ILLUMINATION NETWORKS: FAST REALISTIC RENDERING WITH ARBITRARY REFLECTANCE FUNCTIONS

Chris Buckalew and Donald Fussell

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-89-01 January 1989

ILLUMINATION NETWORKS: FAST REALISTIC RENDERING WITH ARBITRARY REFLECTANCE FUNCTIONS

Chris Buckalew and Donald Fussell
Department of Computer Sciences, The University of Texas at Austin

ABSTRACT

We present a technique for modeling global illumination which allows a wide variety of reflectance functions. Scene coherence is exploited in a preprocessing step in which the geometry is analyzed using iterative techniques. Memory is traded for speed, in anticipation of the high memory capacities of workstations of the future. The algorithm operates well over a wide range of time and image quality constraints: realistic results may be produced very quickly while very accurate results may be produced given more time and space. The method can be extended for animation and parallelization.

CR Categories and Subject Descriptors: 1.3.3 [Computer Graphics]: Picture/Image Generation--Display algorithms. 1.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

General Terms: Algorithms

Additional Key Words and Phrases: global illumination, radiosity, ray tracing, memory, specular, diffuse, data structure, incremental, ray space.

INTRODUCTION

Most techniques used to model global illumination are well-suited to particular surface reflectance functions. Ray tracing methods such as [14] render specular reflection from surfaces efficiently and radiosity methods such as [3] and [10] handle diffuse reflection from surfaces very well. Generalizations of ray tracing techniques to handle diffuse reflectance functions ([1] [4] [8] [13]) and of radiosity techniques to handle specular reflectance functions ([6]) have been significantly slower. This has led to the development of hybrid methods which capitalize on the strengths of both techniques ([12]) to efficiently render scenes containing both diffuse and specular surfaces.

We have developed a fast algorithm for rendering scenes containing both diffuse and non-diffuse surfaces. The method can produce illumination of arbitrary accuracy, with very realistic results produced in a very short time. Although we have explicitly traded memory for speed, the memory requirements of this method are much smaller than the virtual memory capacities of most of today's workstations and well within the real memory capacities of tomorrow's machines. The algorithm can be extended to allow fast animation that saves information between frames, and it can be efficiently parallelized.

ILLUMINATION NETWORKS--THE PATCH-LINK MODEL

In diffuse reflection, a surface element interacts with most of the other elements that are visible from its front surface. Radiosity techniques solve this problem with virtual frame buffers called hemicubes that represent the scene from each of the elements' points of view. Unfortunately, the hemicubes must be stored at huge memory cost or recalculated frequently, and they contain little directional information. For non-diffuse reflection, light incident on each surface element reflects in a particular direction, so detailed directional information must be used in modeling this type of reflection. Ray tracing methods solve this problem by calculating rays from one surface element to another as they are needed. In specular environments relatively few rays are needed, so the high cost of computing them individually is worthwhile. However, ray tracing diffuse environments requires far more rays to sample the many elements of the scene providing diffuse illumination of each element, and the expense of the ray-object intersection calculations may become significant.

We have combined the good features of both techniques while avoiding some of the drawbacks by using a data structure called an illumination network. This data structure is based on a model of light and surface interaction which we call the patch-link model. Surfaces are broken into small areas called patches. Associated with each patch is a set of links. Each link connects the patch to another patch. Light can travel through the scene only by means of these links. The reflectance function associated with each patch consists of a mapping from "incoming" links to "outgoing" links; that is, the input to the function is a collection of incoming light intensities over the set of links, which the function maps to a collection of outgoing light intensities over the same set of links.

The set of links associated with each patch connect it to much of its environment, thus fulfilling the function of hemicubes at more modest storage cost. In addition, the links encode the directional relationships of the patches they connect, serving the function of rays in ray tracing methods. The high cost of computing the large number of links necessary is avoided and scene coherence is exploited by calculating all the links incrementally in a preprocessing step.

The patch-link model clearly works well in the limit, where patch size approaches zero and the number of links ap-

proaches infinity, and clearly works poorly where patch size is very large and number of links very small. The correct choice for these sizes depends very much on the reflectance functions of the objects in the scene and the desired resolution and accuracy of the image. Shiny objects and sharp shadows, for instance, require smaller patch sizes, and greater accuracy requires more links.

AN OVERVIEW OF THE ALGORITHM

The algorithm consists of three main parts. The first is a preprocessing step in which the objects in the scene are divided into patches and the links between patches in the scene are established. Reflectance functions are set up at this time as well: reflectance functions are described by an NxN matrix, where the (i,j)th element of the matrix gives the fraction of the light arriving on link i that will leave on link j. The reflectance function, then, is determined by the coefficients of this matrix. For a completely diffuse patch, all the coefficients of the matrix would be equal, with the sums of each of the rows equal to one. In reality a surface will absorb some of the incident radiation, so the sums of each of the rows will actually be less than one--if this were not the case convergence might not be achievable. For a perfect mirror, each row would have one coefficient of 1 with the rest 0, because all the light coming in over link i will leave by the link associated with the reflectance angle of link i.

The second part of the algorithm is the distribution process. Light travels outward over the links associated with the light-emitting patches and arrives at the patches on the other end of these links. For each of these patches, the incoming light is passed through the patch's reflectance function, and the result is passed outward along the patch's links to the patches on the other end of the links, some of which will be the original emitting patches. This process continues until there is no more light incoming to any patch, which occurs when all the light has been reflected out of the scene or absorbed.

Following convergence, the scene must be rendered; the algorithm gives view-independent results, so viewing parameters are set up and light falling on the screen is determined. The preprocess and the distribution process are cleanly separated, so that once the illumination network is set up in the preprocess, it may be reused by multiple passes of the distribution process. Objects' reflectance functions and light source strengths may be changed between passes; the preprocess merely encodes the geometry of the scene into an illumination network.

LINKS

The collection of links represents a finite subset of all the light rays which pass through the scene. An extreme approach is to select at least one ray which connects each pair of patches and add it to the collection of links. If there are N patches this process will result in at least N^2 links. Another approach is to choose an "evenly distributed" and geometrically uniform set of rays which intersect the scene. Any of the rays in this set which intersect two objects will become links. Links are implemented as pointers; if a ray intersects two patches which face each other, each patch will have a pointer for that link, pointing to the other patch. The geometry thus becomes implicit: the pointer indicates the patch that is hit if light leaves the patch in a certain direction. In our implementation the direction in 3-space associated with a pointer is determined by its location in the patch's array of such pointers.

Other techniques, such as [2] and [11], have used a finite set of rays to partition all the rays intersecting the scene into areas of interest, but our method uses only a predetermined finite set of rays for all light transport throughout the scene. We would like this set to be uniform so that it lends itself to incremental ray-object intersection calculations. To achieve this end we chose a formulation of ray space similar in some respects to that of [2].

[2]'s ray space is 5-dimensional; a ray is represented by a 3-D origin and a 2-D direction. We use a different, 4-dimensional formulation. Rays are represented by lines, each described by two slopes and a two-dimensional intercept. The 2-D intercept gives the line's intersection with an intercept plane (one of x-y, x-z, or y-z planes) and the slope of the line in each of the intercept plane's two dimensions. There is no ray origin or ray direction as with [2]; origins are built into the data structure (the ray-object intersection points serve as origins) and the ray's direction along the line (plus or minus) is implied by the direction that the object faces.

This formulation of ray space may be visualized more easily by considering the analogous formulation for two-dimensional scenes. In 2-space lines are described by one intercept (on the x-axis or on the y-axis) and one slope in the appropriate dimension (dy/dx if intercept is on the y-axis or dx/dy if the intercept is on the x-axis). Thus this ray space for 2-D scenes is two-dimensional.

However, some rays cannot be described by a y-intercept and y-slope, just as some rays cannot be described by an x-intercept and an x-slope. To solve this problem we partition ray space into two regions, one for rays whose y-slope is between -1 and +1, the other for rays whose x-slope is between -1 and +1. If we restrict our scene to the first quadrant of the unit square (in 3-space, the first octant of the unit cube), then we can put boundaries on the range of the x- and y-intercepts. If y-slopes must be less than one, then the lowest possible y-intercept is -1. Similarly if y-slopes must be greater than -1, the highest possible y-intercept is 2. This is shown in Figure 1.

All the rays passing through the first quadrant of the unit square are contained in two areas of ray space, one for each dimension, limited by -1 and +1 on the slope axis and -1 and +2 on the intercept axis. These areas, and

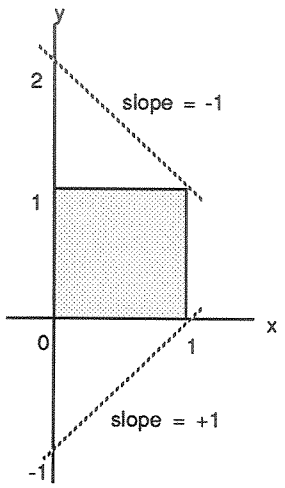

FIGURE 1: slope and intercept bounds on the unit square

some sample rays, are shown in Figure 2. Our uniform selection of rays that will become links is simply a uniform sampling of points in the ray space we have formulated. We choose an appropriate interval for each of the slope and intercept axes, set up a grid with these intervals, and select the grid points.

In 3-space the process is exactly analogous. We have three volumes of ray space, one for each intercept plane, each region bounded by -1 and +1 on the slope axes and -1 and +2 on the intercept axes. Our finite set of rays is chosen by a uniform point sample in this four-dimensional ray space, just as in the 2-space case. Two things to note: not all the rays inside these limits actually pass through the scene, and the rays are uniform in terms of slopes and intercepts. They are not uniform in other ways; for example, consider all the rays of this sampling that pass through the origin: the spatial density of the rays decreases as the slopes approach zero. This is shown in the 2-dimensional example in Figure 3. We correct for this error by associating a weighting factor with each slope which is applied to all light traveling along any link having that slope.

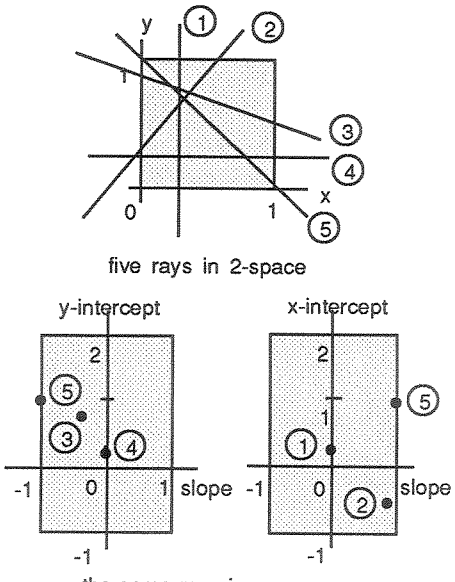

FIGURE 2: scene space and ray space

In the current implementation, objects are limited to planar polygons. It is very easy to calculate ray-object intersections incrementally with this set of rays. Once we calculate the intersection point of a given ray and a given object, it is a simple matter to determine the intersection point of the same object and the "next" ray. The planar geometry of the object and the geometry of the relationship of one ray to the next result in very simple incremental calculations, which are described more fully in a later section.

PATCHES AND THE ILLUMINATION NETWORK

Now that we have selected a uniform set of rays which will become links, we must divide the scene into patches. If the patch size is too small, it will not intersect very many rays, and thus will not have very many links. If a patch is too large, one ray for each slope must be chosen as a link out of perhaps many such rays. A bad choice for the link might result in serious errors. We deal with this problem by dividing each object into rectangular areas each of which is small enough that it can intersect no more than one ray for each slope. This is accomplished by sizing the rectangular areas such that when projected onto the intercept planes the sides of the projected rectangles will be equal to the intercept interval.

The illumination network data structure is simple. Each patch has an array of pointers, one for each slope, and with each pointer is associated a buffer for incoming light called the in-buffer. Each pointer represents a link, and it points to the patch on the other end of the link. The in-buffer accumulates unprocessed light that has come in over the link, but that has not yet been sent through the reflectance function. The size of the array and the total number of patches depends on the resolution required.

For patches that have only a diffuse component to the reflec-

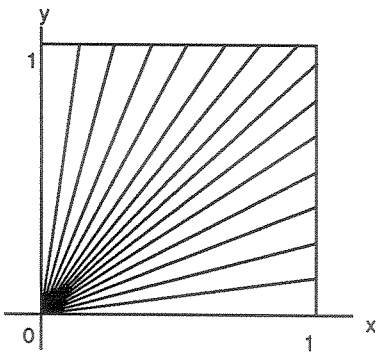

FIGURE 3: spatial density varies with slope

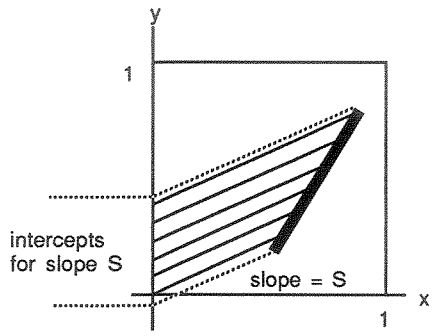

FIGURE 4: intercepts associated with a given slope

tance function, the direction of incoming light is not important since it will be evenly distributed. For these patches, then, a great deal of memory may be saved by eliminating their in-buffers and substituting a global in-buffer which receives all the light arriving at the patch, regardless of direction.

THE PREPROCESSING STEP

This part of the algorithm sets up the illumination network and fills in the links. The preprocess utilizes the slope-intercept format of rays: a ray will intersect an object if its intercept lies inside a polygon formed by projecting the object onto the intercept plane along the ray's slope. This is illustrated in a 2-D example in Figure 4. Basically, then, for each slope S, we project each object onto the intercept plane along S and run a standard scan-conversion algorithm on the resulting polygon. All rays with slope S whose intercepts lie within the polygon intersect the object and those intersection points can be calculated incrementally.

The incremental calculation of ray-object intersections is the key to the speed of the preprocess. Given an object and two slopes, the intercepts are easily determined by scan-converting, as described above. However, we must also determine the coordinates of the ray-object intersections that project to these intercepts in order to find which patches the ray-object intersections are in. Suppose that the plane equation of an object is Ax + By + Cz + D = 0 and we wish to find the intersection points of all the rays intersecting the y-z plane with slopes yslope and zslope. If the intercept of a particular ray is (yint, zint), then the coordinates of the point (x, y, z) where the ray intersects the object are determined as follows:

$$Ax + By + Cz + D = 0$$

$$Ax + B(yslope*x + yint) + C(zslope*x + zint) + D = 0$$

$$Ax + B*yslope*x + C*zslope*x = -D - B*yint - C*zint$$

$$x = \frac{(-D - B*yint - C*zint)}{(A + B*yslope + C*zslope)}$$

This is the result for the x-coordinate; y and z are found similarly. If the denominator is zero, then the y- and z-slopes are both parallel to the object's plane, and if there is a solution, it will not be unique.

This calculation is not at all fast, and it may be made faster by noticing that the denominator need be calculated only once for each pair of slopes. It is then combined with the numerator coefficients so that

$$D' = D \;/\; (A + B*yslope + C*zslope)$$

$$B' = B \;/\; (A + B*yslope + C*zslope)$$

$$C' = C \;/\; (A + B*yslope + C*zslope)$$

The calculation is now

$$x = -D' - B'*yint - C'*zint$$

which is certainly faster, but if we take advantage of the fact that we are using a scan-conversion algorithm to determine yint and zint, it may be made faster yet. If zint remains constant throughout each scan and if at each step of the scan conversion yint is incremented by Δy, then

$x_0 = -D' - B'*yint - C'*zint$ at the start of the scan, and

$x_{i+1} = x_i - B'*\Delta y$ thereafter.

Since $B'*\Delta y$ is fixed throughout the scan conversion, each step costs only an addition. The other two coordinates are similarly determined.

These intersection points are stored until all intersections have been found for S. At this point each ray with slope S is examined for intersections; links are set up between any pair of patches which have consecutive intersections on the ray and which face each other, as shown in Figure 5.

The regularity of the geometry of the patches and links can cause noticeable aliasing. To attenuate the effects of this problem we ray-object intersection points are jittered by a random fraction of the intercept interval amount. This procedure results in a small perturbation in the slope represented by each link. This loss of regularity is achieved very cheap-

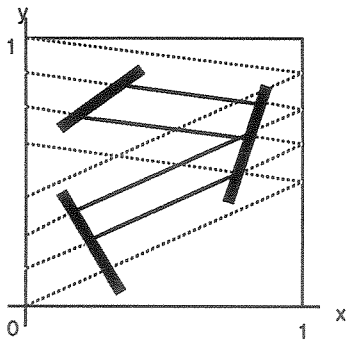

FIGURE 5: three objects and four of the links between them

ly, and the resultant noise reduces the effect of the aliasing artifacts. The general problem of aliasing due to uniform point sampling and methods of dealing with it are discussed in [9].

THE DISTRIBUTION PROCESS

This part of the algorithm distributes the light throughout the scene until convergence is achieved. Light is first "shot" from the light sources, which have patches and links like any other objects. This light is sent into the scene via the light sources' links, and is accumulated in the recipient patches' in-buffers. Following the light sources' depletion, each patch in the scene is sequentially examined for any unprocessed light in the patch's in-buffers. If any is found, the contents of all the patch's in-buffers is run through the reflectance function and the result is sent outward over the patch's links into other patches' in-buffers. This process is continued until no patch has any light in its in-buffers.

The reflectance function calculations consume most of the time used in this process. As mentioned above, the reflectance function multiplies the array of in-buffers by a matrix of coefficients and the result is mapped back out to the array of links. Extremely specular reflectance functions result in sparse reflectance matrices, while extremely diffuse functions have reflectance matrices that have few non-zero entries. For the case of a reflectance function having a constant component (a totally diffuse reflectance function would be constant), the matrix multiplication may be partially or completely bypassed; the portion of the incoming light contributing to the constant component is summed into a global accumulator. The contents of the accumulator are then divided up equally over the links. For example, if a surface reflects 60% of incoming light in a tight specular pattern and 40% in a diffuse pattern, then 40% of the incoming light will be summed into the global buffer and distributed evenly while the remainder will be sent through the reflectance matrix, whose coefficients will indicate only the specular reflectance. This usually results in a sparser matrix, with a resultant time savings. When the patch is totally diffuse, the matrix will be zero, and incoming light is processed very swiftly.

RENDERING

The algorithm is view-independent; no eyerays need be calculated prior to rendering. Given the standard viewing parameters, eyerays are shot from the eyepoint through each pixel of the image plane. Each eyeray is matched to the "closest" link (in terms of slope and intercept), and the eyeray's intersection point with the nearest object (found with a standard Z-buffer algorithm) determines the patch it hits. The amount of light that the pixel sees is given by the total amount of light reflected from that patch over that link, which has been accumulated in an out-buffer associated with the link.

This procedure is not sufficient to produce good results, however. Sudden jumps in intensity may occur as the eyerays move from patch to patch, or the slope that the eyeray is coerced to changes. To avoid these problems, the light assigned to an eyeray is interpolated in two different ways: the light intensities associated with the rays surrounding the eyeray are interpolated for patches in the intersected patch's 8-neighborhood, and then these values are themselves interpolated. This process is illustrated in Figure 6. A given eyeray is associated with the four slopes that surround it--the x-z slopes immediately to the left and right of the eyeray's x-z slope, and the x-y slopes immediately above and below the eyeray's x-y slope--these are called the bounding slopes. The out-buffers associated with these four slopes are evaluated for each of the patches in the intersected patch's 8-neighborhood. For each of the four patches in each "corner" of the 8-neighborhood, the value associated with the eyeray's slope is bilinearly interpolated from the values of the bounding slopes (a). These four resulting amounts are averaged to get a value for the center of the corner areas (b). This procedure results in four values, one at each corner of the intersected patch. The value for the intersection point is then bilinearly interpolated from these four corner points (c).

A great deal of memory may be saved at the cost of view independence if viewing parameters are determined during the preprocess and out-buffers allocated for only those patches and links that are actually utilized in rendering. For the special case of totally diffuse surfaces, all the out-buffers for a given patch will contain the same value. In this case more memory may be saved by substituting a global out-buffer for each patch, and time may be saved by skipping the first interpolation step.

ANALYSIS OF RESULTS

The algorithm was implemented in C on a Convex C1 and an HP9000 series 300. All time figures are given for the

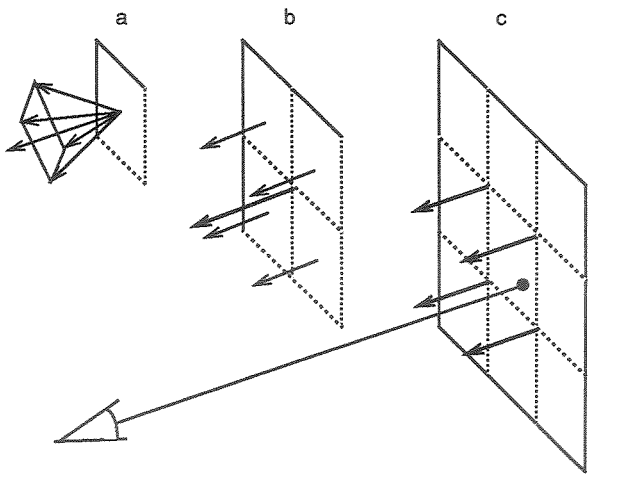

FIGURE 6: Non-diffuse rendering

Convex; the HP workstation time figures were about 4 times slower. Times were measured using the gprof utility in UNIX. All images are 500x500 pixels.

Figure 7 demonstrates the diffuse reflection performance of the algorithm. The light source is a large area located behind the eyepoint. Diffuse reflection from the colored side walls shows on the white back wall. In Figure 8, the light source is the white area at the top of the scene, and the color stripes on the back wall are reflected by the totally specular floor. The irregularities of the reflections are the result of the slight jittering of the ray-object intersections, while the spread of the reflections is a function of the degree of specularity of the floor.

Figures 9 and 10 show the obligatory office scene. This scene was rendered at low resolution, with 2834 patches and 1201 links at each patch. 1.8 million ray-object intersections were calculated in the preprocess, which required 160 seconds for each image. Light sources are the pole lamps, the skylight, the monitor screen, and the tiny lights on the fronts of the machines.

In Figure 9, all the surfaces are diffuse. Memory for the illumination network was 15 Mbytes. The distribution process required 35 seconds, and initialization, eyeray calculation, and rendering took 77 seconds. Note the light reflecting from the desk diffusing onto the wall at low center and onto the chair pedestal. The green from the carpet also diffuses onto the lower wall.

In Figure 10, the carpet has been given a heavy coat of wax. The illumination network took 18.5 Mbytes. The distribution process jumped to 169 seconds. Initialization, eyeray calculation, and rendering took 115 seconds due to the additional time needed for the non-diffuse rendering. Note the reflections from the light areas of the back wall on the floor foreground and the reflections from the side of the desk on the floor background. The desk front also reflects on the floor, and on the far left the light wall reflects in the floor.

EVALUATION OF THE ALGORITHM

While we have only tested diffuse and specular reflectance functions, the method may be used for less ordinary reflectance functions; one example is the reflectance function of an anisotropic surface [7]. Another example is very careful treatment of reflection where different wavelength bands reflect in slightly different directions. This would be implemented with a different reflectance matrix for each color channel.

Different patch and ray-object intersection densities may be needed in different parts of the scene. Very small or highly specular objects will be more accurately rendered if they have been divided into smaller patches, whereas very small patches might result in much unnecessary computation in the case of, for example, a large stretch of blank wall. Sharp shadow boundaries also may require smaller patches. Object size and specularity are known at preprocess time, and objects with appropriate characteristics may be divided into more patches than they would otherwise. Sharp shadows, on the other hand, cannot be easily detected until the distribution process, so this problem would require that the distribution process be adaptive.

The method will also handle transparent objects with no modification: instead of a reflectance function, the object's matrix will be the identity scaled by the object's transmittance coefficient. Translucent objects such as frosted glass may be modeled by jittering the outgoing light direction from the incoming direction by an amount determined by the degree of translucence.

This algorithm may be used on other than planar objects with no loss of speed in the distribution process, since the geometry of the scene has been built into the illumination network at that point. The preprocess depends on the incremental calculation of ray-object intersections for much of its speed, however, and unless an efficient method of calculating these intersections is found the advantages of precompiling the illumination network may be lost.

One very nice feature of the algorithm is that the time and space it requires are determined primarily by the surface area of all the objects in the scene and by the desired resolution, not the number of objects in the scene. The time required for the preprocess is mainly a function of the number of ray-object intersections to be calculated, and the time required for the distribution process usually depends primarily on the number of patches. There are pathological counterexamples to this generalization--for instance, two mirrors facing each other will result in a very long time for the distribution process--but for most scenes it holds true.

FUTURE WORK

Since the geometry of a scene is built into a data structure, an interesting extension of this work involves animation. If for each link an accumulator keeps a total of the amount of light that has passed over the link (out-buffers corresponding to the in-buffers mentioned previously) then we can save information between frames. When an object moves between two frames, we determine the links that are affected and "back" light down the links; that is, light that previously traveled over the links will be negated and sent backwards over the same links. This "negative light" will be run through the inverses of the reflectance functions at each patch, and the resulting (smaller) amounts of negative light are dispatched onto the links. This process continues until convergence.

At this point the scene is exactly as it would have looked had the affected links never been there. The links are then removed and new links are established which reflect the geometry of the updated scene. Light is then released down these new links just as before until convergence and the

scene is rendered. Note that if the vast majority of the links were unaffected by the change, most of the work done for the first frame will have been saved.

An alternative approach might be to determine objects' motion in space-time and calculate several frames' worth of links at a time, in the manner of [5]. This would require prior knowledge of objects' future motion, which the first proposal avoids.

Another area for exploration is the parallelization potential of this algorithm. In the preprocess, the incremental ray-object intersection calculations are both slope-independent and object-independent; slope-object pairs may be taken from a queue by idle processors. In the distribution process, each patch can have its own processor which periodically examines the patch's in-buffers and processes any light it finds there and sends the results out over the patch's links.

To simulate participating media, phantom objects may be inserted into links. The reflection direction of the phantom object is determined by the scattering properties of the medium. The probability of a phantom object being inserted into a link is a function of the length of the link, and its location is a function of the distribution of the medium.

ACKNOWLEDGEMENTS

Our thanks to Emilia Villarreal, A.T. Campbell, and K.R. Subramanian for many helpful technical discussions and suggestions. We would also like to thank the folks at the Center for High-Performance Computing of the University of Texas, in particular Jesse Driver and Dan Reynolds, for their assistance and the use of their machines.

REFERENCES

[1] Arvo, James, "Backward Ray Tracing," *Developments in Ray Tracing (SIGGRAPH '86 Course Notes)*, Vol.12, August 1986.

[2] Arvo, James and David Kirk, "Fast Ray Tracing by Ray Classification," *Computer Graphics (SIGGRAPH '87 Proceedings)*, Vol.21, No.4, July 1987, pp.55-64.

[3] Cohen, Michael F., Donald P. Greenberg, "A Radiosity Solution for Complex Environment," *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol.19, No.3, July 1985, pp.31-40.

[4] Cook, Robert L., Thomas Porter, Loren Carpenter, "Distributed Ray Tracing." *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol.19, No.3, July 1985, pp.111-120.

[5] Glassner, Andrew S., "Spacetime Ray Tracing for Animation," *IEEE Computer Graphics and Applications*, Vol.4, No.3, March 1988, pp.60-70.

[6] Immel, David S., Micheal F. Cohen, Donald P. Greenberg, "A Radiosity Method for Non-Diffuse Environments," *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol.20, No.4, August 1986, pp.133-142.

[7] Kajiya, James T., "Anisotropic Reflection Models," *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol.19, No.3, July 1985, pp.15-21.

[8] Kajiya, James T., "The Rendering Equation," *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol.20, No.4, August 1986, pp.143-150.

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

[10] Nashita, Tomoyuki and Eihachiro Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection,", *Computer Graphics (SIGGRAPH 85 Proceedings)*, Vol.19, No.3, July 1985, pp.22-30.

[11] Ohta, Masataka and Mamoru Maekawa, "Ray Coherence Theorem and Constant Time Ray Tracing Algorithm," *Computer Graphics 1987 (Proceedings of CG International '87)*, ed. T.L. Kunii, pp.303-314.

[12] Wallace, John R., 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.

[13] Ward, Gregory J., Frances M. Rubinstein, Robert D. Clear, "A Ray Tracing Solution for Diffuse Interreflection," *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol.22, No.4, August 1988, pp.85-92.

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