Adaptive Radiosity Textures for Bidirectional Ray Tracing

Paul S. Heckbert

Dept. of Electrical Engineering and Computer Science
University of California, Berkeley, CA 94720

Abstract

We present a rendering method designed to provide accurate, general simulation of global illumination for realistic image synthesis. Separating surface interaction into diffuse plus specular, we compute the specular component on the fly, as in ray tracing, and store the diffuse component (the radiosity) for later-reuse, similar to a radiosity algorithm. Radiosities are stored in *adaptive radiosity textures* (*rexes*) that record the pattern of light and shadow on every diffuse surface in the scene. They adaptively subdivide themselves to the appropriate level of detail for the picture being made, resolving sharp shadow edges automatically.

We use a three-pass, bidirectional ray tracing algorithm that traces rays from both the lights and the eye. The "size pass" records visibility information on diffuse surfaces; the "light pass" progressively traces rays from lights and bright surfaces to deposit photons on diffuse surfaces to construct the radiosity textures; and the "eye pass" traces rays from the eye, collecting light from diffuse surfaces to make a picture.

CR Categories: I.3.3 [**Computer Graphics**]: Picture/Image Generation - *display algorithms;* I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism - *visible line/surface algorithms.*

General Terms: algorithms.

Additional Key Words and Phrases: global illumination, density estimation, texture mapping, quadtree, adaptive subdivision, sampling.

1 Introduction

The presentation is divided into four sections. We first discuss previous work on the global illumination problem. Then we outline our bidirectional ray tracing approach in an intuitive way. Next we describe our implementation and some early results. We conclude with a summary of the method and of our experiences to date.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

2 The Global Illumination Problem

The primary goal of realistic image synthesis is the development of methods for modeling and realistically rendering three-dimensional scenes. One of the most challenging tasks of realistic image synthesis is the accurate and efficient simulation of *global illumination* effects: the illumination of surfaces in a scene by other surfaces. Early rendering programs treated the visibility (hidden surface) and shading tasks independently, employing a *local illumination* model which assumed that the shading of each surface is independent of the shading of every other surface. Local illumination typically assumes that light comes from a finite set of point light sources only. Global illumination models, on the other hand, recognize that the visibility and shading are interrelated: the shade of a surface point is determined by the shades of all of the surfaces visible from that point.

The intensity of light traveling in a given outgoing direction $\vec{out}$ from a surface point is the integral of the incident intensity times the *bidirectional distribution function* (BDF) over all possible incoming directions $\vec{in}$:

$$\text{intensity}(\vec{out}) = \int_{sphere} \text{intensity}(\vec{in})\, \text{BDF}(\vec{in}, \vec{out})\, d(\vec{in})$$

The bidirectional distribution function is the fraction of energy reflected or transmitted from the incoming direction $\vec{in} = (\phi_i, \theta_i)$ to the outgoing direction $\vec{out} = (\phi_o, \theta_o)$; it is the sum of the bidirectional reflectance distribution function (BRDF) and the bidirectional transmittance distribution function (BTDF). See [Hall89] for a more detailed discussion of the physics of illumination. We will characterize previous global illumination algorithms by the approximations they make to the above integral.

Because of the superposition properties of electromagnetic radiation, we can segregate surface reflectance into two types: diffuse and specular. We define *diffuse interaction* (both reflection and transmission) to be the portion of interaction that scatters light equally in all directions, and *specular interaction* to be the remaining portion. $\text{BDF} = \text{BDF}_{diff} + \text{BDF}_{spec}$. For many materials, specular interaction scatters light in only a small cone of directions. When this cone includes just a finite number of directions, each a cone with solid angle zero, we call the interaction *ideal specular*, otherwise, when the cone(s) have a positive finite angle, we call it *rough specular*. Two examples of ideal specular surfaces are: (1) a perfect mirror that reflects in one direction, and (2) a perfect transmitter that refracts in one direction and reflects in another. An ideal specular surface with micro-bumps behaves statistically like a rough specular surface. Our three classes of interaction are diagrammed in figure 1.

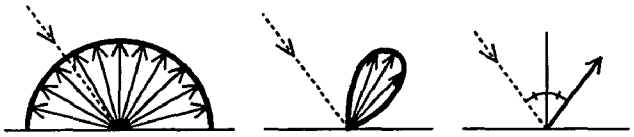

Figure 1: *Three classes of reflectance: diffuse, rough specular, and ideal specular; showing a polar plot of the reflectance coefficient for fixed incoming direction and varying outgoing direction. Transmittance is similar.*

A diffuse surface appears equally bright from all viewing directions, but a specular surface's brightness varies with viewing direction, so we say that diffuse interaction is *view-independent* while specular interaction is *view-dependent*. The simplest materials have a position-invariant, isotropic BDF consisting of a linear combination of diffuse and ideal specular interaction, but a fully-general BDF can simulate textured, anisotropic, diffuse and rough specular surfaces.

2.1 Ray Tracing vs. Radiosity

The two most popular algorithms for global illumination are ray tracing and radiosity. Ray tracing is both a visibility algorithm and a shading algorithm, but radiosity is just a shading algorithm.

2.1.1 Ray Tracing

Classic ray tracing generates a picture by tracing rays from the eye into the scene, recursively exploring specularly reflected and transmitted directions, and tracing rays toward point light sources to simulate shadowing [Whitted80]. It assumes that the BDF contains no rough specular, and that the incident light relevant to the diffuse computation is a sum of delta functions in the direction of each light source. This latter assumption implies a local illumination model for diffuse.

A more realistic illumination model includes rough specular BDF's and computes diffuse interaction globally. Exact simulation of these effects requires the integration of incident light over cones of finite solid angle. Ray tracing can be generalized to approximate such computations using *distribution ray tracing* [Cook84], [Lee85], [Dippe85], [Cook86], [Kajiya86]. (We propose the name "distribution ray tracing" as an alternative to the current name, "distributed ray tracing", which is confusing because of its parallel hardware connotations.) In distribution ray tracing, rays are distributed, either uniformly or stochastically, throughout any distributions needing integration. Many rays must be traced to accurately integrate the broad reflectance distributions of rough specular and diffuse surfaces: often hundreds or thousands per surface intersection.

2.1.2 Radiosity

The term *radiosity* is used in two senses. First, radiosity is a physical quantity equal to power per unit area, which determines the intensity of light diffusely reflected by a surface, and second, radiosity is a shading algorithm. The meaning of each use should be clear by context.

The *Classic radiosity* algorithm subdivides each surface into polygons and determines the fraction of energy diffusely radiated from each polygon to every other polygon: the pair's *form factor*. From the form factors, a large system of equations is constructed whose solution is the radiosities of each polygon [Siegel81], [Goral84], [Nishita85]. This system can be solved either with Gauss-Seidel iteration or, most conveniently, with progressive techniques that compute the matrix and solve the system a piece at a time [Cohen88]. Form factors can be determined analytically for simple geometries [Siegel81], [Baum89], but for complex geometries a numerical approach employing a visibility algorithm is necessary. The most popular visibility method for this purpose is a *hemicube* computed using a z-buffer [Cohen85], but ray tracing has recently been promoted as an alternative [Wallace89], [Sillion89]. Classic radiosity assumes an entirely diffuse reflectance, so it does not simulate specular interaction at all.

The output of the radiosity algorithm is one radiosity value per polygon. Since diffuse interaction is by definition view-independent, these radiosities are valid from any viewpoint. The radiosity computation must be followed by a visibility algorithm to generate a picture.

The radiosity method can be generalized to simulate specular interaction by storing not just a single radiosity value with each polygon, but a two-dimensional array [Immel86], [Shao88], [Buckalew89]. The resulting algorithm, which we call *directional radiosity*, simulates both diffuse and specular interaction globally, but the memory requirements are so excessive as to be impractical.

2.1.3 Hybrid Methods

Ray tracing is best at specular and radiosity is best at diffuse, and the above attempts to generalize ray tracing to diffuse and to generalize radiosity to specular stretch the algorithms beyond the reflectance realms for which each is best suited, making them less accurate and less efficient. Another class of algorithms is formed by hybridizing the methods, using a two-pass algorithm that applies a radiosity pass followed by the ray tracing pass. This is the approach used by [Wallace87] and [Sillion89].

The first pass of Wallace's algorithm consists of classic radiosity extended to include diffuse-to-diffuse interactions that bounce off planar mirrors. He follows this with a classic ray tracing pass (implemented using a z-buffer). Unfortunately, the method is limited to planar surfaces (because of the polygonization involved in the radiosity algorithm) and to perfect planar mirrors.

Sillion's algorithm is like Wallace's but it computes its form factors using ray tracing instead of hemicubes. This eliminates the restriction to planar mirrors. The method still suffers from the polygonization inherent in the radiosity step, however.

2.2 Sampling Radiosities

Many of the sampling problems of ray tracing have been solved by recent adaptive algorithms [Whitted80], [Cook86], [Lee85], [Dippe85], [Mitchell87], [Painter89], particularly for the simulation of specular interaction. The sampling problems of the radiosity algorithm are less well studied, probably because its sampling process is less explicit than that of ray tracing.

We examine four data structures for storing radiosities: light images, polygons, samples in 3-D, and textures. Several different algorithms have been used to generate these data structures: radiosities have been generated analytically, with hemicubes at the receiver (gathering), with hemicubes at the sender (shooting), and by tracing rays from the eye or from the light.

2.2.1 Light Images

The simplest data structure, the *light image*, simulates only shadows, the first order effects of diffuse interreflection. Light images are pictures of the scene from the point of view of each light source. They are most often generated using the z-buffer shadow algorithm, which saves the z-buffers of these light images and uses them while rendering from the point of view of the eye to test if visible points are in shadow [Williams78], [Reeves87]. This shadow algorithm is more flexible than most, since it is not limited to polygons, but it is difficult to tune. Choosing the resolution for the light images is critical, since aliasing of shadow edges results if the light images are too coarse.

2.2.2 Polygonized Radiosity

The Atherton-Weiler algorithm is another method for computing shadows that renders from the point of view of the lights [Atherton78]. It uses the images rendered from the lights to generate "surface detail polygons", modifying the scene description by splitting all polygons into shadowed and unshadowed portions that are shaded appropriately in the final rendering from the eye. Surface detail polygons are an example of *polygonized radiosity*, the storage of radiosity as polygons. The shadows computed by the Atherton-Weiler algorithm are a first-approximation to the interreflection simulated by radiosity algorithms.

The most common method for computing polygonized radiosity is, of course, the classic radiosity algorithm. A major problem with this algorithm is that surfaces are polygonized before radiosities are computed. Difficulties result if this polygonization is either too coarse or too fine.

Sharp shadow edges caused by small light sources can be undersampled if the polygonization is too coarse, resulting in blurring or aliasing of the radiosities. Cohen developed the "substructuring" technique in response to this problem [Cohen86]. It makes an initial pass computing radiosities at low resolution, then splits polygons that appear to be in high-variance regions and recomputes radiosities. Substructuring helps, but it is not fully automatic, as the subdivision stopping criterion appears to be a polygon size selected in some ad hoc manner. The limitations of the method are further demonstrated by the absence to date of radiosity pictures in published work exhibiting sharp shadow edges.

The other extreme of radiosity problems is oversampling of radiosities due to polygonization that is too fine for the hemicube. The resulting quantization can be cured by adaptive subdivision of the hemicube or of the light rays [Wallace89], [Baum89].

We conclude that polygonization criteria remain a difficult problem for the radiosity method.

It is interesting to note the similarities between radiosity algorithms and the Atherton-Weiler algorithm. Conceptually, the original radiosity method gathers light to each polygon by rendering the scene from the point of view of each receiver, but the progressive radiosity algorithm shoots light by rendering the scene from the point of view of each sender (a light source). A progressive radiosity algorithm using a hemicube is thus much like repeated application of the Atherton-Weiler shadow algorithm.

2.2.3 Samples in 3-D

Radiosities can be computed using brute force distribution ray tracing [Kajiya86], but the method is inefficient because it samples the slowly-varying radiosity function densely. To exploit the coherence of radiosity values, Ward sampled the diffuse component sparsely, and saved this information in a world space octree [Ward88]. Because his algorithm shot rays from the eye toward the lights, and not vice-versa, it had difficulty detecting light sources reflected by specular surfaces.

2.2.4 Radiosity Texture

The fourth data structure for radiosities is the *radiosity texture*. Instead of polygonizing each surface and storing one radiosity value per polygon, radiosity samples are stored in a texture on every diffuse surface in the scene [Arvo86]. Arvo called his textures "illumination maps". He computed them by tracing rays from the light sources.

2.3 Light Ray Tracing

Rays traced from the eye we call *eye rays* and rays traced from the lights we call *light rays*. We avoid the terms "forward ray tracing" and "backward ray tracing" because they are ambiguous: some people consider photon motion "forward", while others consider Whitted's rays "forward".

Light ray tracing was originally proposed by Appel [Appel68], who "stored" his radiosities on paper with a plotter. Light ray tracing was proposed for beams in previous work with Hanrahan [Heckbert84] where we stored radiosities as surface detail polygons like Atherton-Weiler. This approach was modified by Strauss, who deposited light directly in screen pixels when a diffuse surface was hit by a beam, rather than store the radiosities with the surface [Strauss88]. Watt has recently implemented light beam tracing to simulate refraction at water surfaces [Watt90]. Arvo used light ray tracing to compute his radiosity textures [Arvo86]. Light ray tracing is often discussed but has been little used, to date.

3 Bidirectional Ray Tracing Using Adaptive Radiosity Textures

In quest of realistic image synthesis, we seek efficient algorithms for simulating global illumination that can accommodate curved surfaces, complex scenes, and arbitrary surface characteristics (BDF's), and generate pictures perceptually indistinguishable from reality. These goals are not realizable at present, but we can make progress if we relax our requirements.

We make the following assumptions:

(1) Only surfaces are relevant. The scattering or absorption of volumes can be ignored.

(2) Curved surfaces are important. The world is not polygonal.

(3) Shadows, penumbras, texture, diffuse interreflection, specular reflection, and refraction are all important.

(4) We can ignore the phenomena of fluorescence (light wavelength crosstalk), polarization, and diffraction.

(5) Surface properties can be expressed as a linear combination of diffuse and specular reflectance and transmission functions:

$$\text{BDF} = k_{dr}\text{BRDF}_{diff} + k_{sr}\text{BRDF}_{spec} + k_{dt}\text{BTDF}_{diff} + k_{st}\text{BTDF}_{spec}$$

The coefficients k_{ij} are not assumed constant.

(6) Specular surfaces are not rough; all specular interaction is ideal.

3.1 Approach

Our approach is a hybrid of radiosity and ray tracing ideas. Rather than patch together these two algorithms, however, we seek a simple, coherent, hybrid algorithm. To provide the greatest generality of shape primitives and optical effects, we choose ray tracing as the visibility algorithm. Because ray tracing is weak at simulating global diffuse interaction, the principal task before us is therefore to determine an efficient method for calculating radiosities using ray tracing.

To exploit the view-independence and coherence of radiosity, we store radiosity with each diffuse surface, using an *adaptive radiosity texture*, or *rex*. A rex records the pattern of light and shadow and color bleeding on a surface. We store radiosity as a texture, rather than as a polygonization, in order to decouple the data structures for geometry and shading, and to facilitate adaptive subdivision of radiosity information; and we store it with the surface, rather than in a global octree [Ward88], or in a light image, based on the intuition that radiosities are intrinsic properties of a surface. We expect that the memory required for rexes will not be excessive, since dense sampling of radiosity will be necessary only where it has a high gradient, such as at shadow edges.

Next we need a general technique for computing the rexes. The paths by which photons travel through a scene can motivate our algorithm (figure 2). We can characterize each interaction along a photon's path from light (L) to eye (E) as either diffuse (D) or specular (S). Each path can therefore be labeled with some string in the set given by the regular expression $L(D|S)^*E$. Classic ray tracing simulates only $LDS^*E \mid LS^*E$ paths, while classic radiosity simulates only LD^*E. Eye ray tracing has difficulty finding paths such as LS^+DE because it doesn't know where to look for specularly reflected light when integrating the hemisphere. Such paths are easily simulated by light ray tracing, however.

We digress for a moment to discuss units. Light rays carry power (energy/time) and eye rays carry intensity (energy / (time * projected area * solid angle)). Each light ray carries a fraction of the total power emitted by the light.

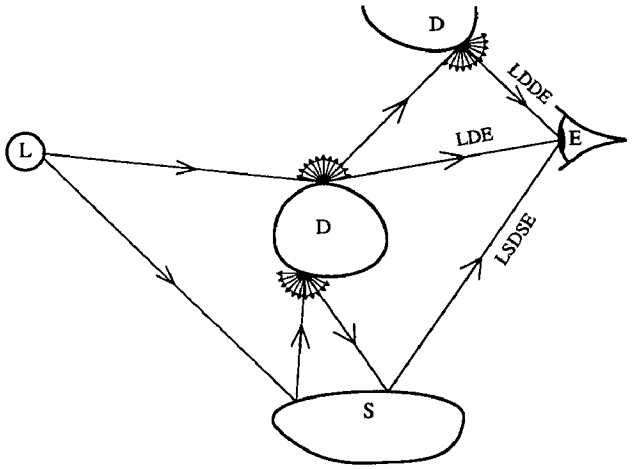

Figure 2: *Selected photon paths from light (L) to eye (E) by way of diffuse (D) and specular (S) surfaces. For simplicity, the surfaces shown are entirely diffuse or entirely specular; normally each surface would be a mixture.*

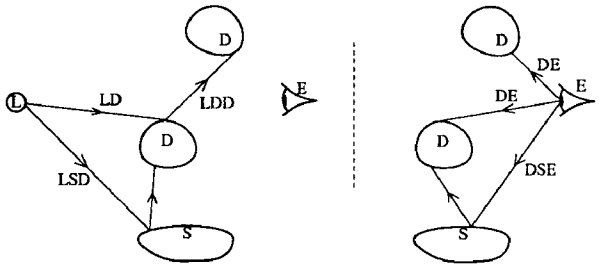

Figure 3: *Left: first level light ray tracing propagates photons from the light to the first diffuse surface on a path (e.g. LD and LSD); higher levels of progressive light ray tracing simulate indirect diffuse interaction (e.g. LDD). Right: eye ray tracing shoots rays from the eye, extracting radiosities from diffuse surfaces (e.g. it traces DE and DSE in reverse).*

We can simulate paths of the form LS^*D by shooting light rays (photons) into the scene, depositing the photon's power into the rex of the first diffuse surface encountered (figure 3, left). Such a light ray tracing pass will compute a first approximation to the radiosities. This can be followed by an eye ray tracing pass in which we trace DS^*E paths in a backward direction, extracting intensity from the rex of the first diffuse surface encountered (figure 3, right). The net effect of these two passes will be the simulation of all LS^*DS^*E paths. The rays of the two passes "meet in the middle" to exchange information. To simulate diffuse interreflection, we shoot progressively from bright surfaces [Cohen88] during the light ray tracing pass, thereby accounting for all paths: $L(S^*D)^*S^*E = L(D|S)^*E$. We call these two passes the *light pass* and *eye pass*. Such *bidirectional ray tracing* using adaptive radiosity textures can thus simulate all photon paths, in principle.

Our bidirectional ray tracing algorithm is thus a hybrid. From radiosity we borrowed the idea of saving and reusing the diffuse component, which is view-independent, and from ray tracing we borrowed the idea of discarding and recomputing the specular component, which is view-dependent.

3.2 All Sampling is Adaptive

There are three separate multidimensional sampling processes involved in this approach: sampling of directions from the light, sampling of directions from the eye (screen sampling), and sampling of radiosity on each diffuse surface.

3.3 Adaptive Radiosity Textures (Rexes)

Rexes are textures indexed by surface parameters u and v, as in standard texture mapping [Blinn76], [Heckbert86]. We associate a rex with every diffuse or partially-diffuse surface. By using a texture and retaining the initial geometry, instead of polygonizing, we avoid the polygonized silhouettes of curved surfaces common in radiosity pictures.

In the bidirectional ray tracing algorithm, the rexes collect power from incident photons during the light pass, and this information is used to estimate the true radiosity function during the eye pass (figure 4). Our rexes thus serve much like *density estimators* that estimate the probability density of a random variable from a set of samples of that random variable [Silverman86]. Density can be estimated using either histogram methods, which subdivide the domain into buckets; or kernel estimators, which store every sample and reconstruct the density as a sum of weighted kernels (similar to a spline).

The resolution of a rex should be related to its screen size. Ideally, we want to resolve shadow edges sharply in the final picture, which means that rexes should store details as fine as the preimage of a screen pixel. On the other hand, resolution of details smaller than this is unnecessary, since subpixel detail is beyond the Nyquist limit of screen sampling. Cohen's substructuring technique is adaptive, but its criteria appear to be independent of screen space, so it cannot adapt and optimize the radiosity samples for a particular view.

To provide the light pass with information about rex resolution we precede the light pass with a *size pass* in which we trace rays from the eye, labeling each diffuse surface with the minimum rex feature size.

3.3.1 Adaptive Light Sampling

Adaptive sampling of light rays is desirable for several reasons. Sharp resolution of shadow edges requires rays only where the light source sees a silhouette. Also, it is only necessary to trace light paths that hit surfaces visible (directly or indirectly) to the eye. Thirdly, omnidirectional lights disperse photons in a sphere of directions, but when such lights are far from the visible scene, as is the sun, the light ray directions that affect the final picture subtend a small solid angle. Finally, stratified sampling should be used for directional lights to effect their goniometric distribution. Thus, to avoid tracing irrelevant rays, we sample the sphere of directions adaptively [Sillion89], [Wallace89].

For area light sources, we use stratified sampling to distribute the ray origins across the surface with a density proportional to the local radiosity. Stratified sampling should also be used to shoot more light rays near the normal, since it is intensity that is constant with outgoing angle, while power is proportional to the cosine of the angle with the normal. If the surface has both a standard texture and a rex mapped onto it, then the rex should be modulated by this standard texture before shooting. With area light sources, the distribution to be integrated is thus four-dimensional: two dimensions for surface parameters u and v, and two dimensions for ray direction. For best results, a 4-D data structure such as a k-d tree should be used to record and adapt the set of light rays used.

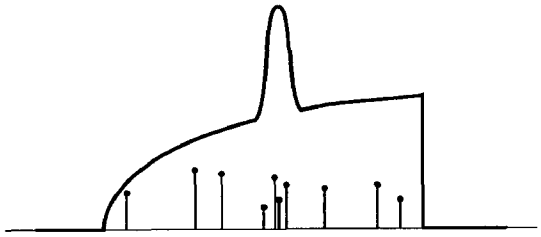

Figure 4: *Photons incident on a rex (shown as spikes with height proportional to power) are samples from the true, piecewise-continuous radiosity function (the curve). We try to estimate the function from the samples.*

3.3.2 Adaptive Eye Sampling

Eye rays (screen pixels) are sampled adaptively as well. Techniques for adaptive screen sampling have been covered well by others [Warnock69], [Whitted80], [Mitchell87], [Painter89].

3.4 Three Pass Algorithm

Our bidirectional ray tracing algorithm thus has three passes. We discuss these passes here in a general way; the details of a particular implementation are discussed in §4. The passes are:

size pass – record screen size information in each rex

light pass – progressively trace rays from lights and bright surfaces, depositing photons on diffuse surfaces to construct radiosity textures

eye pass – trace rays from eye, extracting light from diffuse surfaces to make a picture

Specular reflection and transmission bounces are followed on all three passes. Distribution ray tracing can be used in all passes to simulate the broad distributions of rough specular reflections and other effects.

3.4.1 Size Pass

As previously described, the size pass traces rays from the eye, recording information about the mapping between surface parameter space and screen space. This information is used by each rex during the light pass to terminate its adaptive subdivision.

3.4.2 Light Pass

Indirect diffuse interaction is simulated during the light pass by regarding bright diffuse surfaces as light sources, and shooting light rays from them, as in progressive radiosity. The rex records the shot and unshot power.

The adaptive algorithm for light ray tracing must ensure that: (a) a minimum level of light sampling is achieved; (b) more rays are devoted near silhouettes, shadows, and high curvature areas; (c) sharp radiosity gradients are resolved to screen pixel size; and (d) light rays and rexes are subdivided cooperatively.

3.4.3 Eye Pass

The eye pass is like a standard ray tracing algorithm except that the diffuse intensity is extracted out of the rex, instead of from a shadow ray. The radiosity of a surface patch is its power divided by its world-space surface area.

After the three passes are run, one could move the eye point and re-run the eye pass to generate other views of the scene, but the results would be inferior to those made by recomputing the rexes adapted to the new viewpoint.

3.4.4 Observations

Because light rays are concentrated on visible portions of the scene and radiosity is resolved adaptive to each surface's projection in screen space, the radiosity calculation performed in the light pass is view-dependent. But this is as it should be: although the exact radiosity values are view-independent, the radiosity sample locations needed to make a picture are not. When computing moving-camera animation, one could prime the rexes by running the size pass for selected key frames to achieve more view-independent sampling.

4 Implementation and Results

The current implementation realizes many, but not all, of the ideas proposed here. It performs bidirectional ray tracing using adaptive sampling for light, eye, and rex. It has no size pass, just a light pass and an eye pass. The program can render scenes consisting of CSG combinations of spheres and polyhedra. Specular interaction is assumed ideal, and diffuse transmission is not simulated. The light pass shoots photons from omnidirectional point light sources, and does not implement progressive radiosity. The implementation thus simulates only LS^*DS^*E paths at present. We trace ray trees, not just ray paths [Kajiya86].

4.0.5 Data Structures

Quadtrees were used for each of the 2-D sampling processes [Samet90]: one for the outgoing directions of each light, one for the parameter space of each radiosity texture, and one for the eye.

The light and eye quadtrees are quite similar; their records are shown below in pseudocode. Each node contains pointers to its child nodes (if not a leaf) and to its parent node. Light space is parameterized by (r, s), where r is latitude and s is longitude. and eye space (screen space) is parameterized by (x, y). Each node represents a square region of the parameter space whose corner is given by (r_0, s_0) or (x_0, y_0) and whose size is proportional to 2^{-level}.

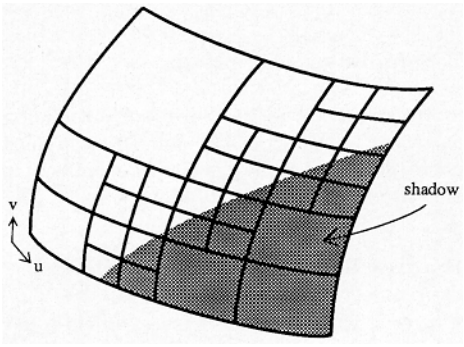

Figure 5: *Rex quadtree on a surface. Adaptive rex subdivision tries to subdivide more finely near a shadow edge.*

The light quadtree sends one light ray per node at a location uniformly distributed over the square. Also stored in each light quadtree node is the ID of the surface hit by the light ray, if any, and the surface parameters (u, v) at the intersection point. This information is used to determine the distance in parameter space between rex hits.

Eye quadtrees are simpler. Each node has pointers to the intensities at its corners. These are shared with neighbors and children. Eye ray tracing is currently uniform, not stochastic.

A rex quadtree node represents a square region of (u, v) parameter space on a diffuse surface (figure 5). Leaves in the rex quadtree act as histogram buckets, accumulating the number of photons and their power. Rex nodes also record the world space surface area of their surface patch.

```
light_node: type =                    {LIGHT QUADTREE NODE}
   record
      leaf: boolean;                  {is this a leaf?}
      mark: boolean;                  {should node be split?}
      level: int;                     {level in tree (root=0)}
      parent: ^light_node;            {parent node, if any}
      nw, ne, se, sw: ^light_node;    {four children, if not a leaf}
      r0, s0: real;                   {params of corner of square}
      r, s: real;                     {dir. params of ray (lat,lon)}
      surfno: int;                    {id of surface hit, if any}
      u, v: real;                     {surf params of surface hit}
   end;

eye_node: type =                      {EYE QUADTREE NODE}
   record
      leaf: boolean;                  {is this a leaf?}
      mark: boolean;                  {should node be split?}
      level: int;                     {level in tree (root=0)}
      parent: ^eye_node;              {parent node, if any}
      nw, ne, se, sw: ^eye_node;      {four children, if not a leaf}
      x0, y0: real;                   {coords of corner of square}
      inw, ine, ise, isw: ^color;     {intensity samples at corners}
   end;

rex_node: type =                      {REX QUADTREE NODE}
   record
      leaf: boolean;                  {is this a leaf?}
      mark: boolean;                  {should node be split?}
      level: int;                     {level in tree (root=0)}
      parent: ^rex_node;              {parent node, if any}
      nw, ne, se, sw: ^rex_node;      {four children, if not a leaf}
      u0, v0: real;                   {surf params of square corner}
      area: real;                     {surface area of this bucket}
      count: int;                     {#photons in bucket, if leaf}
      power: color;                   {accumulated power of bucket}
   end;
```

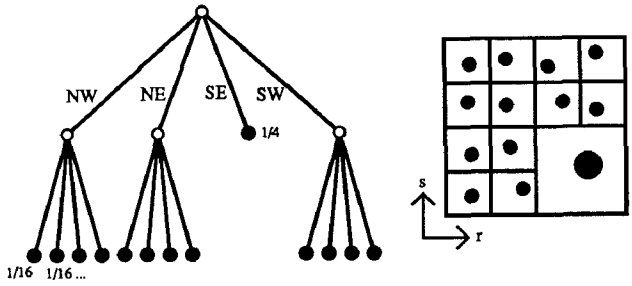

Figure 6: *Light quadtree shown schematically (left) and in light direction parameter space (right). When a light quadtree node is split, its power is redistributed to its four sub-nodes, which each send a ray in a direction (r,s) jittered within their parameter square. The fractional power of each light ray is shown next to the leaf node that sends it.*

The current implementation uses the following algorithm.

4.1 Light Pass

First, rex quadtrees are initialized to a chosen starting level (level 3, say, for 8x8 subdivision), and the counts and powers of all leaves are zeroed.

For each light, light ray tracing proceeds in breadth first order within the light quadtree, at level 0 tracing a single ray carrying the total power of the light, at level 1 tracing up to 4 rays, at level 2 tracing up to 16 rays, etc (figure 6). At each level, we adaptively subdivide both the light quadtree and the rex quadtrees. Changing the rex quadtrees in the midst of light ray shooting raises the histogram redistribution problem, however: if a histogram bucket is split during collection, it is necessary to redistribute the parent's mass among the children. There is no way to do this reliably without a priori knowledge, so we clear the rex at the beginning of each level and reshoot.

Processing a given level k of light rays involves three steps: (1) rex subdivision to split rex buckets containing a high density of photons, (2) light marking to mark light quadtree nodes where more light rays should be sent, and (3) light subdivision to split marked light nodes.

Rex subdivision consists of a sweep through every rex quadtree in the scene, splitting all rex buckets whose photon count exceeds a chosen limit. All counts and powers are zeroed at the end of this sweep.

Light marking traverses the light quadtree, marking all level k nodes that meet the subdivision criteria listed below.

(1) Always subdivide until a minimum level is reached.

(2) Never subdivide beyond a maximum level (if a size pass were implemented, it would determine this maximum level locally).

Otherwise, look at the light quadtree neighbors above, below, left, and right, and subdivide if the following is true:

(3) The ray hit a diffuse surface, and one of the four neighbors of the rex node hit a different surface or was beyond a threshold distance in (u,v) parameter space from the center ray's.

To help prevent small feature neglect, we also mark for subdivision all level $k-1$ leaves that neighbor on level k leaves that are marked for subdivision. This last rule guarantees a *restricted quadtree* [Von Herzen87] where each leaf node's neighbors are at a level within plus or minus one of the center node's.

Light subdivision traverses the light quadtree splitting the marked nodes. Subdividing a node splits a ray of power p into four rays of power $p/4$ (figure 6). When a light node is created (during initialization or subdivision) we select a point at random within its square (r,s) domain to achieve jittered sampling [Cook86] and trace a ray in that direction. Marked nodes thus shoot four new rays, while unmarked nodes re-shoot their rays. During light ray tracing we follow specular bounces, splitting the ray tree and subdividing the power according to the reflectance/transmittance coefficients k_{ij}, and deposit their power on any diffuse surface that are hit. When a diffuse surface is hit, we determine (u,v) of the intersection point, and descend the surface's rex quadtree to find the rex node containing that point. The power of that node is incremented by the power of the ray times the cosine of the incident angle.

4.2 Eye Pass

The eye pass is a fairly standard adaptive supersampling ray tracing algorithm: nodes are split when the intensity difference between the four corners exceeds some threshold. To generate a picture, nodes larger than a pixel perform bilinear interpolation to fill in the pixels they cover, while nodes smaller than a pixel are averaged together to compute a pixel. The picture is stored in floating point format initially, then scaled and clamped to the range [0,255] in each channel.

4.3 Results

Figures 7-12 were generated with this program. Figures 7, 8, and 9 show the importance of coordinating the light ray sampling process with the rex resolution. Sending too few light rays results in a noisy radiosity estimate from the rex, and too coarse a rex results in blocky appearance. When the rex buckets are approximately screen pixel size and the light ray density deposits several photons per bucket (at least 10, say), the results are satisfactory. We estimate the radiosity using a function that is constant within each bucket; this simple estimator accounts for the blockiness of the images. If bilinear interpolation were used, as in most radiosity algorithms, we could trade off blockiness for blurriness.

Figure 10 shows adaptive subdivision of a rex quadtree, splitting more densely near shadow edges (the current splitting criteria cause unnecessary splitting near the border of the square). Its rex quadtree is shown in figure 11.

Figure 12 shows off some of the effects that are simulated by this algorithm.

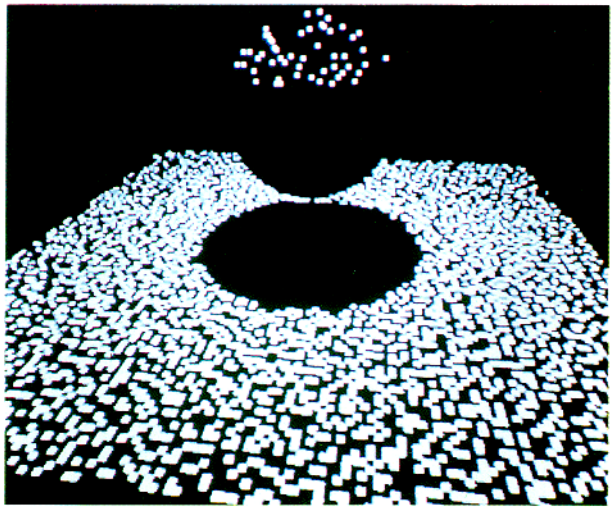

Figure 7: *Noisy appearance results when too few light rays are received in each rex bucket (too few light rays or too fine a rex). Scene consists of a diffuse sphere above a diffuse floor both illuminated by an overhead light source.*

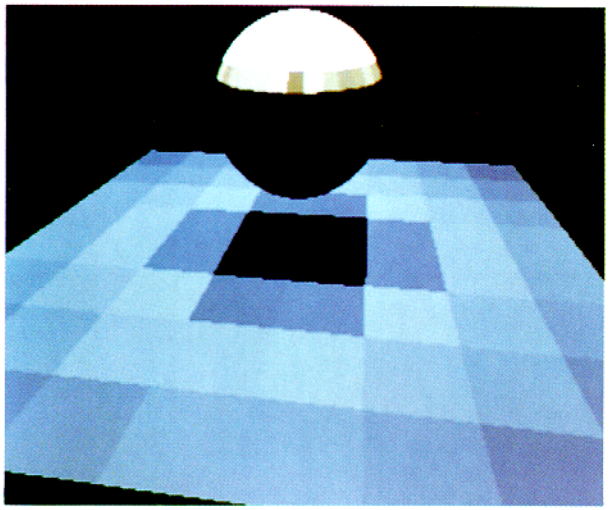

Figure 8: *Blocky or blurry appearance results when rex buckets are much larger than a screen pixel (too coarse a rex).*

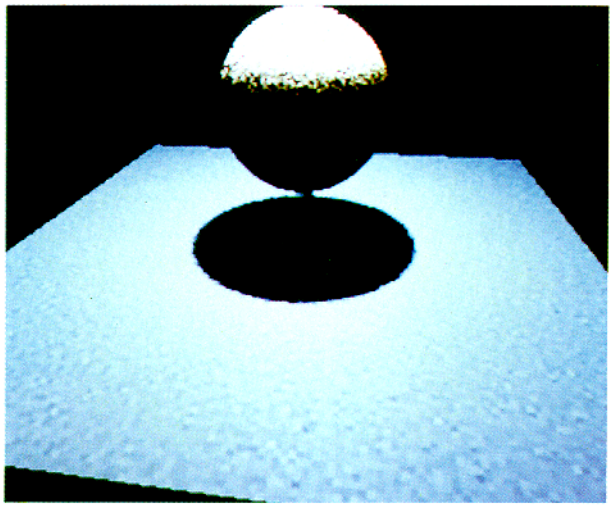

Figure 9: *Proper balance of light sampling and rex sampling reduces both noise and blockiness.*

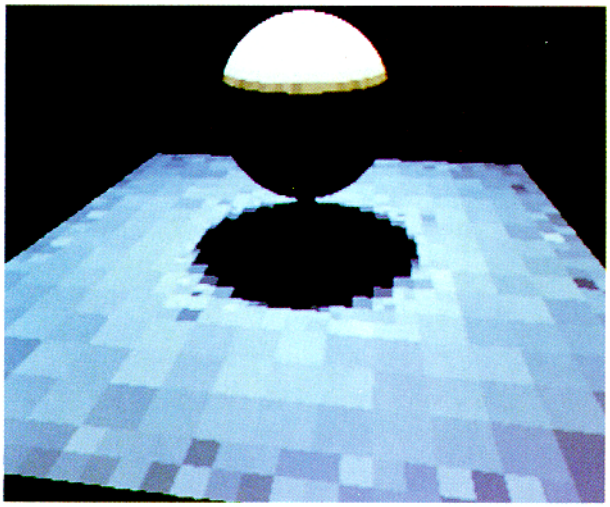

Figure 10: *Rex with adaptation: the rex of the floor is initially a single bucket, but it splits adaptively near the edges of the square and near the shadow edge.*

Statistics for these images are listed below, including the number of light rays, the percentage of light rays striking an object, the resolution of the rex, the resolution of the final picture, the number of eye rays, and the CPU time. All images were computed on a MIPS R2000 processor. The lens image used about 20 megabytes of memory, mostly for the light quadtree. Ray trees were traced to a depth of 5.

#LRAYS	%HIT	REX	EYE	#ERAYS	TIME	FIG
87,400	10%	128^2	256^2	246,000	1.0 min.	fig. 7
87,400	10%	8^2	256^2	139,000	0.6 min.	fig. 8
822,000	68%	128^2	256^2	146,000	3.5 min.	fig. 9
331,000	20%	vbl	256^2	139,000	1.3 min.	fig. 10
1,080,000	61%	256^2	512^2	797,000	6.4 min.	fig. 12

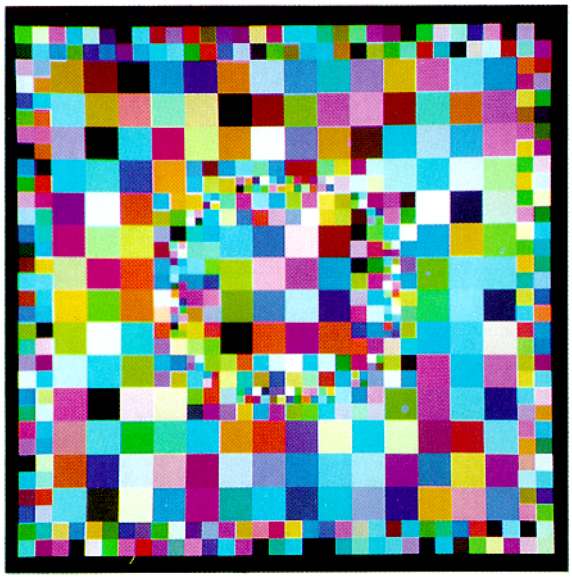

Figure 11: *Rex quadtree in (u, v) space of previous figure's floor. Each leaf node's square is colored randomly. Note the subdivision near the shadow edge and the quadtree restriction.*

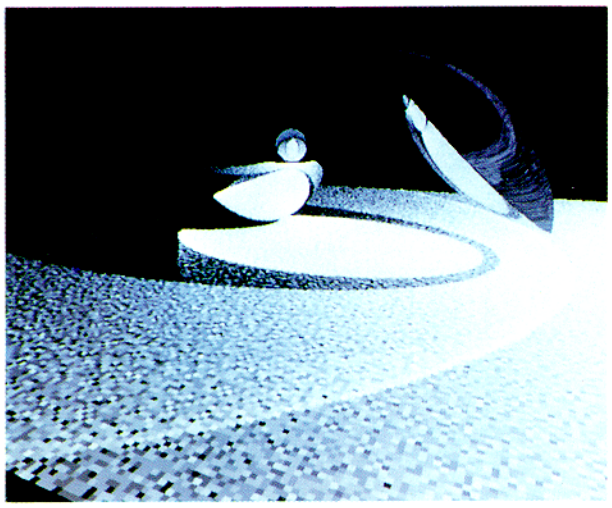

Figure 12: *Light focusing and reflection from a lens and chrome ball. Scene is a glass lens formed by CSG intersection of two spheres, a chrome ball, and a diffuse floor, illuminated by a light source off screen to the right. Note focusing of light through lens onto floor at center (an LSSD path), reflection of refracted light off ball onto floor (an LSSSD path involving both transmission and reflection), the reflection of light off lens onto floor forming a parabolic arc (an LSD path), and the reflection of the lens in the ball (a LSSDSSE path, in full).*

5 Conclusions

The bidirectional ray tracing algorithm outlined here appears to be an accurate, general approach for global illumination of scenes consisting of diffuse and pure specular surfaces. It is accurate because it can account for all possible light paths; and it is general because it supports both the radiosity and ray tracing realms: shapes both planar and curved materials both diffuse and specular, and lights both large and small. Distribution ray tracing can be used to simulate effects not directly supported by the algorithm.

Adaptive radiosity textures (rexes) are a new data structure that have several advantages over previous radiosity storage schemes. They can adaptively subdivide themselves to resolve sharp shadow edges to screen pixel size, thereby eliminating visible artifacts of radiosity sampling. Their subdivision can be automatic, requiring no ad hoc user-selected parameters.

The current implementation is young, however, and many problems remain. A terse list follows: Good adaptive sampling of area light sources appears to require a 4-D data structure. Better methods are needed to determine the number of light rays. The redistribution problems of histograms caused us to send each light ray multiple times. To avoid this problem we could store all (or selected) photon locations using kernel estimators [Silverman86]. Excessive memory is currently devoted to the light quadtree, since one node is stored per light ray. Perhaps the quadtree could be subdivided in more-or-less scanline order, and the memory recycled (quadtree restriction appears to complicate this, however). Adaptive subdivision algorithms that compare the ray trees of neighboring rays do not mix easily with path tracing and distribution ray tracing, because the latter obscure coherence. Last but not least, the interdependence of light ray subdivision and rex subdivision is precarious.

In spite of these challenges, we are hopeful. The approach of bidirectional ray tracing using adaptive radiosity textures appears to contain the mechanisms needed to simulate global illumination in a general way.

6 Acknowledgements

Thanks to Greg Ward for discussions about the global illumination problem, to Steve Omohundro for pointing me to the density estimation literature, to Ken Turkowski and Apple Computer for financial support, and to NSF grant CDA-8722788 for "Mammoth" time.

7 References

[Appel68] Arthur Appel, "Some Techniques for Shading Machine Renderings of Solids", *AFIPS 1968 Spring Joint Computer Conf.*, vol. 32, 1968, pp. 37-45.

[Arvo86] James Arvo, "Backward Ray Tracing", *SIGGRAPH '86 Developments in Ray Tracing seminar notes*, Aug. 1986.

[Atherton78] Peter R. Atherton, Kevin Weiler, Donald P. Greenberg, "Polygon Shadow Generation", *Computer Graphics* (SIGGRAPH '78 Proceedings), vol. 12, no. 3, Aug. 1978, pp. 275-281.

[Baum89] Daniel R. Baum, Holly E. Rushmeier, James M. Winget, "Improving Radiosity Solutions Through the Use of Analytically Determined Form Factors", *Computer Graphics* (SIGGRAPH '89 Proceedings), vol. 23, no. 3, July 1989, pp. 325-334.

[Blinn76] James F. Blinn, Martin E. Newell, "Texture and Reflection in Computer Generated Images", *CACM*, vol. 19, no. 10, Oct. 1976, pp. 542-547.

[Buckalew89] Chris Buckalew, Donald Fussell, "Illumination Networks: Fast Realistic Rendering with General Reflectance Functions", *Computer Graphics* (SIGGRAPH '89 Proceedings), vol. 23, no. 3, July 1989, pp. 89-98.

[Cohen85] Michael F. Cohen, Donald P. Greenberg, "The Hemi-Cube: A Radiosity Solution for Complex Environments", *Computer Graphics* (SIGGRAPH '85 Proceedings), vol. 19, no. 3, July 1985, pp. 31-40.

[Cohen86] Michael F. Cohen, Donald P. Greenberg, David S. Immel, Philip J. Brock, "An Efficient Radiosity Approach for Realistic Image Synthesis", *IEEE Computer Graphics and Applications*, Mar. 1986, pp. 26-35.

[Cohen88] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, Donald P. Greenberg, "A Progressive Refinement Approach to Fast Radiosity Image Generation", *Computer Graphics* (SIGGRAPH '88 Proceedings), vol. 22, no. 4, Aug. 1988, pp. 75-84.

[Cook84] Robert L. Cook, Thomas Porter, Loren Carpenter, "Distributed Ray Tracing", *Computer Graphics* (SIGGRAPH '84 Proceedings), vol. 18, no. 3, July 1984, pp. 137-145.

[Cook86] Robert L. Cook, "Stochastic Sampling in Computer Graphics", *ACM Transactions on Graphics*, vol. 5, no. 1, Jan. 1986, pp. 51-72.

[Dippe85] Mark A. Z. Dippe, Erling Henry Wold, "Antialiasing Through Stochastic Sampling", *Computer Graphics* (SIGGRAPH '85 Proceedings), vol. 19, no. 3, July 1985, pp. 69-78.

[Goral84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, Bennett Battaile, "Modeling the Interaction of Light Between Diffuse Surfaces", *Computer Graphics* (SIGGRAPH '84 Proceedings), vol. 18, no. 3, July 1984, pp. 213-222.

[Hall89] Roy Hall, *Illumination and Color in Computer Generated Imagery*, Springer Verlag, New York, 1989.

[Heckbert84] Paul S. Heckbert, Pat Hanrahan, "Beam Tracing Polygonal Objects", *Computer Graphics* (SIGGRAPH '84 Proceedings), vol. 18, no. 3, July 1984, pp. 119-127.

[Heckbert86] Paul S. Heckbert, "Survey of Texture Mapping", *IEEE Computer Graphics and Applications*, vol. 6, no. 11, Nov. 1986, pp. 56-67.

[Immel86] David S. Immel, Michael F. Cohen, Donald P. Greenberg, "A Radiosity Method for Non-Diffuse Environments", *Computer Graphics* (SIGGRAPH '86 Proceedings), vol. 20, no. 4, Aug. 1986, pp. 133-142.

[Kajiya86] James T. Kajiya, "The Rendering Equation", *Computer Graphics* (SIGGRAPH '86 Proceedings), vol. 20, no. 4, Aug. 1986, pp. 143-150.

[Lee85] Mark E. Lee, Richard A. Redner, Samuel P. Uselton, "Statistically Optimized Sampling for Distributed Ray Tracing", *Computer Graphics* (SIGGRAPH '85 Proceedings), vol. 19, no. 3, July 1985, pp. 61-67.

[Mitchell87] Don P. Mitchell, "Generating Antialiased Images at Low Sampling Densities", *Computer Graphics* (SIGGRAPH '87 Proceedings), vol. 21, no. 4, July 1987, pp. 65-72.

[Nishita85] Tomoyuki Nishita, Eihachiro Nakamae, "Continuous Tone Representation of 3-D Objects Taking Account of Shadows and Interreflection", *Computer Graphics* (SIGGRAPH '85 Proceedings), vol. 19, no. 3, July 1985, pp. 23-30.

[Painter89] James Painter, Kenneth Sloan, "Antialiased Ray Tracing by Adaptive Progressive Refinement", *Computer Graphics* (SIGGRAPH '89 Proceedings), vol. 23, no. 3, July 1989, pp. 281-288.

[Reeves87] William T. Reeves, David H. Salesin, Robert L. Cook, "Rendering Antialiased Shadows with Depth Maps", *Computer Graphics* (SIGGRAPH '87 Proceedings), vol. 21, no. 4, July 1987, pp. 283-291.

[Samet90] Hanan Samet, *The Design and Analysis of Spatial Data Structures*, Reading, MA, Addison-Wesley, 1990.

[Shao88] Min-Zhi Shao, Qun-Sheng Peng, You-Dong Liang, "A New Radiosity Approach by Procedural Refinements for Realistic Image Synthesis", *Computer Graphics* (SIGGRAPH '88 Proceedings), vol. 22, no. 4, Aug. 1988, pp. 93-101.

[Siegel81] Robert Siegel, John R. Howell, *Thermal Radiation Heat Transfer*, Hemisphere Publishing Corp., Washington, DC, 1981.

[Sillion89] Francois Sillion, Claude Puech, "A General Two-Pass Method Integrating Specular and Diffuse Reflection", *Computer Graphics* (SIGGRAPH '89 Proceedings), vol. 23, no. 3, July 1989, pp. 335-344.

[Silverman86] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, London, 1986.

[Strauss88] Paul S. Strauss, *BAGS: The Brown Animation Generation System*, PhD thesis, Tech. Report CS-88-2, Dept. of CS, Brown U, May 1988.

[Von Herzen87] Brian Von Herzen, Alan H. Barr, "Accurate Triangulations of Deformed, Intersecting Surfaces", *Computer Graphics* (SIGGRAPH '87 Proceedings), vol. 21, no. 4, July 1987, pp. 103-110.

[Wallace87] John R. Wallace, Michael F. Cohen, Donald P. Greenberg, "A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods", *Computer Graphics* (SIGGRAPH '87 Proceedings), vol. 21, no. 4, July 1987, pp. 311-320.

[Wallace89] John R. Wallace, Kells A. Elmquist, Eric A. Haines, "A Ray Tracing Algorithm for Progressive Radiosity", *Computer Graphics* (SIGGRAPH '89 Proceedings), vol. 23, no. 3, July 1989, pp. 315-324.

[Ward88] Gregory J. Ward, Francis M. Rubinstein, Robert D. Clear, "A Ray Tracing Solution for Diffuse Interreflection", *Computer Graphics* (SIGGRAPH '88 Proceedings), vol. 22, no. 4, Aug. 1988, pp. 85-92.

[Warnock69] John E. Warnock, *A Hidden Surface Algorithm for Computer Generated Halftone Pictures*, TR 4-15, CS Dept, U. of Utah, June 1969.

[Watt90] Mark Watt, "Light-Water Interaction using Backward Beam Tracing", *Computer Graphics* (SIGGRAPH '90 Proceedings), Aug. 1990.

[Whitted80] Turner Whitted, "An Improved Illumination Model for Shaded Display", *CACM*, vol. 23, no. 6, June 1980, pp. 343-349.

[Williams78] Lance Williams, "Casting Curved Shadows on Curved Surfaces", *Computer Graphics* (SIGGRAPH '78 Proceedings), vol. 12, no. 3, Aug. 1978, pp. 270-274.