Adaptive Mesh Generation for Global Diffuse Illumination

A. T. Campbell, III, and Donald S. Fussell

Department of Computer Sciences

The University of Texas at Austin

Austin, TX 78712

ABSTRACT

Rapid developments in the design of algorithms for rendering globally illuminated scenes have taken place in the past five years. Net energy methods such as the hemicube and other radiosity algorithms have become very effective at computing the energy balance for scenes containing diffusely reflecting objects. Such methods first break up a scene description into a relatively large number of elements, or possibly several levels of elements. Energy transfers among these elements are then determined using a variety of means. While much progress has been made in the design of energy transfer algorithms, little or no attention has been paid to the proper generation of the mesh of surface elements. This paper presents a technique for adaptively creating a mesh of surface elements as the energy transfers are computed. The method allows large numbers of small elements to be placed at parts of the scene where the most active energy transfers occur without requiring that other parts of the scene be needlessly subdivided to the same degree. As a result, the computational effort in the energy transfer computations can be concentrated where it has the most effect.

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, mesh-generation, diffuse, data structure, incremental.

1. INTRODUCTION

Accurate modeling of illumination has long been a goal of computer graphics. Until ten years ago, local illumination was the only factor generally considered. The development of ray tracing for handling global specular

reflection and refraction [17], and of net energy techniques for handling global diffuse illumination [8] [3] [11] [12] led researchers to realize that these effects could and should be modeled, although the computational burden of doing so can be great.

While great strides have been made in improving the efficiency of ray tracing for specular illumination, extensions of ray tracing to handle diffuse lighting effects remain quite expensive [6] [16]. Net energy methods such as hemicube and other types of radiosity algorithms are better suited to global diffuse illumination both because the energy distribution throughout the scene can be computed in a view-independent way and because such techniques as progressive refinement [5] can make the production of approximate solutions reasonably efficient.

Methods for computing the energy balance in globally illuminated scenes in which all surfaces are not perfect specular reflectors involve the numerical integration of a large system of highly interrelated equations [9]. Ray tracing techniques do this using Monte Carlo methods, while the most popular net energy methods based on hemicubes essentially use the rectangle rule for integration. The latter process consists of two steps. First, the surfaces in the scene are subdivided into a number of relatively small surface patches of equal size. Following this, a variety of algorithms have been used to distribute the energy initially available from light emitting surfaces throughout the scene. In some cases, energy distribution algorithms have incorporated techniques to refine the initial surface subdivision as needed in regions of significant intensity gradation.

The major step in the development of adaptive refinement of surface meshes was taken with the development of patch-element radiosity computations in [4]. This technique subdivides energy receiving surfaces wherever an initial calculation indicates significant intensity gradations are occurring. The new subdivided surfaces, called elements, are used to refine form-factor calculations for larger patches, thereby allowing more accurate illumination computations to be performed at reasonable cost. While this method was a great step forward in enabling effective global illumination computations to be performed, it suffers several drawbacks. First, since shadow boundaries are never explicitly determined, the ability of the initial patch calculations to detect areas of intensity gradation where shadows occur is critical. Small objects casting shadows can easily be missed if the initial patch sizes are too large. Second, since emitting surfaces are not

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.

Figure 1: Light Leaking

subdivided, accurate soft shadowing is not done. Finally, the subdivision of receiving polygons into a regular grid can lead to the phenomenon of “light leaks” where the grid partitions along a surface do not coincide with light-occluding polygons abutting the surface and thus allow light to leak past the occluding polygons, as shown in Figure 1.

Besides these mesh-related problems, the matrix/hemicube method used suffered from problems with time and memory requirements and aliasing of the results caused by the use of hemicubes. Recent developments have resulted in great improvements in these latter areas. Speed has been improved by replacing the original technique of simultaneously solving all the integral equations in the system by a progressive refinement technique which quickly computes a good approximation and then gradually refines it to converge on the solution, as well as by exploiting the capabilities of many machines to perform hardware depth-buffer calculations [5]. This technique involves distributing energy from sources throughout the scene in decreasing order of source intensity. As a result, hemicube aliasing problems and errors caused by the violation of form-factor approximation assumptions are more severe than those of matrix-solution methods [1] [15]. Moreover, it is difficult to adaptively refine a surface mesh using this technique without sacrificing much of the performance advantage gained.

In [15], a ray-tracing radiosity computation technique was introduced which alleviates hemicube aliasing and seems better suited to adaptive mesh refinement. This method gives improved results in the computation of penumbra effects if the initial mesh is fine enough. However, the use of point sampling to determine visibility in form-factor calculations still suffers from an inability to detect errors caused by missing shadows cast by objects small relative to the initial patch size.

In this paper we present a new radiosity approach for global diffuse illumination which automatically generates an initial surface mesh by efficiently subdividing input surfaces along shadow boundaries. This initial mesh is then adaptively refined using both further shadow boundary subdivision and intensity-gradient refinement. Radiosity computations are done using a ray-tracing

approach like that of [15], although ray tracing is not required for visibility calculation due to the shadow boundary subdivision. In contrast with existing radiosity techniques, the surface elements created are rarely rectangular. This technique alleviates the problems of potentially missed objects and light leaks.

The following section reviews the essential theory and equations of illumination necessary for presentation of this work. Section 3 presents the algorithm in full detail. Our implementation and results are detailed in Section 4, including test data, timing figures, and analysis.

2. SURFACE SUBDIVISION

Radiosity approaches to global diffuse illumination computation involve the solution of a large system of integral equations which express the energy transfers among all the surface elements in an energetically closed environment. These equations take the form

$$B_i A_i = E_i A_i + \rho_i \sum_{j=1}^n B_j A_j F_{ij} \quad (1)$$

where B_i is the *radiosity* (radiated energy per unit area) of surface i , A_i the area of the surface, E_i the emitted energy per unit area, ρ_i the reflectance of the surface, and F_{ij} the form factor from surface j to surface i . The *form factor* F_{ij} gives the fraction of energy leaving surface j that arrives at surface i , and is given by the equation

$$F_{ij} = \frac{1}{A_j} \int_{A_j} \int_{A_i} \frac{\cos \phi \cos \theta V}{\pi r^2} dA_i dA_j \quad (2)$$

where V is a boolean which is 1 when dA_j is visible to dA_i and 0 otherwise, and all other terms are as shown in Figure 2.

The system of equations in 1 can be solved to an arbitrary desired accuracy for the radiosities of the surface elements if we know the form factors, reflectances, and areas accurately. This requires solving the integral equations in 2, or, when this is impossible, finding reasonable approximate solutions to them. Approximation can be done by assuming that the finite surface elements are very small relative to their distances apart, and that the surfaces are either entirely visible or entirely invisible to each other. Under these conditions, V can be determined by a simple hidden surface computation, and all other terms in the integrand become constants across the surfaces involved, so that 2 reduces to

$$F_{ij} = A_i \frac{\cos \phi \cos \theta}{\pi r_{ij}^2} \quad (3)$$

for visible surfaces and 0 otherwise. For any interesting scene, these assumptions do not hold for input polygons.

Figure 2: Form Factor Geometry

Therefore, these surfaces are broken up into smaller patches for which the assumptions are reasonable for most pairs of patches. Even in this case, the assumptions are unlikely to be reasonable unless the number of patches generated is unreasonably large, so a better approach is to numerically approximate the integrals. This can be done by breaking the source patch into n pieces of size ρA_j and the receiving patch into m pieces of size δA_i , so that 2 may be approximated by

$$F_{ij} = \frac{1}{A_j} \sum_{k=1}^n \sum_{l=1}^m \frac{\cos \phi_{kl} \cos \theta_{kl} V_{kl}}{\pi r_{kl}^2} \delta A_i \delta A_j \quad (4)$$

in the general case. If only the source patch needs to be subdivided to make the assumptions reasonable, then this simplifies to

$$F_{ij} = \frac{1}{A_j} \sum_{k=1}^n \frac{\cos \phi_{ki} \cos \theta_{ki} V_{ki}}{\pi r_{ki}^2} \delta A_j \quad (5)$$

and likewise to

$$F_{ij} = \frac{1}{A_j} \sum_{l=1}^m \frac{\cos \phi_{jl} \cos \theta_{jl} V_{jl}}{\pi r_{jl}^2} \delta A_i \quad (6)$$

when only the receiving patch needs subdivision.

The patch-element adaptive mesh subdivision used with matrix and progressive refinement solution methods use primarily the receiver subdivision approach of 6, although in some difficult cases sources are subdivided also. The ray tracing method of [15] can replace subdivision with point

sampling in the latter case. Indication that refinement of initial patches into elements is needed is obtained by measuring the intensity gradient between neighboring vertices in the mesh after initial radiosity computation, and then subdividing as necessary until a desired gradient threshold is reached. This works well as long as the initial mesh of patches is sufficiently dense, but if shadow boundaries are missed in the initial step, further subdivision will not take place. If in spite of subdivision individual elements extend to both sides of abutting polygons, light leaks are also possible.

Our approach not only refines the mesh automatically, but generates the initial mesh automatically as well. We consider the input polygons to be the initial mesh and combine shadow boundary determination with intensity and gradient criteria to steer further subdivision. As a result, no shadow boundaries are missed, light leaks are prevented, and the shape of the mesh elements more closely follows final isointensity gradations. The method is based on Equation 4, where both source and receiving elements are subdivided.

3. ADAPTIVE MESH COMPUTATION

As we have suggested, the initial mesh for the algorithm is a set of input polygons. Our strategy is to process these in decreasing order of available energy, meaning that the most energetic light source is the first polygon processed. The algorithm proceeds much like the ray-tracing progressive radiosity technique of [15]. No hemispheres are used for form-factor computation. Rather, rays are cast between emitting and receiving elements to estimate form-factors between these elements. These rays are not, however, used for visibility computation.

Once selected, a light source polygon is tested to ensure that it is small enough and far enough away from the rest of the scene that no penumbra calculations are necessary. If this is not the case, the light polygon must be subdivided into pieces which meet these criteria.

Visibility is determined through the use of a BSP-tree based shadow algorithm similar to that of [2]. As a source element is being processed, each of the other polygons in the scene is considered in turn. These are sorted in order of increasing distance from the centroid of the source using the BSP tree. The first step in the processing of one of these polygons is to determine whether any of the polygons closer to the source casts a shadow on the polygon under consideration. If so, the polygon is split along the shadow boundary (note that sharp shadows generated using the source centroid as a point light source are used). Once this splitting is complete for all potentially shadowing polygons, the original polygon has been subdivided into a set of receiving elements, each of which can be considered totally visible or totally hidden from the source.

Once the visibility determination is complete, form factors for each of the visible receiving elements are calculated. If these differ by more than a user-specified tolerance, the re-

```

(* cast shadows and compute illumination *)
Procedure IllumWorld(NumPolys, PolyList)
Begin
  Tree := BspBuild(NumPolys, PolyList);
  While EnergyRemains(Tree) Do Begin
    Src := FindBrightest(Tree);
    BspTraverse(Tree, Src, FRONTTOBACK,
      NumRcv, RcvPolys);
    DivSrc(Src, RcvPolys[1], NumSrc,
      SrcPieces);
    For i := 1 To NumSrc Do Begin
      Tree := SbspShadow(SrcPieces[i],
        NumRcv,
        RcvPieces);
      IllumPieces(Tree);
    End;
    Dissipate(Src)
  End
End;

```

Figure 3: Overview of Algorithm

ceiving element is subdivided and form factors recalculated for each resulting element until the tolerance is met. These form factors are normalized to ensure energy balance and are then used to compute radiosities and sample the incoming energy at each receiving element vertex. This provides an intensity for each vertex, which can be used as a basis for Gouraud shading if it is desired that the image be rendered as the mesh is progressively determined.

Further subdivision may be deemed necessary if the intensity variation across the element exceeds a user-supplied threshold. In such a case, the receiving element is subdivided along its largest dimension, and the energy estimation step is repeated until the variations for all the elements involved are below the threshold. Note that after each illumination step of our method, the variation limit is maintained. This is in contrast to the methods of [4], in which an entire coarse global solution is performed before any element subdivision is involved.

After illumination, we add the current polygon to the list of potential shadowing polygons and move on to the next candidate for illumination.

When one source element has fully scattered its light, we repeat the process for the remaining source elements on the current light source polygon. Once the polygon has distributed all its available energy, we move to the next brightest source polygon and repeat the process. This is continued until the total untransmitted energy has converged to zero within some tolerance. This tolerance is a user-supplied fraction of the input energy.

These steps are summarized in Figure 3.

We will now discuss each of these steps in greater detail.

Figure 4: BSP Tree Example

3.1. INITIALIZATION AND DATA STRUCTURES

In order to facilitate the sorting of polygons and elements involved in shadow determination, as well as to perform visible surface determination when progressive rendering is done, the input polygons are used to generate a BSP tree representing these elements as in [7].

The reader may recall that a BSP (Binary Space Partitioning) tree can be used to represent a collection of polygons in a volume of space by recursively subdividing the volume along planes determined by the orientations of the polygons within the volume. Polygons may be chosen in any order to determine these partitioning planes, with the most desirable order being one which results in the fewest polygons being split along plane boundaries as the process proceeds. (Note that determining such an optimal order is known to be NP-complete [10].) The resulting data structure is a binary tree, in which each interior node represents a partitioning plane and its defining polygon, along with any other coplanar polygons that may exist in the input database, and the leaf nodes represent convex volumes of space determined by the partitioning. Henceforth we will ignore these leaf nodes and assume that all nodes represent partitioning planes/polygons. Figure 4(a) shows a two dimensional example, in which line segments represent polygons. Figure 4(b) shows the BSP tree for this scene.

As described in [7], BSP trees can be used to determine the visibility priority of a collection of polygons from any viewing position. This is achieved by using an *inorder* traversal of the tree. Each node is processed recursively by inserting the coordinates of the viewing position into the planar equation of the partitioning plane at that node. The sign of the result indicates whether the viewing position is in the "front" half-space determined by the plane, the "back" halfspace, or on the plane itself, where "front" and "back" are relative to the plane normal. If the viewing position is in one of the half-spaces, the subtree representing that halfspace is processed first. If the viewing position is on the plane, the subtrees can be processing in any order. Once the first subtree has been

Figure 5: Polygon Substructuring

processed, the polygons within the current node can be output and the other subtree processed to terminate the routine for that node. Thus the exact order of traversal is determined by the viewing position, and it is guaranteed that this order will output polygons in front to back order relative to the viewing position. Further details can be obtained in [7]. Figure 4(c) shows the output order using this algorithm on the viewing position and scene shown in Figure 4(a).

The BSP tree calculated in the initialization step, which we call a polygon BSP tree (pBSP tree) is used to sort surface elements in front to back order from the point of view of the centroid of each emitting element in turn. This ordering is used to speed up shadow calculations.

Some other operations are also enhanced through the use of the pBSP tree. In the processing of a source polygon, before all the other polygons are sorted in front to back order, backfacing polygons are removed from consideration and polygons in the back halfspace of the emitting surface's plane are also removed. The pBSP tree can also be used for visible surface determination for progressive rendering when the energy transfer computations at each step are complete using the technique of [7].

In processing a source element, all polygons that are in the front halfspace of this element and are not backfacing are sorted in front to back order using a pBSP tree traversal from the point of view of the centroid of the emitter. We call a node of the pBSP tree a *pnode*. It contains the equation of the plane passing through a set of coplanar input mesh polygons, the boundary representations of those polygons, and a representation of the surface elements into which each of these polygons has been subdivided. The collection of elements for each polygon in the pnode forms a BSP tree in two rather than three dimensions, which we call an element BSP tree (eBSP tree). This eBSP tree initially consists of a single *enode* representing the input polygon for that element tree. Surfaces in an eBSP tree are always found at the leaves, with internal enodes representing splitting lines. When an element at one of these leaves is split, its enode is replaced by a tree whose root represents the splitting line, with left and right children being the new leaves representing the two new elements formed by the split. This process is illustrated in Figure 5.

3.2. LIGHT SOURCE SUBDIVISION

For each iteration of the illumination process, we use the polygon with most untransmitted energy as the light source.

After the source polygon, *S*, is chosen, we use the pBSP tree to determine *R*, the nearest frontfacing receiver polygon. Then we calculate *A*, the solid angle subtended by *S* in the viewing hemisphere centered at *R*'s midpoint. If *A* is small enough, then *S* may be approximated as a point light source for shadowing computations. We have experimentally determined that .005 steradians is an effective size tolerance. The average color of the source polygon is used as the light color in illumination calculations.

If the light source exceeds our size criterion, we first check if it has already been subdivided by previous calculations. If so, the size check is recursively applied to elements successively deeper in the subdivision hierarchy until either we find small enough pieces or the substructure can be traversed no further. If bottom level subpolygons are too large, they are subdivided across their long axes until the size criterion is met.

3.3. SHADOW BOUNDARY COMPUTATIONS

The first step in processing a receiving polygon for any given emitting element is to determine whether any of the polygons closer to the emitter casts a shadow on the current receiving polygon. This is done by testing the receiving polygon against the merged shadow volume generated by the emitter centroid and closer polygons. This volume consists of a collection of semi-infinite pyramids, one pyramid emanating from each shadowing polygon. The merged structure is maintained as a second BSP tree, called a shadow volume BSP tree (sBSP tree), similar to the SVBSP trees of [2], whose nodes we refer to as *anodes*.

The sBSP tree is made up of the shadow volume planes cast by each of the previously-processed polygons. Internal snodes represent clipping planes, and leaf snodes represent regions which are classified as either totally lit or totally in shadow with respect to the current light source. Elements are filtered down this tree to determine which of their regions are lit or in shadow.

Our algorithm is similar to that of [2], but features important improvements in the shadow testing algorithm. We use the pBSP tree to generate a front-to-back ordering from the light source position. Polygons are tested for shadows in this order, which guarantees that no polygon will be processed before any that can shadow it. After shadow testing, the processed polygons are added to the sBSP tree.

For the illumination step, we start out with the light source position, the pBSP tree of input polygons, and an empty merged shadow volume. Each polygon is filtered down the sBSP tree. If the entire polygon is completely lit or shadowed,

the polygon does not need to be subdivided. If, however, the polygon crosses a shadow boundary, the polygon is split across this edge.

When a polygon is split, we must refine its eBSP. Each split adds a level to the sBSP tree. We maintain the elements in this structure for efficiency in future illumination passes. This differs from [2], who maintain a simple linked list of fragments.

After the first illumination pass, many of the original polygons may have been split multiple times by previous shadow planes. As a polygon is tested for shadows, we first check if the original polygon boundary is totally lit or in shadow. If so, we are done with this polygon. If not, we recursively descend the eBSP tree, each time checking if the current element is totally lit or shadowed. In this way we are potentially able to avoid testing all the leaf enodes individually. When we finally do have to process a leaf enode, we know that it definitely falls across a shadow edge and must be split.

Pseudocode is provided in Figure 6.

A straightforward implementation of the described algorithm produces a mesh which is indeed fine and coarse in all the right places. However, the continuous splitting of elements across shadow boundaries throughout all illumination calculation leads to excessive mesh refinement at shadow boundaries. In addition to the general undesirability of too fine a mesh, the memory costs can soar dramatically.

Since shadow boundaries become less important after the brighter light sources are processed, at some point we can terminate splitting across shadow boundaries to prevent large numbers of unnecessary small elements from being created. The polygon can still be split for form factor computation as usual, but its pieces may be merged back after illumination computations have been made. We have found that stopping the shadow clipping after half of the light has been dissipated is effective. Using this optimization, light leaks may occur, but if a proper clipping termination threshold is used, the energy levels of the light sources are low enough to make this effect negligible.

3.4. INTENSITY CALCULATIONS

Once shadow calculations are done for a particular source-receiver pair, the next step tests the quality of the resulting receiving mesh against our criterion of balanced energy distribution across mesh elements. This is of course done only for elements which have previously been determined to be completely visible to the light source. The form factor for the receiving element is first estimated. This is done by computing the point to point form factors between the centroid of the emitter and each of the receiver's vertices. These vertex form factors are averaged and multiplied by the receiver's area to provide a form factor approximation

```
(* Use sBSP tree for shadowing *)
Procedure SbspShadow(Src, NumRcv, Rcv)
Begin
  ShadTree := SbspInit0;
  For i := 1 To NumRcv Do Begin
    ShadowCalc(Src, ShadTree, Rcv[i]);
    ShadTree := SbspUpdate(Src, ShadTree,
                          Rcv[i])
  End;
  SbspDestroy(ShadTree)
End;

(* Traverse nested polygons *)
(* LeafShadowCalc() does single-level
   sBSP shadowing, as in [2]. *)
Procedure ShadowCalc(Src, Tree, Rev)
Begin
  DivdnSrcPlane(Src, Rev, Front, Back);
  If Front = NIL Then
    SetIllum(Rcv, NONE);
  Else If HasChildren(Rcv) Then Begin
    If (Back <> NIL) Then Begin
      ShadowCalc(Src, Tree, Rcv.Pos);
      ShadowCalc(Src, Tree, Rcv.Neg)
    End;
    Else Begin
      PolyCopy(Rcv, Rcv1);
      LeafShadowCalc(S, Tree, Rcv1);
      If (HasIllum(Rcv1, NONE)) Then
        SetIllum(Rcv, NONE)
      Else If (HasIllum(Rcv1, TOTAL))
        SetIllum(Rcv, TOTAL)
      Else Begin
        SetIllum(Rcv, PARTIAL);
        LeafShadowCalc(S, Tree,
                      Rev.Pos);
        LeafShadowCalc(S, Tree,
                      Rcv.Neg)
      End
    End
  End
Else Begin
  If (Back 0 NIL) Then Begin
    SetIllum(Rcv, PARTIAL);
    AddChildren(Rcv, Front, Back);
    ShadowCalc(Src, Tree, Front)
  End
  Else
    LeafShadowCalc(Src, Tree, Rcv);
End
End;
```

Figure 6: Shadow Testing

for the entire element.

Note that in the event that the receiver has a large aspect ratio, there can be significant differences in the form factors at its vertices. In such cases, the two longest edges of the receiver can be bisected to form a pair of elements of lower aspect ratio, and the vertex form factors for each resulting element calculated. Once the differences in vertex form factors fall within a specified tolerance, this subdivision is terminated and the averages calculated. This process can be done efficiently since all points on the original receiver are known to be illuminated by the source and therefore no visibility calculations are involved in determining form factors at the bisection points.

The sum of all element form factors, R , is computed. In theory, this sum should be 1, but as with other radiosity techniques, errors resulting from the approximate calculation of form factors can cause this to vary. Thus we scale form factors, both for elements and vertices, by $1/R$ to conserve energy balance. These normalized vertex form factors are used to calculate the vertex intensity contributions from the current light source. If these intensity contributions make the element intensity gradient exceed the intensity tolerance, the element must be subdivided as for large form-factor variations. The intensities at the newly-created vertices are computed and intensity variations determined for each new element. This process continues until the variation of each piece is within the specified bound.

Once the intensities of the vertices have been determined, an image can be displayed by Gouraud shading the receiving elements. If this rendering technique is used, however, care must be taken to ensure that the T-vertices created by our mesh generation algorithm do not cause shading anomalies. One way to handle this problem is to make a T-vertex a three-way vertex, with two collinear edges instead of a single edge orthogonal to the remaining edge. The element with the two collinear edges is then triangulated before shading is done.

4. RESULTS

We have implemented our algorithm in the C programming language. Mesh generation and illumination computation is performed on a SUN 4/260, which is rated at ten VAX 11/780 mips. Polygon scan-conversion and smooth shading are done separately on an HP 9000 series 300. All timing figures are given for the SUN machine. Rendering time on the HP is only a few seconds. Times were measured using the UNIX *time* facility.

Figures 8 through 15 show the progress of our algorithm on a domestic scene. The input data consists of 211 polygons. Timing data is given in Figure 7.

Notice how fine detail is prominent in bright areas and near shadow boundaries, but not anywhere else. The algorithm has successfully pinpointed the areas of high gradation in the scene. Note that the meshes are not triangulated at T-vertices as described above.

Iterations	1	2	5	9
Mesh Polygons	9711	17958	20718	21073
pBSP Build (s)	3.7	3.7	3.7	3.7
sBSP Build (s)	26.0	46.0	60.7	72.8
Shadow Testing (s)	339.8	806.7	1036.2	1300.5
Illumination (s)	24.1	57.8	71.7	94.0
Total Time (s)	393.6	914.2	1172.3	1471.0

Figure 7: Timing Statistics

Figure 8: Mesh for 1 Iteration

Figure 9: Mesh for 2 Iterations

Figure 10: Mesh for 5 Iterations

Figure 11: Mesh for 9 Iterations

Figure 14: Rendering for 5 Iterations

Figure 12: Rendering for 1 Iteration

Figure 13: Rendering for 2 Iterations

Figure 15: Rendering for 9 Iterations

5. CONCLUSIONS AND FURTHER WORK

We have described an algorithm for adaptive mesh generation for global diffuse illumination. It operates by subdividing input polygons at shadow umbra boundaries relative to emitting elements, and then further subdividing until approximately constant intensity for each illuminated element is obtained. The net result is a mesh whose density is proportional to its illumination. Using such a mesh in subsequent energy transfer calculations concentrates the work done in these calculations where it will distribute the most energy in a way that complements such techniques as progressive refinement. The adaptive mesh is itself computed by progressive refinement. Costs of shadow calculations are controlled through the use of a nested BSP tree data structure in which the nodes of the overall polygon BSP tree themselves contain 2-dimensional element BSP trees to sort polygons and order shadow computations.

The current technique is still rather crude. For instance, our algorithm divides each light source into a worst case collection of elements which will cast reasonably accurate penumbras on even the closest surfaces. Explicitly calculating penumbra and umbra boundaries may provide a more effective technique for handling these regions. Also, stopping criteria for shadow clipping are currently ad hoc and need to be subjected to a proper illumination error analysis.

Ultimately our goal is to use adaptive mesh generation as a means of accurately controlling illumination errors without unduly multiplying the number of elements required throughout the scene. Achieving this goal will require a great deal of further analysis and adaptation of the mesh subdivision criteria. Robust criteria for subdividing emitters remain to be determined. Illumination errors caused by point-to-point radiosity computation must be more carefully analyzed. Nevertheless, we believe that our technique represents a promising step towards an understanding of mesh generation constraints comparable to the understanding of energy distribution algorithms obtained over the past several years.

6. ACKNOWLEDGEMENTS

We would like to thank Chris Buckalew, Don Speray, K.R. Subramanian, and Kelvin Thompson for their helpful discussions and suggestions. Thanks are due also to MIPS Computer Corp., John Beck and Jeff Carruth for the loan of one of their workstations, and to the Applied Research Laboratory and the Center for High Performance Computing at the University of Texas at Austin for the use of their computing facilities. We would especially like to thank ALFA Engineering, Inc., particularly Walter S. Reed and Philip D. Heermann, for access to their graphics workstation and for rewarding discussions.

REFERENCES

- [1] Baum, Daniel R., Holly E. Rushmeier and 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.
- [2] Chin, Norman and Steven Feiner, Near Real-Time Shadow Generation Using BSP Trees, *Computer Graphics* (SIGGRAPH '89 Proceedings), Vol. 23, No. 3, July 1989, pp. 99-106.
- [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., Donald P. Greenberg, David S. Immel and Philip J. Brock, A Radiosity Solution for Complex Environments, *IEEE Computer Graphics and Applications* Vol. 6, No. 3, March 1986, pp. 26-35.
- [5] 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.
- [6] Cook, Robert L., Thomas Porter, Loren Carpenter, Distributed Ray Tracing, *Computer Graphics* (SIGGRAPH '85 Proceedings), Vol. 19, No. 3, July 1985, pp. 111-120.
- [7] Fuchs, Henry, Zvi M. Kedem and Bruce Naylor, On Visible Surface Generation by A Priori Tree Structures, *Computer Graphics* (SIGGRAPH '80 Proceedings), Vol. 14, No. 3, July 1980, pp. 124-133.
- [8] Goral, Cindy M., Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile, Modeling the Interaction of Light between Diffuse Surfaces, *Computer Graphics* (SIGGRAPH '84 Proceedings), Vol. 18, No. 3, July 1984, pp. 213-222.
- [9] Kajiya, James T., The Rendering Equation, *Computer Graphics* (SIGGRAPH '86 Proceedings), Vol. 20, No. 4, August 1986, pp. 143-150.
- [10] Naylor, Bruce, A Priori Based Techniques for Determining Visibility Priority for 3-D Scenes, PhD Dissertation, University of Texas at Dallas, May, 1981.
- [11] Nishita, 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.

- [12] Nishita, Tomoyuki, I. Okamura and Eihachiro Nakamae, Shading Models for Point and Linear Light Sources, *ACM Transactions on Graphics*, Vol. 4, No. 2, April 1985, pp. 124-146.
- [13] Siegel, Robert and John R. Howell, *Thermal Radiation Heat Transfer*, Hemisphere Publishing Corp., Washington DC, 1981.
- [14] Thibault, William and Bruce Naylor, Set Operations on Polyhedra Using Binary Space Partitioning Trees, *Computer Graphics* (SIGGRAPH 87 Proceedings), Vol. 21, No. 3, July 1987, pp. 153-162.
- [15] 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.
- [16] 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.
- [17] Whitted, Turner, An Improved Illumination Model for Shaded Display, *Communications of the ACM*, Vol. 23, No. 6, June 1980, pp. 343-349.