 programs that do everything from CAD translation to surface tessellation to lighting calculations and rendering to image filtering, composition and conversion. With this sophistication comes great versatility, but learning the ins and outs of the programs, even the few needed for simple rendering, is impractical for most designers.

To overcome system complexity and improve the reliability of rendering results, we have written an executive control program, called **rad.** This program takes as its input a single file that identifies the material and scene description files needed as well as qualitative settings related to this environment and the simulation desired. The control program then calls the other programs with the proper parameters in the proper sequence.

The intricacies of the *Radiance* rendering pipeline are thus replaced by a few intuitive variable settings. For example, there is a variable called "DETAIL", which might be set to "low" for an empty room, "medium" for a room with a few pieces of furniture and "high" for a complicated room with many furnishings and textures. This variable will be used with a few others like it to determine how many rays to send out in the Monte Carlo sampling of indirect lighting, how closely to space these values, how densely to sample the image plane, and so on. One very important variable that affects nearly all rendering parameters is called "QUALITY". Low quality renderings come out quickly but may not look as good as medium quality renderings, and high quality renderings take a while but when they finish, the images can go straight into a magazine article.

This notion of replacing many algorithm-linked rendering parameters with a few qualitative variables has greatly improved the usability of *Radiance* and the reliability of its output. The control program also keeps track of octree creation, secondary source generation, aborted job recovery, image filtering and anti-aliasing, and running the interactive renderer. The encoding of expertise in this program has been so successful, in fact, that we rely on it ourselves almost 100% for setting parameters and controlling the rendering process.

Although the addition of a control program is a big improvement, there are still many aspects of *Radiance* that are not easily accessible to the average user. We have therefore added a number of utility scripts for performing specific tasks from the more general functions that constitute the system. One example of this is the **falsecolor** program, which calls other image filter programs and utilities to generate an image showing luminance contours or other data associated with a scene or

rendering. Figure 9c shows our previous rendering (Figure 9a) superimposed with illuminance contours. These contours tell the lighting designer if there is enough light in the right places or too much light in the wrong places -- information that is difficult to determine from a normal image†.

Even with a competent rendering control program and utility scripts for accomplishing specific tasks, there are still many°designers who would not want to touch this system with an extended keyboard. Modern computer users expect a list of pull-down menus with point-and-click options that reduce the problem to a reasonably small and understandable set of alternatives. We are currently working on a graphical user interface (GUI) to the **rad** control program, which would at least put a friendlier face on the standard system. A more effective long-term solution is to customize the rendering interface for each problem domain, e.g. interior lighting design, daylighting, art, etc. Due to our limited resources and expertise, we have left this customization task to third parties who know more about specific applications, and who stand to benefit from putting their GUI on our simulation engine. So far, there are a half dozen or so developers working on interfaces to *Radiance.*

4. Applications and Results

The real proof of a physically-based rendering system is the problems it solves. Here we see how well we have met the challenges and goals we set out. *Radiance* has been used by hundreds of people to solve thousands of problems over the years. In the color pages we have included some of the more recent work of some of the more skilled users. The results have been grouped into two application areas, electric lighting problems and daylighting problems.

4.1. Electric Lighting

Electric lighting was the first domain of *Radiance,* and it continues to be a major strength. A model may contain any number of light sources of all shapes and sizes, and the output distributions may be entered as either near-field or far-field data. The dual nature of light sources (mentioned in section 3.5) also permits detailed modeling of fixture geometry, which is often important in making aesthetic decisions.

There are several application areas where electric lighting is emphasized. The most obvious application is lighting design. Figure 13 shows a comparative study between three possible lighting alternatives in a hotel lobby space. Several other designs were examined in this exploration of design visualization. With such a presentation, the final decision could be safely left to the client.

One design application that requires very careful analysis is indirect lighting. Figure 14 shows a simulation of a new control center for the London Underground. The unusual arrangement of upwardly directed linear fluorescents was designed to provide general lighting without affecting the visibility of the central display panel (image left).

Stage lighting is another good application of physically-based rendering. The designs tend to be complex and changing, and the results must be evaluated aesthetically (i.e. visually). Figure 15 shows a simulation of a scene from the play *Julius Caesar*. Note the complex shadows cast by the many struts in the stage set. Computing these shadows with a radiosity algorithm would be extremely difficult.

†Actually, *Radiance* pictures do contain physical values through a combination of the 4-byte floating-point pixel representation and careful tracking of exposure changes [27], but the fidelity of any physical image presentation is limited by display technology and viewing conditions. We therefore provide the convenience of extracting numerical values with our interactive display program.

4.2. Daylighting

Daylight poses a serious challenge to physically-based rendering. It is brilliant, ever-changing and ever-present. At first, the daylight simulation capabilities in *Radiance* were modest, limited mostly to exteriors and interiors with clear windows or openings. Designers, especially architects, wanted more. They wanted to be able to simulate light through venetian blinds, intricate building facades and skylights. In 1991, the author was hired on sabbatical by EPFL to improve the daylight simulation capabilities of *Radiance,* and developed some of the techniques described earlier in this paper. In particular, the large source adaptive subdivision, virtual source and secondary source calculations proved very important for daylighting problems.

The simplest application of daylight is exterior modeling. Many CAD systems have built-in renderers that will compute the solar position from time of day, year, and location, and generate accurate shadows. In addition to this functionality, we wanted *Radiance* to show the contributions of diffuse skylight and interreflection. Figure 16 shows the exterior of the Mellencamp Pavillion, an Indiana University project that recently acquired funding (along with its name).

A more difficult daylighting problem is atrium design*. Designing an atrium requires thorough understanding of the daylight availability in a particular region to succeed. Figure 17 shows an atrium space modeled entirely within *Radiance,* without the aid of a CAD program [13]. The hierarchical construction of *Radiance* scene files and the many programmable object generators makes text-editor modeling possible, but most users prefer a "mousier" approach.

Daylighted interiors pose one of the nastiest challenges in rendering. Because sunlight is so intense, it is usually diffused or baffled by louvers or other redirecting systems. Some of these systems can be quite elaborate, emphasizing the need for simulation in their design. Figure 18 shows the interior of the pavillion from Figure 16. Figure 19 shows a library room illuminated by a central skylight. Figure 20a shows a simulation of a daylighted museum interior. Daylight is often preferred in museums as it provides the most natural color balance for viewing paintings, but control is also very important. Figure 20b shows a false color image of the illuminance values on room surfaces; it is critical to keep these values below a certain threshold to minimize damage to the artwork.

5. Conclusion

We have presented a physically-based rendering system that is accurate enough, general enough, and practical enough for the vast majority of lighting design and architectural applications. The simulation uses a light-backwards ray-tracing method with extensions to handle specular, diffuse and directional-diffuse reflection and transmission in any combination to any level in any environment. Not currently included in the calculation are participating media, diffraction and interference, phosphorescence, and polarization effects. There is nothing fundamental preventing us from modeling these processes, but so far there has been little demand for them from our users.

The principle users of *Radiance* are researchers and educators in public and private institutions, and lighting specialists at large architectural, engineering and manufacturing firms. There are between 100 and 200 active users in the U.S. and Canada, and about half as many overseas. This community is continually growing, and as the Radiance interface and documentation improves, the growth rate is likely to increase.

*An atrium is an enclosed courtyard with a glazed roof structure for maximizing daylight while controlling the indoor climate.

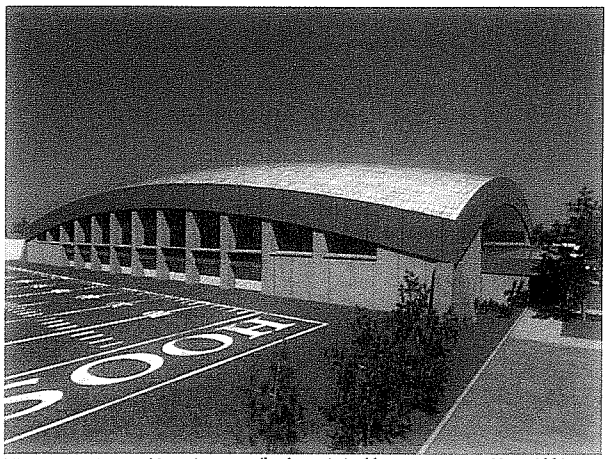


Figure 16. Design of the Mellencamp Pavillion, currently under construction at Indiana University.

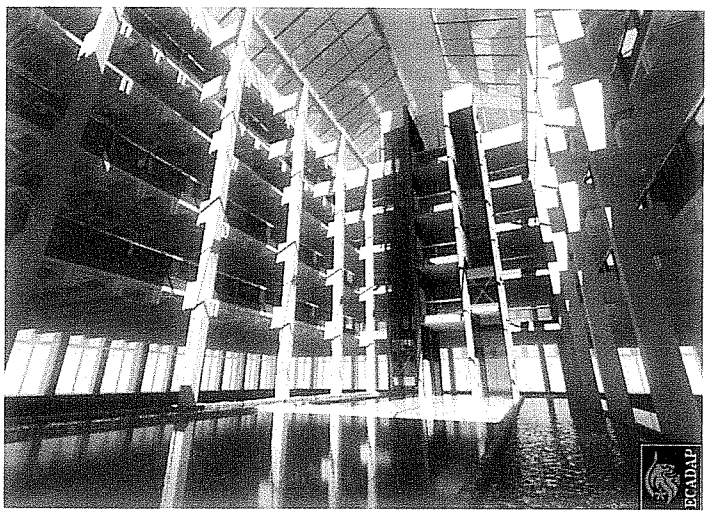


Figure 17. U.K. Atrium design modeled without the benefit of a CAD system, using only Radiance scene generation utilities.

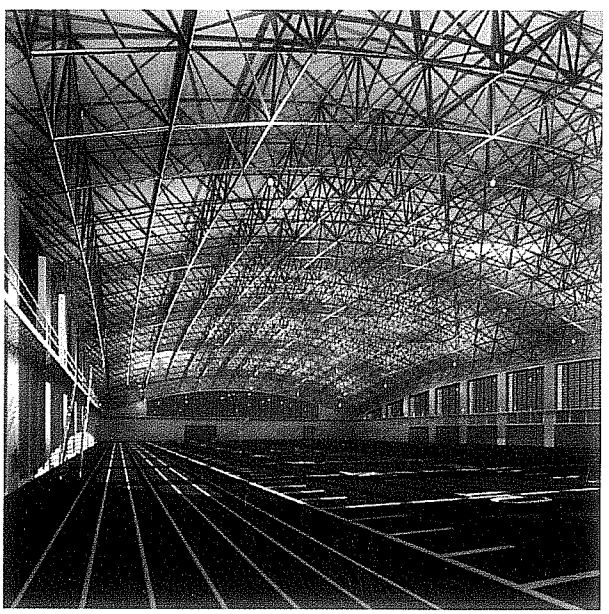


Figure 18. Interior view of Mellencamp Pavillion, above.

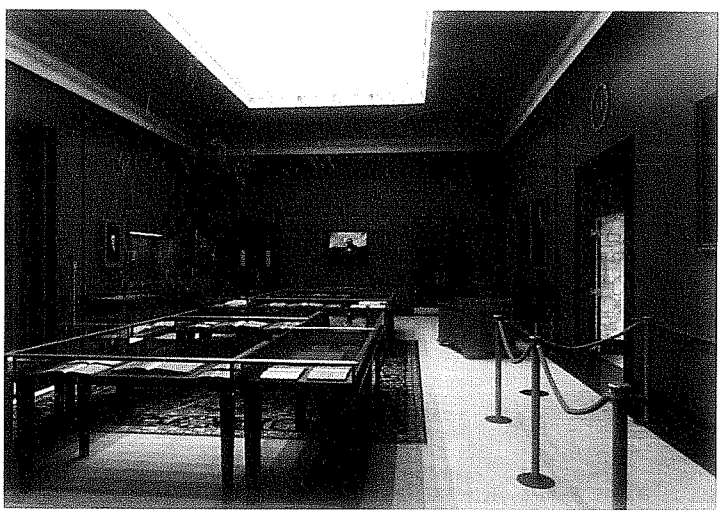


Figure 19. Indiana University library space, illuminated by a central skylight.

Figure 20a. Art gallery illuminated by skylights.

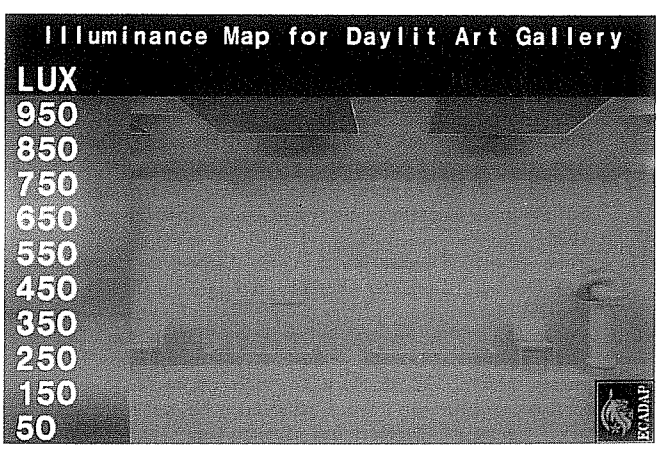


Figure 20b. Corresponding surface illuminances.

For the graphics research community, we hope that *Radiance* will provide a basis for evaluating new physically-based rendering techniques. To this end, we provide both the software source code and a set of precomputed test cases on our ftp server. The test suite includes diffuse and specular surfaces configured in a simple rectangular space with and without obstructions. More complicated models are also provided in object libraries and complete scene descriptions.

6. Acknowledgements

Most of the color figures in this paper represent the independent work of *Radiance* users, and are reprinted with permission. Figure 5 was created by Charles Ehrlich from a design by Mark Mack Architects of San Francisco. Figure 11 was created by Cindy Larson. Figures 12 and 13 were created by Martin Moeck of Siemens Lighting, Germany. Figure 14 was created by Steve Walker of Ove Arup and Partners, London. Figure 15 was created by Robert Shakespeare of the Theatre Computer Visualization Center at Indiana University. Figures 16, 18 and 19 were created by Scott Routen and Reuben McFarland of the University Architect's Office at Indiana University. Figures 17 and 20 were created by John Mardaljevic of the ECADAP Group at De Montfort University, Leicester.

The individuals who have contributed to *Radiance* through their support, suggestions, testing and enhancements are too numerous to mention. Nevertheless, I must offer my special thanks to: Peter Apian-Bennewitz, Paul Bourke, Raphael Compagnon, Simon Crone, Charles Ehrlich, Jon Hand, Paul Heckbert, Cindy Larson, Daniel Lucias, John Mardaljevic, Kevin Matthews, Don McLean, Georg Mischler, Holly Rushmeier, Jean-Louis Scartezzini, Jennifer Schuman, Veronika Summeraur, Philip Thompson, Ken Turkowski, and Florian Wenz.

Work on Radiance was sponsored by the Assistant Secretary for Conservation and Renewable Energy, Office of Building Technologies, Buildings Equipment Division of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. Additional funding was provided by the Swiss federal government as part of the LUMEN Project.

7. Software Availability

Radiance is available by anonymous ftp from two official sites:

hobbes.lbl.gov	128.3.12.38	Berkeley, California
nestor.epfl.ch	128.178.139.3	Lausanne, Switzerland

For convenience, *Radiance 2.4* has been included on the CD-ROM version of these proceedings.

From **Mosaic,** try the following URL:

file://hobbes.lbl.gov/www/radiance/radiance.html

8. References

[1] Baum, Daniel, Holly Rushmeier, James Winget, "Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors," *Computer Graphics*, Vol. 23, No. 3, July 1989, pp. 325-334.

[2] Baum, Daniel, Stephen Mann, Kevin Smith, James Winget, "Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accurate Radiosity Solutions," *Computer Graphics*, Vol. 25, No. 4, July 1991.

[3] Chen, Shenchang Eric, Lance Williams, "View Interpolation for Image Synthesis," *Computer Graphics*, August 1993, pp. 279-288.

[4] Compagnon, Raphael, B. Paule, J.-L. Scartezzini, "Design of New Daylighting Systems Using ADELINE Software," *Solar Energy in Architecture and Urban Planning*, proceedings of the 3rd European Conference on Architecture, Florence, Italy, May 1993.

[5] Cook, Robert, Thomas Porter, Loren Carpenter, "Distributed Ray Tracing," *Computer Graphics*, Vol. 18, No. 3, July 1984, pp. 137-147.

[6] Dorsey, Julie O'B., Francois Sillion, Donald Greenberg, "Design and Simulation of Opera Lighting and Projection Effects," *Computer Graphics*, Vol. 25, No. 4, July 1991, pp. 41-50.

[7] Glassner, Andrew S., "Space subdivision for fast ray tracing" *IEEE Computer Graphics and Applications* Vol. 4, No. 10, October 1984, pp. 15-22.

[8] Goral, Cindy, Kenneth Torrance, Donald Greenberg, Bennet Battaile, "Modeling the Interaction of Light Between Diffuse Surfaces," *Computer Graphics*, Vol. 18, No. 3, July 1984, pp. 213-222.

[9] Grynberg, Anat, *Validation of Radiance*, LBID 1575, LBL Technical Information Department, Lawrence Berkeley Laboratory, Berkeley, California, July 1989.

[10] Kajiya, James T., "The Rendering Equation," *Computer Graphics*, Vol. 20, No. 4, August 1986.

[11] Kernighan, Brian, Dennis Ritchie, *The C Programming Language*, Prentice-Hall, 1978.

[12] Kirk, David, James Arvo, "Unbiased Sampling Techniques for Image Synthesis," *Computer Graphics*, Vol 25, No. 4, July 1991, pp. 153-156.

[13] Mardaljevic, John and Kevin Lomas, "Creating the Right Image," *Building Services / The CIBSE Journal*, Vol 15, No. 7, July 1993, pp. 28-30.

[14] Mardaljevic, John, K.J. Lomas, D.G. Henderson, "Advanced Daylighting Design for Complex Spaces" *Proceedings of CLIMA 2000*, 1-3 November 1993, London UK.

[15] Meyer, Gary, Holly Rushmeier, Michael Cohen, Donald Greenberg, Kenneth Torrance, "An Experimental Evaluation of Computer Graphics Imagery," *ACM Transactions on Graphics*, Vol. 5, No. 1, pp. 30-50.

[16] Nicodemus, F.E., J.C. Richmond, J.J. Hsia, *Geometrical Considerations and Nomenclature for Reflectance*, U.S. Department of Commerce, National Bureau of Standards, October 1977.

[17] Perlin, Ken, "An Image Synthesizer", *Computer Graphics*, Vol. 19, No. 3, July 1985, pp. 287-296.

[18] Rushmeier, Holly, *Extending the Radiosity Method to Transmitting and Specularly Reflecting Surfaces*, Master's Thesis, Cornell Univ., Ithaca, NY, 1986.

[19] Rushmeier, Holly, Charles Patterson, Aravindan Veerasamy, "Geometric Simplification for Indirect Illumination Calculations," *Proceedings of Graphics Interface '93*, May 1993, pp. 227-236.

[20] Smits, Brian, James Arvo, David Salesin, "An Importance-Driven Radiosity Algorithm," *Computer Graphics*, Vol 26, No. 2, July 1992, pp. 273-282.

[21] Snyder, John M., Alan H. Barr, "Ray Tracing Complex Models Containing Surface Tessellations," *Computer Graphics* Vol. 21, No. 4, pp. 119-128, July 1987.

[22] Teller, Seth and Pat Hanrahan, "Global Visibility Algorithms for Illumination Computations," *Computer Graphics*, pp. 239-246, August 1993.

[23] Tumblin, Jack, Holly Rushmeier, "Tone Reproduction for Realistic Images," *IEEE Computer Graphics and Applications*, Vol. 13, No. 6, November 1993, pp. 42-48.

[24] Wallace, John, Michael Cohen, Donald Greenberg, "A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods," *Computer Graphics*, Vol. 21, No. 4, July 1987.

[25] Ward, Gregory, Francis Rubinstein, Robert Clear, "A Ray Tracing Solution for Diffuse Interreflection," *Computer Graphics*, Vol. 22, No. 4, August 1988.

[26] Ward, Gregory, "Adaptive Shadow Testing for Ray Tracing," *Second EUROGRAPHICS Workshop on Rendering*, Barcelona, Spain, April 1991.

[27] Ward, Gregory, "Real Pixels," *Graphics Gems II*, Edited by James Arvo, Academic Press 1991, pp. 80-83.

[28] Ward, Gregory, Paul Heckbert, "Irradiance Gradients," *Third EUROGRAPHICS Workshop on Rendering*, Bristol, United Kingdom, May 1992.

[29] Ward, Gregory, "Measuring and Modeling Anisotropic Reflection," *Computer Graphics*, Vol. 26, No. 2, July 1992, pp. 265-272.

[30] Ward, Gregory, "A Contrast-Based Scalefactor for Luminance Display," *Graphics Gems IV*, Edited by Paul Heckbert, Academic Press, 1994.

[31] Whitted, Turner, "An Improved Illumination Model for Shaded Display," *Communications of the ACM*, Vol. 23, No. 6, June 1980, pp. 343-349.