AN ENERGY-BALANCE METHOD FOR ANIMATION

Chris Buckalew and Donald Fussell

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

TR-90-06 April 1990

An Energy-Balance Method for Animation

Chris Buckalew
Donald Fussell

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712

ABSTRACT

An energy-balance method for fast rendering of frames of an animation sequence is described. The technique is based on illumination networks, which was introduced in 1989 as a fast rendering method which could produce both specular and diffuse effects. Since animation sequences are well known to reveal rendering flaws much more readily than single frames, a careful analysis of the sources of error in illumination net style algorithms has been undertaken. This has led to a new algorithm exploiting the concepts of illumination networks which is not only faster at rendering sequences of images than the previous algorithm, but also faster and more accurate on single frames, and better adapted for supporting general reflectance functions. We describe the analysis and the new algorithm and demonstrate its performance on animation sequences and on scenes containing not only specular and diffuse surfaces but also anisotropically reflecting brushed metal surfaces.

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, specular, diffuse, data structure, reflectance function, animation, anisotropic reflection.

1. INTRODUCTION

Rendering scenes realistically is a major focus of research in computer graphics. Beginning with raytracing in 1980 [13] and continuing with radiosity and various other methods, the problem has been attacked by many researchers. Even though these advances have allowed progressively better images to be generated in progressively less time, high quality rendering of globally illuminated scenes is still relatively slow and images can still be distinguished easily from photographs in most cases.

Generating an animation sequence of realistic globally illuminated images is a major challenge given the computational demands for rendering individual frames. In the past, animation has generally been done in a rather straightforward fashion: each frame of an animation sequence is rendered independently and the resulting images are combined in the proper order. This approach ignores an ameliorating characteristic of most animations: the scene ordinarily changes little from one frame to the next. Among the few algorithms designed to exploit this type of frame to frame coherence are [5] and [1].

Unfortunately, these algorithms are designed for specific rendering techniques which limit the scenes for which they produce good results. An animation method for raytracing such as [5] is useful primarily for specular environments, while radiosity method is primarily useful for purely diffuse environments. We were interested in developing a method with two main characteristics: it would take advantage of the frame-to-frame scene coherence described above to a very high degree, and it could render scenes realistically which contain different types of surfaces.

Illumination networks, introduced in 1989 in [2], looked attractive as a basis for two reasons: first, computation of "illumination geometry," that is, what objects in the scene illuminate what other objects, is performed independently from the actual illumination of the scene and the results are stored in such a way that they may be relatively easily modified, and second, I-nets provides the means for implementing many different kinds of reflectance functions.

The I-nets algorithm is a two-part procedure: in the build process, the illumination geometry described above is encoded into a large data structure called an I-net. Surfaces are divided into patches, each of which has a large array of pointers, called links, into its environment which connect it to other patches. These links correspond to a predetermined set of possible ray directions. In the second part of

the algorithm, called the distribution process, light is distributed into the scene along the links attached to light-emitting patches. Incoming light to each patch is mapped to emitted light through the reflectance function associated with the patch's surface. Outgoing light is then distributed into the scene along the links. This process continues until convergence, at which point primary rays are calculated and pixel values are determined.

Two characteristics of the I-nets method are well suited for exploiting the frame-to-frame coherence of animation sequences. First, the links of the I-net provide an easily accessed handle on the geometry of the scene; only the links which are attached to objects which move between two frames must be changed while all others remain intact. A second advantage can overcome the problem that even a small change in scene geometry can precipitate a large change in scene illumination; even if consecutive frames look very similar, every surface is likely to have a slightly different intensity than in the frame before. Scene illumination, then, may need to be completely redone for each frame. This is easily accomplished with I-nets since scene illumination is encapsulated in the distribution process and is not entangled with other work that does not need to be repeated for each frame.

Animation sequences have a way of revealing faults in a rendering algorithm that might pass unnoticed in a single image. To produce good results for animation, then, the sources of error in a rendering method must be carefully analyzed and differences of the resulting image with reality fully understood. Unfortunately, such an analysis reveals that the original I-nets contains several sources of unacceptable error due to its use of link-forming rays which are uniformly separated in slope-intercept ray space. While this method, which we will refer to as *ray-space I-nets* provides good rendering speed, the errors cannot be tolerated for animation. Thus we have developed a new algorithm, *geodesic I-nets*, which not only eliminates these errors, but is also faster than ray-space I-nets, not only for animation, but also for single frames. Moreover, the new algorithm provides a superior method for generating discrete reflectance functions of various types.

In this paper, we will first describe our strategy for using I-nets as a basis for an efficient energy-balance animation method. Then we will describe how I-nets forms an approximation to an ideal energy-balance technique, and show how the approximations of ray-space I-nets are undesirable. We when present the new geodesic I-net algorithm to correct these errors, and describe in detail its operation. Finally, we provide results which indicate the performance of geodesic I-nets on single images with various reflectance functions, including anisotropically reflecting brushed metal, as well as on a short animation sequence.

2. FAST ANIMATION

In many rendering methods the scene geometry is computed at the same time as pixels are colored; in classical raytracing, for example, the complete process of rendering, from determining the nature of the intersected surface to coloring the pixel, is done for each primary ray before going on to the next. This monolithic process makes it difficult to save any information from one frame to the next in an animation sequence.

In the I-nets method, however, determination of scene geometry is decoupled from the lighting and rendering of the scene. Since the I-net data structure encodes the scene geometry, the I-net may be built just once, for the first frame, and modified between frames to reflect the changed scene geometry. Each frame will require a complete distribution process, in which light is spread through the scene and patches' illuminations are calculated, while only the first frame requires the complete build process, in which the I-net is constructed. Ordinarily the build process requires much more computation time than the distribution process, so building the I-net just once for an animation sequence should result in a very large savings.

A method based on I-nets which takes advantage of the frame-to-frame coherence of an animation sequence involves finding a very efficient way to update the I-net between frames of the sequence. The most attractive method of doing this would consist of three steps: 1) determine which patches will move and thus determine and delete the links of the I-net which are affected; 2) move the patches; and 3) create and insert new links which correspond to the new scene geometry. This procedure is attractive because it does not require prior knowledge of patch movement and thus would be suitable for near-interactive usage. Unfortunately the I-net, although large, is but a distillation of much more information that is calculated, used, and discarded during the build process. Modifying the I-net after each frame, then, requires that this extra information either be stored or recalculated, both unpleasant prospects.

Instead, we have chosen to use the extra information where it is already available, in the build process. In one build process we can compute the links for many frames for little more than the cost of computing the links for one frame. The extra links for subsequent frames are stored until needed, at which time they replace outdated links. In this way we avoid storing or recomputing a great deal of information, but we must know the future movement of objects at the time the first frame is calculated.

How can the same build process be used to compute the links for many different frames? The answer is to store a moving object at different times as different object instances, each with a temporal label. For instance, if an object moves during fifteen frames, it will appear in the input file of the build process as fifteen object instances, each with a label of 1 through 15 indicating the frame in which the instance

exists. Basically, then, each link is tagged according to the temporal labels of the object instances it connects, and it is saved until a frame matching its tag is being rendered, at which time the link replaces an outdated link.

Each frame has a file associated with it called its connect file. The connect file for a frame contains links that are to be changed before the frame. These links could be new links to moving objects or old links to be re-established between immobile objects in an area just vacated by a moving object.

In single-frame I-nets, for each ray which intersects the scene, all the ray-object intersections are calculated and stored in order along that ray. Links are created which connect consecutive ray-object intersections whose objects face each other. Each link is stored in the I-net as two pointers, one from each object to the other.

The procedure in the case of moving objects is more complex. In the simplest case, two immobile objects A and B (called permanent objects), have no temporal instances of any objects (called temporary objects) between them. In this case, an ordinary link (two pointers) is established in the I-net just as before. In the case illustrated in Figure 1, a temporary object C, say with temporal label 5, lies between two permanent objects A and B. In this case a link between A and B is inserted in the I-net and in the connect file for frame 6. Another link is also computed, a link between A and C (because they face each other), and this link is stored in the connect file for frame 5. Also stored in frame 5's connect file is a null pointer for B. For frames 1 through 4, then, the link between A and B is active. Before frame 5, however, the pointer from A to B is replaced by a pointer from A to C, the pointer from B to A is replaced by a null pointer, and a pointer from C to A is inserted into the I-net. Before frame 6, the previous link between A and B is re-established, replacing the pointer from A to C with a pointer from A to B and replacing B's null pointer with a pointer from B to A. It is not necessary to replace the pointer from C to A with a null pointer because object C is completely ignored in all frames but frame 5. In the figure links are arrows and active objects are drawn with solid lines while inactive objects are dotted.

Of course other possibilities exist as well. Pseudocode for determining links appears in Table 1.

It should be apparent that the I-net built in this way is just the same as before except that is has some extra temporary objects built into it, each of which is hooked into, and then detached from, the structure by pointers in turn. For the cost of some extra storage and updating of the pointers, any work associated with objects that do not move is saved. The gain is clear–the more objects that do not move, and the greater the number of frames over which to amortize the cost of building the I-net, the greater the savings in incrementing the I-net as opposed to building it from scratch. The drawback to this extension is the extra storage of the temporary objects, and the need for prior knowledge of objects' movements.

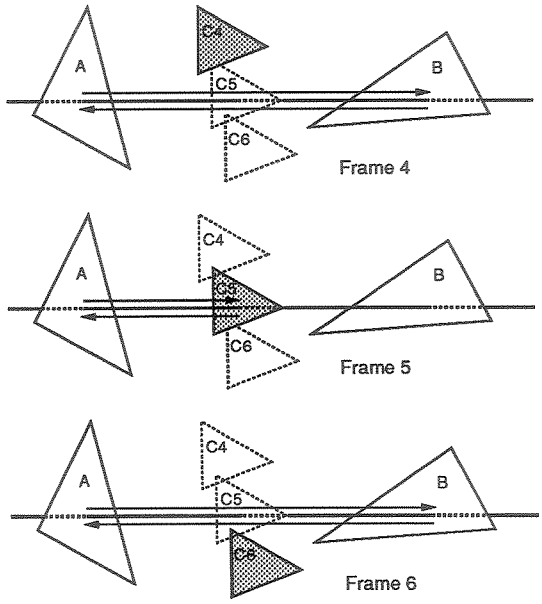

Figure 1: Different links required for the same ray

```
For each ray direction {
  For each individual ray {
    Visiting each ray-object intersection in order {
      If object O1 is permanent and it faces down the ray {
        If there exists another permanent object O2
        further down the ray {
          Insert a link into the I-net between O1 and O2;
          For all temporary objects T_i between O1 and O2 {
            If no object associated with frame i + 1
            lies between O1 and O2 {
              Add this link to connect file i + 1;
            }
          }
        }
      }
      Else O1 is temporary {
        Scan down (or up, depending on which way the object
        faces) the ray until a permanent object or temporary
        object of the same frame is reached and insert a link
        between these two into connect file i;
      }
    }
  }
}
```

Table 1: Pseudocode for determining links

3. GEODESIC I-NETS

Animation stresses rendering algorithms more than still images. Artifacts that do not appear objectionable in a single image may show up clearly in an animated sequence of frames. We now include a brief review of by now familiar radiative heat transfer theory, and discuss the problems of ray-space I-nets. This leads to the development of geodesic I-nets, which will be described in some detail.

3.1. RADIATIVE HEAT TRANSFER THEORY

A familiar scientific basis used to develop and analyze global illumination algorithms is the energy-balance method, adapted from radiative heat transfer theory [10]. In an energy-balance method all the energy in the scene should be tracked: all the light energy that reflects from a surface must be accurately distributed throughout the scene whether it ever reaches the eye or not.

The intensity of light observed at a point on a surface in a given direction is the sum of the intensity emitted by the surface and the intensity reflected by the surface in that direction. This is called the directional intensity, and the preceding statement can be written

$$R_\phi = \epsilon_\phi + r_\phi \tag{1}$$

where R_ϕ is the outgoing intensity from the point in direction ϕ, ϵ_ϕ is the emitted intensity in that direction, and r_ϕ is the reflected intensity in the same direction.

Suppose dA_1 is a differential patch of area on surface A_1, as shown in Figure 2. Define the intensity emitted from dA_1 in direction o to be $\epsilon_{\phi,1}$

The energy a surface receives is a function of the intensity, I, of the incoming light, the solid angle $d\omega$ through which it arrives, and the projected differential area on which it is incident. The total energy per unit area incoming from direction α is given by

$$E_\alpha = I_\alpha cos\theta d\omega \tag{2}$$

where I_α is the incident intensity from direction α, $d\omega$ is the solid angle through which it arrives, and $\cos\theta$ is the projected unit area (θ is the angle between the surface normal and α). Now the intensity reflected in direction ϕ from energy incident from direction α is given by

$$R_{\phi,\alpha} = \rho_{\phi,\alpha} E_\alpha \tag{3}$$

$$= \rho_{\phi,\alpha} I_\alpha \cos\theta d\omega \tag{4}$$

where $\rho_{\phi,\alpha}$ is the bidirectional reflectance of the surface for input direction α and output direction ϕ. Suppose a scene consists of two surfaces A_1 and A_2, as shown in Figure 3. In order to determine the outgoing intensity observed at differential area dA_1 in direction ϕ_1, we need both ϵ_ϕ and the intensity from surface A_2 reflected by dA_1 in direction ϕ. The latter is given by integrating Equation 1 over the solid angle $\Omega_{1,2}$ subtended by A_2:

$$r_{\phi,1} = \int_{\Omega_{1,2}} \rho_{\phi,\alpha,1} I_{\alpha,1} \cos\theta_1 d\omega_1 \tag{5}$$

Since the incident intensities I_1 on surface dA_1 are the same as the outgoing intensities R_2 from surface A_2, then

$$I_{\alpha,1} = R_{\phi,2} \tag{6}$$

and

$$r_{\phi,1} = \int_{\Omega_{1,2}} \rho_{\phi,\alpha,1} R_{\phi,2} \cos\theta_1 d\omega_1 \tag{7}$$

Now the total intensity observed in direction ϕ_1 from dA_1 is

$$R_{\phi,1} = \epsilon_{\phi,1} + \int_{\Omega_{1,2}} \rho_{\phi,\alpha,1} R_{\phi,2} \cos\theta_1 d\omega_1 \tag{8}$$

Similarly, the intensity observed in any other direction from dA_1 and intensities associated with differential areas on A_2 may be defined. Of course, in a real scene, the directional intensity from a point is derived from all the surfaces that are visible from that point, so the above equation must be integrated over all these surfaces.

Equation 8 may be made computationally tractable by the use of several approximations. In hemi-cube methods, as well as in the I-nets technique, the surfaces are discretized into patches and the spectrum of light is divided into independent bands. The methods differ in their treatment of incoming and outgoing light directions. In traditional radiosity methods, the hemisphere above a patch is divided into a set of solid angles determined by a hemicube; these directions are unique to the patch orientation, and the patches visible in each of these directions must be determined. In the I-nets method, the set of all possible directions is approximated by a small set of directions which is used globally for all patches. Also, for each direction, a set of rays is chosen; any light which travels in that direction is transferred over one of these rays.

3.2. PROBLEMS WITH RAY-SPACE I-NETS

Unfortunately, ray-space I-nets suffers from some defects which can lead to unacceptable errors. Two of these defects

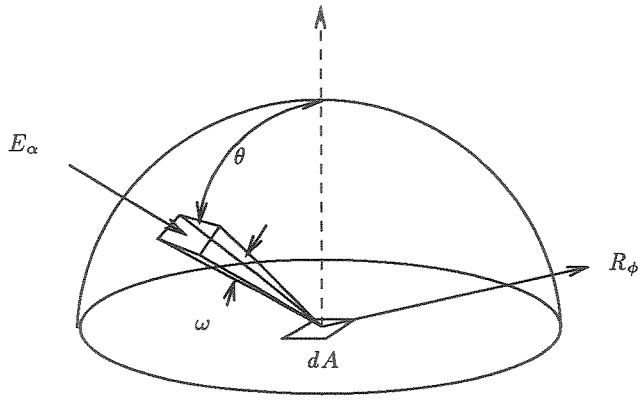

Figure 2: Hemisphere of directions to a differential area

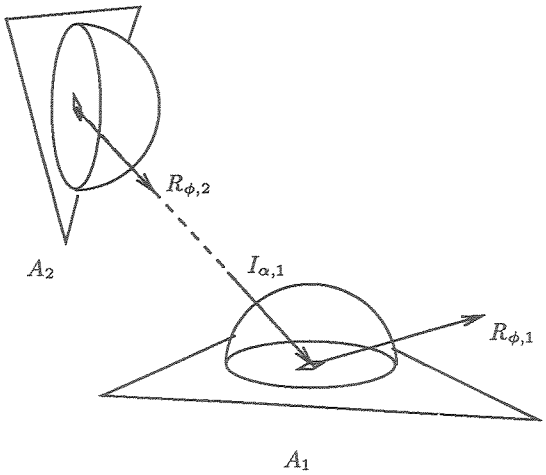

Figure 3: Outgoing intensities become incoming intensities

pertain to the integral of Equation 5 where energy incident on a patch from all directions is approximated by the summation over all possible ray directions of the energy incoming on those rays. The I-nets algorithm defines a ray space in terms of slopes and intercepts; both dimensions of this ray space are sampled at regular intervals to obtain the set of rays used to create links. The sampling uniformities in each of these two dimensions are the sources of these first two defects.

The first of these defects is that the spacing of rays for all directions is determined by points on a common intercept plane; rays at any angle θ to the intercept plane normal pass through the same points as rays normal to the plane, as shown in Figure 4. As a result, oblique rays are more dense by a factor of $1/\cos\theta$ than normal rays. Thus if each ray carries a unit amount of energy, more energy per unit area is transferred to a surface perpendicular to the angle θ rays than to a surface perpendicular to the normal rays.

The second of these problems is the nonuniform spatial distribution of ray directions resulting from the uniform sampling of slopes in ray space. Unfortunately, the angles between these directions vary quite a bit: they are small for the directions near the vertices of the unit cube about the origin and larger toward the middles of the faces as shown in Figure 5. As a result, much more energy is transferred in those directions where the ray directions are "bunched up" in space (where the angles between them are small) than in those directions where the rays are more sparse.

The ray-space I-nets algorithm also assumes that all patches are the same size so that the bidirectional reflectance term of Equation 3 will be the same for all patches of an object. This is not true in the case of small patches or patches along the edge of an object. Finally, while the matrix implementation of reflectance functions used in ray-space I-nets provided a means of implementing a large number of reflectance functions, in general the technique used would have required a large matrix for each polygon in the scene. For mixtures of purely diffuse and specular illumination, this potential expense was eliminated, but the general case was inflexible and potentially expensive.

These four main defects limit the utility of the ray-space I-nets algorithm for animation purposes. Thus, a new type of I-nets algorithm which eliminates these problems while retaining the performance of ray-space I-nets is required.

3.3. RAY NONUNIFORMITIES

To overcome the nonuniformity of the distribution of ray directions, the ray space of the original I-nets algorithm was abandoned. Instead, the set of possible ray directions is determined by tessellating the unit sphere with nearly equilateral triangles; the vertex points serve as endpoints of unit vectors from the origin. This set of unit vectors defines the possible ray directions. Since the triangles used to tessellate

the unit sphere are all the same size, these directions are evenly distributed.

The variable spacing density of rays due to their angle with the intercept plane has been corrected in the new algorithm by spacing the intersections of the rays with the intercept plane by an appropriate amount. To separate the rays which have the same direction D evenly in space, it is necessary to separate their intersections with the intercept plane by a factor of $1/\cos\phi$, where ϕ is the angle between the D and the normal to the intercept plane.

It should be noted at this point these two improvements now account for the cosine factor of Equation 6. In traditional radiosity methods, this factor is dealt with in formfactor calculations, but in geodesic I-nets the cosine factor is implicit in the distribution of rays. Suppose the separation distance of two adjacent rays which have the same direction is δ. This distance will be the same for each ray direction because of correction (2) above. In the direction normal to a surface S, then, the intersections of rays with S will be separated by this distance δ. For a direction D forming an angle θ with the normal, rays will intersect S at a separation of $\delta/\cos\theta$. Thus if the spatial density of rays which intersect S from the normal direction is 1, then the spatial density of rays which intersect S from direction D will be $\cos\theta$. This is shown in Figure 6.

3.4. VARIABLE PATCH SIZES AND BUCKETS

The formfactors of traditional radiosity also contain the reflectance function and patch size information. In I-nets, all patches of a surface have the same size. Since the intensity-to-energy ratio of the bidirectional reflectance function varies with patch size, this property allows the patches' reflectance functions to be the same for all the patches of the surface. For patches that differ in size, however, this assumption is not true. The geodesic algorithm addresses this problem by forming an estimate of patch size based on the number of ray intercepts, H_p, it receives. This estimate is compared to the average number of ray intercepts, H_u, that a unit-sized patch receives. The ratio H_p/H_u is used as a multiplier during the distribution process: the reflectance function for the object is attenuated by this ratio for each patch.

Yet another problem addressed by the geodesic algorithm which enhances its accuracy is the elimination of the jittering of ray-object intersections which was used in ray-space I-nets to hide the artifact termed "plaiding" in [12]. The basis of the problem is that, although the total amount of light energy delivered to a surface by links having angle θ with the surface's normal (because of the intersections' $1/\cos\theta$ distribution over the surface) is correct, it is concentrated in large amounts on just a few patches, whereas it should be spread more evenly over larger numbers of patches. This is shown in Figure 7. Jittering alleviated this problem to some extent by setting each link randomly within the larger area, but the light delivered by that link remained concentrated

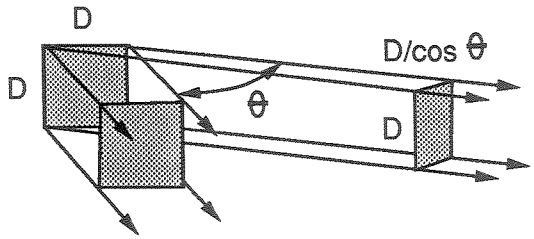

Figure 4: Common intercepts yield different ray densities

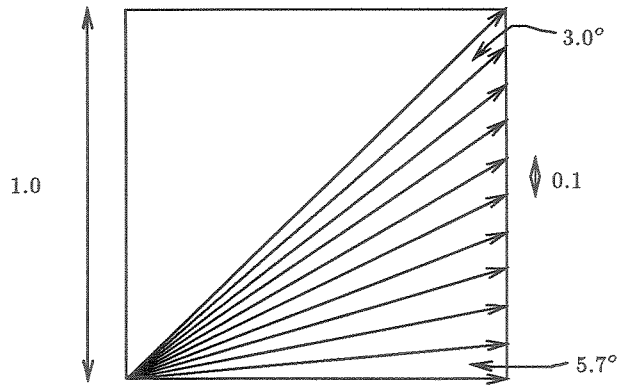

Figure 5: Uniform slope interval yields different ray densities

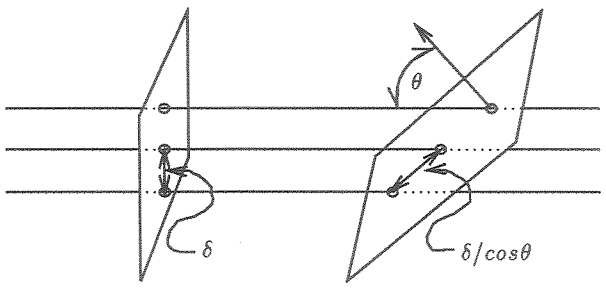

Figure 6: Built-in cosine factor

at that patch.

We have solved this problem by associating a small set of "buckets" with each patch. Basically each bucket defines a certain area over which light will be spread. Light which falls in bucket 0 will be spread over very little area, while light which falls in bucket N will be spread evenly over many neighboring patches. Each patch has N buckets, and during the build process each ray direction is assigned to the appropriate bucket. For instance, a link whose direction is nearly normal to the patch will be assigned to bucket 0, so that light arriving on that link will be spread very little, while a link whose direction is nearly parallel to the patch will be assigned to bucket N, and light arriving on that link will be spread over a large area. For greatest accuracy this spreading should occur periodically during the distribution process.

Another use for the buckets is in compensating for the length of links. Energy transmitted to the end of a link should be spread over an area proportional to the square of the length of the link. Light travelling over short links should not be spread much, while light which travels over long links should be spread over a greater area. When a link is set up between two surfaces, the distance between the surfaces as well as the incident angles is used to determine which bucket each end of the link must be associated with–if the surfaces are far enough apart to warrant a "wider" bucket than the incident angle indicates, the bigger bucket is used.

3.5. RENDERING

Rendering of patches is done in much the same manner as described in [2], with one important difference. To render diffuse patches, both methods bilinearly interpolate center intensity values. In ray-space I-nets nondiffuse patches were rendered one primary ray at a time; for each primary ray values for neighboring ray directions were interpolated to arrive at a nondiffuse intensity amount for that primary ray. This value was then added to the patch's diffuse intensity amount (interpolated according to the intersection point on the patch) to get the pixel value. Each primary ray that intersected a nondiffuse surface resulted in a significant amount of computation.

To reduce this computation the new method calculates a nondiffuse intensity for each patch for one primary ray only, that connecting the eye to the center of the patch. This intensity is then added to the patch's diffuse intensity, and rendering then proceeds just as in the diffuse case. This approximation is only valid when the patch subtends a small area of the screen and the differences between the primary ray to the center of the patch and all other primary rays which hit the patch are small.

In addition to reducing the computation time for rendering, this procedure allows shaded polygons representing the patches to be sent into a graphics pipeline which transforms, Z-buffers, Gouraud shades, and outputs them to the screen. Rendering is thus decoupled from actual pixel shading, which significantly reduces the computation time for rendering scenes containing nondiffuse surfaces.

3.6. IMPLEMENTING DIFFERENT REFLECTANCE FUNCTIONS

Conceptually, I-nets can handle any type of reflectance function that can be described by an $N \times N$ matrix, where N is the number of ray directions. The (i, o)th entry in the NxN matrix gives the fraction of light which arrives from direction i which leaves in direction o. Such matrices can be large, and different ones are needed for different patch orientations in space. Fortunately, the entire matrix can be collapsed into a constant in the case of a diffuse reflectance function, and specular reflectance functions result in a sparse matrix that can also be expressed more compactly. In addition, if an object is planar, only one matrix is needed for the whole object, rather than one for each patch of the object. However,there is clearly a need for a compact, accurate, easily-generated description for general reflectance functions.

3.6.1. THE PROTOTYPE REFLECTANCE FUNCTION

We begin with the idea of a "prototype" reflectance function. Suppose the surface of interest lies in the xz-plane facing upwards. Light can travel in the directions determined by the predetermined set of possible ray directions, and the light arriving from direction i and reflecting from the surface in direction o can be described by a formula in terms of the orientation of the surface and the orientations of directions i and o with respect to the surface. Given a unit amount of light arriving from one of our ray directions i, we can use the reflectance function formula to determine the amount of light leaving the surface in all possible ray directions. Thus for each ray direction, we can construct a list of all directions in which a non-zero amount of light reflects and the amount of such reflection (for practical purposes non-zero means a number larger than a certain threshold). These amounts are called factors. The collection of such lists for each possible ray direction comprises the prototype reflectance function.

Several characteristics of the prototype reflectance function should be clear. The first is that it is merely the evaluation of the reflectance function formula over a given set of ray directions (although it could be obtained by other means, such as experimental evaluation or manual fiddling). As such, the type of reflectance function modeled is not independent of the ray direction resolution: if the reflectance function has very fine features, the number of ray directions must be large enough that the sampling of the prototype reflectance function is adequate to catch these features. A good example of the type of reflectance function that this method cannot handle is a perfect mirror; 100% of light incident from direction I is reflected only in direction R and if the set of ray directions does not happen to include direction

R, the prototype reflectance function reflects no light at all. If the reflectance function does not require infinite resolution, however, an appropriate resolution for the number of ray directions may be determined.

Second, the prototype function can be every bit as large as the $N \times N$ matrix it replaces if there are few non-zero entries. Third, its calculation is an $O(N^2)$ process, since the reflectance function must be evaluated twice for each pair of ray directions. Finally, it is easy to construct for a wide variety of reflectance functions and is equivalent to the $N \times N$ matrix.

For many reflectance functions, the derived prototype reflectance function is not dependent on the colors of objects. If the reflectance function behaves the same for each of the red, green, and blue bands, then a generic factor may be stored in the prototype reflectance function instead of three separate factors for each of the three bands. To get the proper reflectance function, then, the generic factor must be multiplied by the object's color amounts whenever the prototype reflectance function is used. This space-saving tactic can't be used when the three bands reflect differently, as in the case of the reflectance function for a diffraction grating. In this case all three factors must be stored.

3.6.2. THE ACTUAL-TO-PROTOTYPE MAP

If the same reflectance function were to be used for many surfaces all with different orientations, the N^2 construction costs and storage for a prototype reflectance function for each orientation would be prohibitive. Instead, we use the uniformity of the distribution of ray directions to advantage and construct for each different orientation a mapping to the original prototype reflectance function. If S is the set of possible ray directions, the mapping is a one-to-one and onto function from S to S. This mapping is called the actual-to-prototype map. In this way we need only construct *one* prototype for each distinctive reflectance function. Then for different orientations incoming light will be mapped to appropriate directions using the actual-to-prototype map and the prototype reflectance function applied to that incoming light to produce outgoing light. For each direction the prototype-to-actual is again used to retrieve the outgoing light to be sent out over the links.

Given a prototype reflectance function and an arbitrary orientation for which to construct an actual-to-prototype mapping, we need to determine the relationships between two pairs of vectors: the normal vectors and the rotation orientation vectors for both the prototype orientation and the actual orientation. The rotation orientation vector gives the alignment of some surface feature; for instance, the grain direction of brushed metal. For some reflectance functions, such as specular reflectance, this vector is not necessary–rotation of the surface about its normal will not change the prototype reflectance function.

The actual-to-prototype map is constructed by first mapping the ray direction nearest to the actual normal to the ray direction nearest to the prototype normal. Since the ray directions are derived from a tessellation of nearly equilateral triangles, each ray direction has six neighboring directions. Using this relationship between the directions, each successive concentric "ring" of ray directions away from the actual normal is mapped to the corresponding ring of ray directions away from the prototype normal, using the rotation orientation vector to align the ring. Figure 8 illustrates this process. For each ring of the actual orientation, the ray direction closest to the rotation orientation vector (which is the one which has the largest dot product with the ROV) is labeled "1" and the remaining directions in the ring are consecutively numbered counterclockwise. This same procedure is done for the corresponding ring about the prototype normal. Direction 1 for the actual ring is now mapped to direction 1 in the prototype ring, actual direction 2 to prototype direction 1, and so on.

During the distribution process, incoming light accumulates in the in-buffers for each link connected to a patch. To calculate the light reflected from a given incoming direction with index i_a, the actual-to-prototype map is used to find the appropriate incoming index i_p on the prototype surface. Now using the prototype reflectance function, the indices of the reflected directions give locations in a scratch work area where the amount of the light coming from i_p multiplied by the factors given in the prototype reflectance function are accumulated. This is done for all possible ray directions for the patch. Now for each possible ray direction o_a, the corresponding location in the work area is found with the actual-to-prototype map and the amount of light at that location is added to o_a's out-buffer and sent out over that link. This process is shown in Figure 9.

It should also be noted that the prototype reflectance functions need not be calculated every time the program is run. Quite likely a small library of reflectance functions will be used for many different scenes. These may be computed once and stored on disk for future use. Of course, the actual-to-prototype maps must be computed for each different scene since objects will likely have different orientations (although common orientations such as axis alignment may be saved as well). However, since the bulk of the work is in computing the prototype reflectance functions, re-using these will result in significant savings.

4. RESULTS

The method has been implemented in C and runs on a variety of UNIX machines. The data presented here is the result of runs on a MIPS RC6280 computer with 32M of memory. Times were determined with the *prof* utility of UNIX.

Image 1 shows the first six frames of a four second animation sequence consisting of 60 frames. The blades of the ceiling fan are turning and the heat lamp is swinging in a circle over the cabinet. This moving light source is particularly

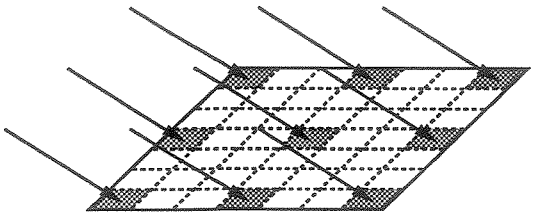

Figure 7: Concentration of light on small areas

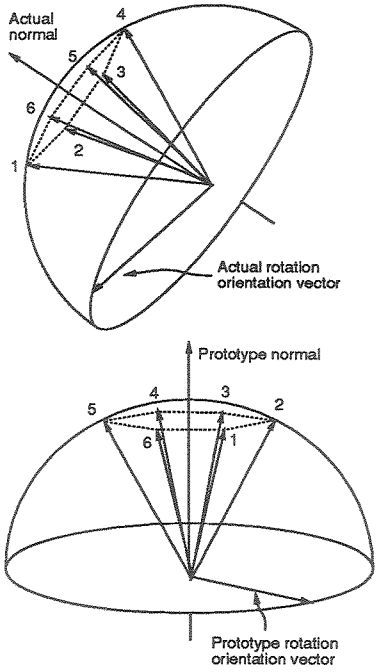

Figure 8: Constructing the actual-to-prototype mapping

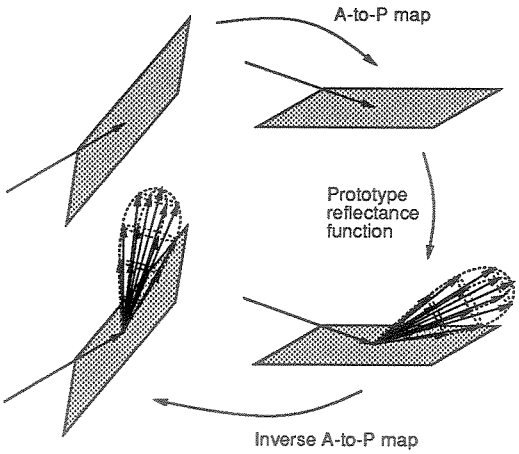

Figure 9: Using the actual-to-prototype map

taxing for animation techniques because the lighting over the entire scene changes even though relatively few objects move. Note for example how the floor under the light gets redder when the light swings over it. More subtle changes not readily visible are even more widespread over the scene.

All surfaces are diffuse to conserve memory. The I-net for this scene occupies 6M of memory. Generating each frame from scratch requires about 8 seconds per frame; 6.4 seconds for initialization and the build process and 1.6 seconds for the distribution process and rendering. The animation procedure was applied to two sequences of 30 frames each. For each sequence the initialization and build process require 16.8 seconds (more than for a single frame because of the extra temporal objects that must be processed) while each frame requires 1.62 seconds for the distribution process and rendering (also more than for a single frame due to the time necessary to update the I-net between each frame). The extra storage required for the temporal objects amounts to 4.6M of memory. For sixty frames, then, generating each one from scratch takes 8 seconds per frame, while the animation method takes 2.18 seconds per frame, a speedup of almost 4 times.

Image 2 shows the same scene except the floor has a broad specular component. This scene requires 7 seconds for initialization (including computing the prototype reflectance function) and the build process and 12 seconds for the distribution process and rendering. The I-net occupies 8.4M of memory.

Images 3 and 4 demonstrate the anisotropic reflectance function of brushed metal. The light source illuminates the flat surface outlined in green. The arrow in the right corner of the surface indicates the direction of the striations of the brushing. The same surface is shown at orientations of 0, 45, 67.5, and 90 degrees relative to a line connecting the eye to the light source. The surface is divided into 364 patches, each of which samples light in 1261 directions. The I-net occupies 7.5M of memory and each image was generated in 9.3 seconds (including computing the prototype reflectance function). Notice that the "horns" of the reflections rotate with the surface, with the back horn diminishing and eventually disappearing when the striations are perpendicular to the eyerays. For comparison, Images 5-8 are photographs of an actual brushed surface in the same orientations. The actual surface has a diffuse component which appears as a "halo" in the photographs; this diffuse component is not modeled in the generated images.

5. CONCLUSIONS

Illumination networks is a technique which is inherently well-suited to efficiently render animated scenes in which the animation has been precomputed due to its explicit representation of illumination geometry and its separation of the tasks of evaluating this geometry and distributing light through a scene. Unfortunately, however, the ray-space algorithm

for I-nets presented in [2] contained inaccuracies which resulted in unacceptable artifacts when used for animation. Analysis of the sources of these artifacts have led to the development of a new I-nets algorithm, which we call geodesic I-nets. This algorithm provides much more accurate results than ray-space I-nets, and with greater efficiency. It also provides a very effective means of implementing a wide variety of reflectance functions for the surfaces rendered.

6. ACKNOWLEDGEMENTS

We would like to thank Emilia Villarreal, Per Hanssen, K.R. Subramanian, Don Speray, Kelvin Thompson, and A.T. Campbell for many helpful technical discussions and suggestions. Thanks to Per Hanssen also for developing most of the reflectance function and ray distribution code. Special thanks are due to MIPS Computer Systems, Inc, especially Jeff Carruth, John Beck, Margaret Bailey, Ken Robertson, and Bill Jobe for their helpfulness and the loan of one of their computers. Also we would like to thank the Center for High-Performance Computing of the University of Texas System, especially Jesse Driver, for their help and the use of their machines.

REFERENCES

[1] Baum, Daniel R., John R. Wallace, Michael F. Cohen, Donald P. Greenberg, The Back-Buffer: An Extension of the Radiosity Method to Dynamic Environments, *The Visual Computer*, Vol. 2, No. 5, pp. 298–306.

[2] Buckalew, Chris and 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.

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

[4] Cohen, Michael F., Shenchang Chen, John Wallace, Donald P. Greenberg, A Progressive Refinement Approach to Fast Radiosity Image Generation, *Computer Graphics* (SIGGRAPH '88 Proceedings), Vol. 22, No. 4, August 1988, pp. 75–84.

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

[6] Goral, Cindy M., 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.

[7] Immel, David S., Michael 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.

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

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

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

[11] 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.

[12] Wallace, John R., 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.

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