A dynamic discontinuity meshing algorithm

Chrysanthou, Yiorgos

A dynamic discontinuity meshing algorithm

Y. Chrysanthou

Department of Computer Science,
QMW University of London,
Mile End Road,
London E1 4NS, UK.

e-mail: yiorgos@dcs.qmw.ac.uk

Abstract

An object space algorithm for computing shadows in dynamic scenes illuminated by area light sources is presented. The method is a combination of the Shadow Tiling used for point sources and the discontinuity meshing (DM) method used in Radiosity. The shadow boundaries as well as other discontinuities in the illumination function, of each surface in the scene, are found and built into a mesh using BSP tree merging. The combination of the space subdivision provided by the tiling and the structured mesh building provided by the merging leads to a significantly faster DM algorithm, which in addition allows for incremental updates after a change in the scene geometry. Experimental results show that interactive frame rates can be achieved using this method on workstations which do not have specialized 3-D graphics hardware.

Key Words: Shadows, shadow volumes, area light sources, BSP trees, discontinuity meshing, dynamic modifications.

Category: Research.

1 Introduction

The presence of shadows in an image helps viewers to better understand the spatial relationships between objects, is vital for interactive applications such as Virtual Reality, and, in general increases the appearance of reality that a picture provides. Many shadow algorithms have been devised that adequately solve the problem, ranging from the very simple (point light sources and local illumination [8]) to the very detailed and realistic (Radiosity [13] and Ray-tracing [7]). Very little work, however has gone into providing algorithms suitable for interaction. The dynamic shadow algorithms currently available find only shadow umbras, for fake shadows [1] or point light sources [6].

Shadows from point sources provide a lot of information about the spatial relations of the objects in the scene. In the real world most of the light sources have a non-zero area. To add to the realism of the images the effect of such sources should be modeled. Shadows due to area sources have soft edges, they are no longer defined by a singular sharp

boundary (umbra), but also have partially lit areas (penumbra). The most accurate object space method for calculating the shadow boundaries and other additional discontinuities in the illumination function is the Discontinuity Meshing (DM), most often presented as a first step to radiosity.

In this paper we present a new DM-algorithm that, as well as being faster than previous methods, it also allows for fast incremental updates during interaction.

In the next Section we give a brief review of shadows from area light sources and Discontinuity Meshing. The description of the new method is divided in two: in Section 3 we describe how the mesh is initially constructed and in Section 4 we show how to update it when an object is transformed. We give the results and conclusion in the two final Sections.

2 Shadows from Area Light Sources

The boundaries between lit and penumbra and between penumbra and umbra areas are celled the *extremal boundaries* of the shadow. The first to compute the exact extremal boundaries were Nishita and Nakamae [22], by using shadow volumes and considering all pairs of objects in the scene. More efficient methods based on Shadow Volume BSP trees were latter presented by Campbell and Fussell [3] and Chin and Feiner [4].

As correctly noted by Campbell and Fussell [3] the illumination function has maxima, minima and discontinuities within the penumbra regions. They used sampling to locate them. These discontinuities occur along curves in the penumbra where the visible part of the source changes qualitatively [12]. These curves are located at the intersection of the *critical surfaces* EV and EEE. EV surfaces are planes defined by an edge and a vertex in the scene, while EEE surfaces are quadratic surfaces defined by three non-adjacent edges [12]. The most abrupt discontinuities (value discontinuities) lie along the edges where objects touch, these are denoted by D^0.

Discontinuity Meshing was developed throught the Radiosity method as a means of constructing a more accurate mesh that will include all these edges. The first study on DM was presented by Heckbert [17] for a 2-D domain by considering every possible interaction between the edges and vertices in the scene and was later extended to a 3-D environment [16]. Concurrently a different 3-D algorithm was proposed by Lishinski *et al* [18]. All of these methods accounted only for EV edges.

EEE surfaces were partly treated by Teller [27], in a related computation where the visible region of a source through a sequence of portals is calculated. Complete DM algorithms were later presented by Drettakis and Fiume [9] and Stewart and Ghali [25]. Most of the other researchers, including the author, have chosen to ignore EEE (and non-emitter EV) surfaces because the error produced by their exclusion is small compared to their cost.

All existing methods for DM share a common fundamental problem that makes them

unusable for interaction. They trace each discontinuity surface in the scene separately, so even though they find all critical edges they cannot find the areas covered in shadow. If they were to be used for interaction, these methods, would have no way of knowing which vertices or edges have a modified visibility with respect to the source, when a polygon is added or removed from the scene. An example of a case where existing algorithms would fail is shown in Figure 9.

In the method described in the next Section we deal with this by taking a step backwards and treating the discontinuity meshing problem as a shadow problem.

3 Constructing the Mesh

The basic structure of the method resembles a point source shadow algorithm [24]. First the polygons are ordered front-to-back as seen from the light source by means of an augmented BSP tree. In this order they are "projected" onto the sides of a hemi-cube placed around the scene. Polygons whose "projections" on the cube overlap have a possible shadow relation. Shadows are casted using these relations. Finally, the vertices of the mesh are illuminated. We will explain each module separately.

3.1 Ordering the Polygons from the Source

Ordering in respect to an area, using the BSP tree, is not as straight forward as ordering in respect to a point. This is because an area cannot necessarily be unambiguously classified against the root plane at each node. Previous researchers dealing with area light sources realized this problem but being unable to find a satisfactory solution, their methods resulted in unnecessary processing [2, 4, 11].

Take for example the simple scene of Figure 1(a). Traversing the tree of Figure 1(b) front-to-back from different points on A gives different orderings: a1 gives {2, 1, 3} while a2 gives {3, 1, 2}. But the different orderings do not imply that there is a cycle or that an order valid for all points on A cannot be found. In fact because we are dealing with oriented polygons and we are only considering a limited viewing area, commonly there will be an invariant ordering, for example {1, 3, 2} here.

As we can see in Figure 1(c) a different tree can be constructed that when traversed gives that ordering from any point on A. The reason why a tree like T_1 does not work is because different points on A lie in different subspaces of polygon 1 and hence produce different orderings on the children of 1. In T_2 this still holds but since both children of 1 are now empty their ordering is irrelevant.

This observation led us to the following simple solution. First we build a normal BSP tree (call it T') using the scene polygons that have the scource totally in front of them and then we add any polygons that cut the source with their plane (we call the latter *offending*). Polygons that have the source totaly behind are irrelevant to the shadow algorithm. If each of the offending polygons reaches a different cell of T' then the resulting tree is an

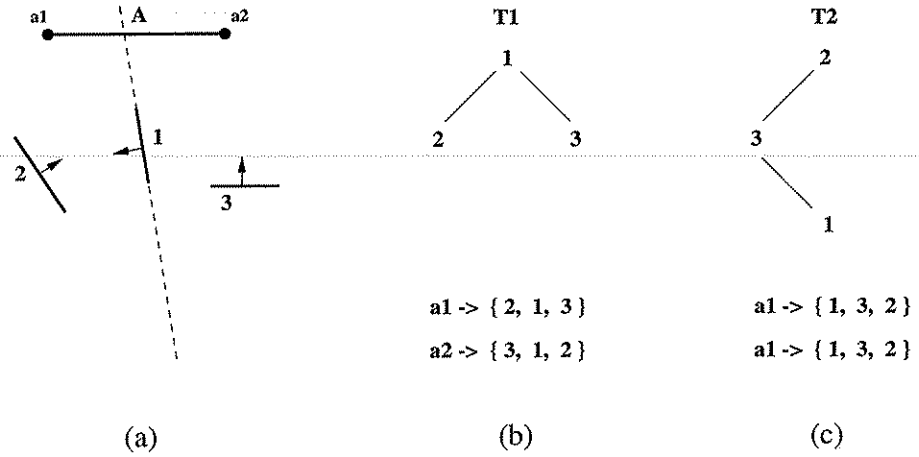


Figure 1: (a), (b) When the plane of an internal node cuts the area (A) different orderings are produced for different points on the area. (c) If the cutting node is placed at the leaves then the ordering is the same for every point on A

ordered list that can be walked left-to-right, if left is the front child in our BSP tree, to give the desired order. If more than one offending polygons reach the same cell of T' then we order them using the graph theoretical approach described in [23, 20].

Evaluation and a generalization of this method to large areas using an additional constrain are given in [28].

3.2 Determining Shadow Relations Between Polygons

A cube with its sides subdivided into a regular grid, large enought to enclose the scene including the volume where objects may possibly move, is constructed and placed around the scene.

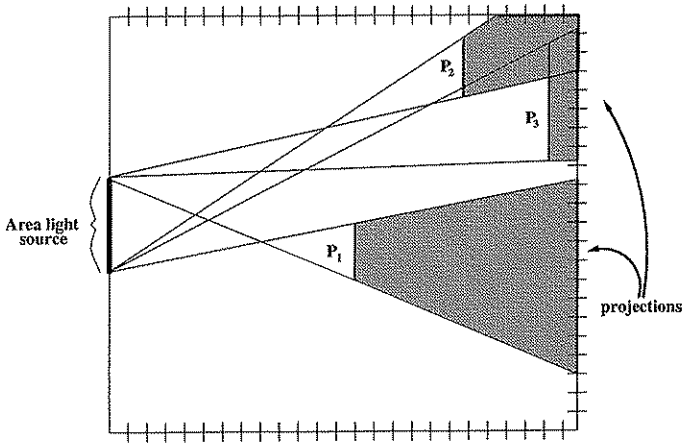


Figure 2: A tiling cube is placed around the scene, only polygons which overlapping penumbra on the cube have shadow relations

As each polygon P_i is processed in front-to-back order, the first time it is encountered all its critical surfaces are created. For convenience we group the critical surfaces of a polygon into three sets the penumbra (PSV), umbra (USV) and internal (ISV) shadow

volumes. The "projection" of P_i onto the cube is the intersection of the cube sides with and the PSV of P_i Figure 2. This intersection is found and is scan-converted into the grid stored at each side of the cube. During the scan-conversion the list of polygons already stored in the grid elements are added to a list. Then P_i only needs to be compared against the critical surfaces of these polygons.

An optimization proposed by Haines in [14] can be used for speeding up the scan-conversion of polygon projections in the tiling cube: we can avoid comparing all 5 sides of the hemi-cube against the penumbra planes by first projecting one of the penumbra vertices onto the cube to find which side it falls. During scan-conversion of the projection on this side, if any boundary edge is crossed then we continue with the cube-side over that edge.

3.3 Casting a Shadow Between two Polygons

Given two polygons (an occluder O and a receiver R), identified by the process above to have a shadow relation we proceed to find the discontinuities and the regions on the receiver, covered by the umbra or penumbra of the occluder.

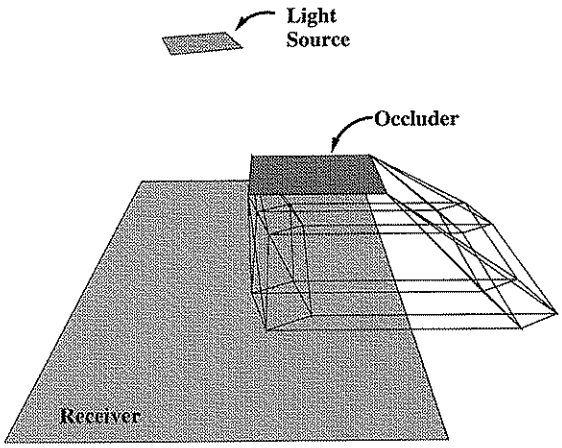


Figure 3: The source, the receiver and the occluder with the complete set of EV planes.

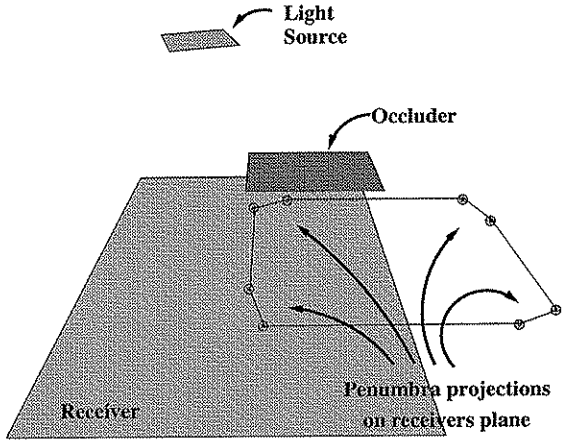


Figure 4: The penumbra vertices are cast on the receivers plane and checked for intersection with the receiver.

First we apply an additional verification test on the shadow relation: the penumbra vertices are projected on R's plane and the area they define is compared against R, Figure 4. Since all critical surfaces from an occluder are enclosed by the penumbra, if the receiver has no intersection with the penumbra then it cannot have an intersection with any of the other surfaces. The shadow casting terminates here if no intersection is found, and we proceed to the next (R, O) pair. If there is some intersection then the rest of the vertices (umbra and internal) are projected onto R's plane, Figure 5, and they are joined to make a DM-tree of discontinuities from O, Figure 6. We call this tree the *single DM-tree* of O on R (or simply *single-tree*). This single DM-tree is then merged into the total DM-tree of R, Figure 7.

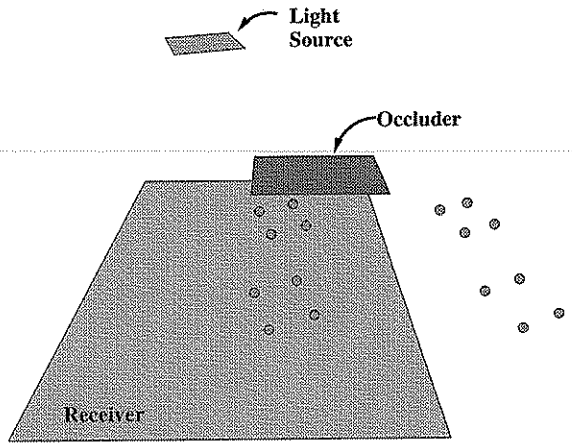


Figure 5: When an intersection is established the rest of the vertices are also projected.

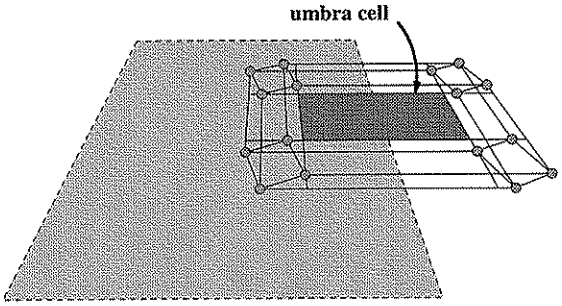


Figure 6: The single-tree is built using the adjacency information in the shadow planes.

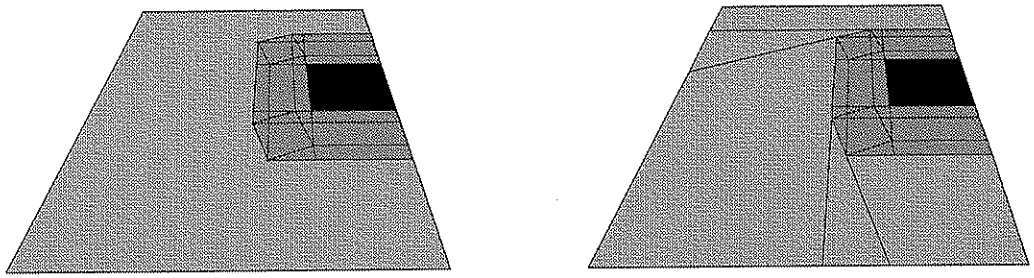


Figure 7: The single-tree is merged into the total-tree of the face, clipping anything outside and adding construction edges to any penumbra edge not spanning its subspace.

3.3.1 Constructing the Single DM-Tree

The construction of the single-tree proceeds in three steps. First, the penumbra subtree is constructed by traversing the penumbra vertices (Figure 4) and connecting them. We know that none of these edges intersect so they form a linear tree. The umbra subtree is then built using the umbra edges but some comparisons are needed here as there may be intersections between them. This subtree is attached at the back of the penumbra one. At this point the umbra cell is identified, if it exists, and it is marked.

Finally, the edges due to surfaces in the ISV are added one by one starting at the first umbra node since they are definitely behind all penumbra nodes. As each internal edge is filtered down the tree, the vertices at its end-points are matched against those of the node edges. If a common vertex is found then the inserted edge and the node edge are connected at that vertex. This ensures that each vertex connects to the correct four edges without depending on machine precision and with minimal computation.

Some notable special cases of this method are:

D^0 edges: If the receiver and occluder are touching then some of the umbra vertices will coincide with penumbra vertices. In such case these (D^0) vertices are marked as both umbra and penumbra and any edge defined by two such vertices is marked as

D^0 edge.

Undefined vertices: When the receiver cuts the occluder with its plane then not all of the $n_o \cdot n_s$ vertices will project correctly. Dummy vertices are used to replace those undefined which are clipped away later during the merging with the total DM-tree of the receiver.

3.3.2 Merging the Single DM-Tree into the DM-Tree of the Receiver

After constructing the single-tree we merge it with the total DM-tree of the receiver. We use the algorithm for BSP tree merging proposed by Naylor [21] with some modifications to allow for trees not spanning the entire subspace in which they reside. Nodes with such a property are the boundary edges of the receiver and also the penumbra nodes of the single-tree. As seen in Figure 7 the latter are only expanded after they reach a cell.

The merging algorithm is recursive and terminates only when one of the trees involved reduces to a cell. A *CellOpTree* operation is then called to apply the union operation on the tree and cell with a result depending on the value of the cell.

In [21] the cell can have only two values, IN or OUT, indicating the containment of the cell in a polyhedron. However, here because the trees we are merging are not defined over the whole of 2-D space but rather over the limited subspace enclosed by the boundary of the receiver, we have an additional value OUT^*. This value is assigned to those cells lying on the outside of polygons boundary edges to show that the cell is outside of the space of interest. The other two values are still used and they refer to the containment of a cell in shadow. In addition to these values each cell carries extra information showing the list of polygons limiting its view from the light (occluders).

When merging polyhedra, if a cell is in either or both then it is assigned an IN value. This is why a tree added to an IN cell is compressed. Here this reasoning is only valid for the umbra cells. For the penumbra we need to keep the subdivision because it matters if a cell is in one shadow or more.

Actually in our implementation we keep the subdivision even in the umbra regions, if the occluders of the single tree and the cell are not from the same object, since during interaction the occluder of the umbra cell may be removed. So the values a cell can take and the result of the *CellOpTree* operation are:

- OUT^*, only for cells lying on the outside of boundary edges
  $OUT^* + tree = OUT^*$.
- OUT, for un-occluded cells
  $OUT + tree = tree$.
- IN, for occluded cell, along with this is stored a list with the occluding polygons, umbra ones first.

IN + *tree* = *tree'*, where *tree'* has the same structure as *tree* but each of its cells has the list of polygons in the cell added to its own list.

3.4 Computing Illumination Intensities on the Vertices

The strength of our algorithm is most apparent in the illumination step. Other DM-algorithms need to compare the source against a great percentage of the scene polygons before identifying the visible parts of the source from each vertex, even if the vertex is un-occluded or in umbra. In our method, however, not only do we know for each vertex if it is lit, in umbra or in penumbra, but we also know exactly which occluders block each penumbra vertex before we begin to illuminate it so there is no searching and no redundant occluder/source comparisons.

As a result of using a Winged Edge Data Structure, each vertex v_i holds a pointer to one of the edges of which it forms the end-point. From this edge the set of mesh cells C sharing v_i can be found. Each of these cells holds an *occluder-list* (O_{C_j}) which is a list of the faces that block the light source from the cells view, either partly or fully. The occluders that block the source fully, are stored (and flagged) at the head of the occluder-list.

Using these occluder lists we determine the visibility of the source for v_i. We have three cases to consider:

1. Any of the O_{C_i} are empty: The vertex is illuminated as un-obstructed. Vertices v_a, v_b and v_c in Figure 8. To avoid light leaks at D^0 vertices (v_c), umbra cells are always displayed with ambient light regardless of the vertex colour value.

2. One of the O_{C_j}s contains an umbra element: The vertex is given an ambient colour value (vertex v_d in Figure 8). In the rare case where a D^0 vertex is covered by the penumbra caused by different face then the vertex is treated as penumbra. These cases can be easily identified by the elements in the occluder lists.

3. All the O_{C_j}s are non-empty and contain no umbra elements: The occluder sets O_{C_j} of all the cells in C are put together using an intersection operation and the active subset $O_{covering}$ of the occluders that cover the vertex, is found. The polygons in set $O_{covering}$ are then used to determine the visible parts of the source, from the vertex. Examples of these cases are vertices v_e and v_f.

It is important that the above tests are performed in the given order otherwise shadow leaks may occur (eg for vertex v_c).

4 Dynamic Modifications

As the results presented in Section 5 indicate, the algorithm constructs the discontinuity meshing with considerable speed. However, that was not the main purpose of this research. The aim was not just to build another, faster, DM-algorithm but to build one that can

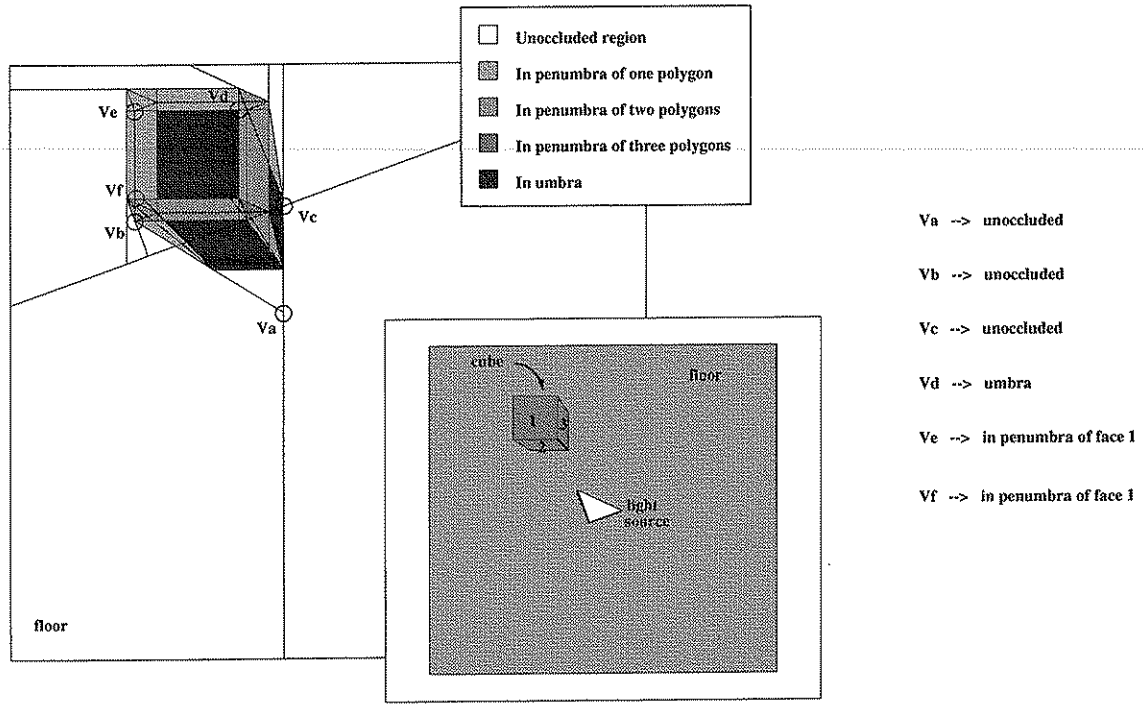


Figure 8: Possible classifications of a vertex during illumination.

take advantage of the spatio-temporal coherence in interactive applications and allow for the necessary modifications to be performed in a fraction of the normal construction time.

Incremental modifications are made possible due to a combination of certain aspects of the algorithm:

1. The space subdivision scheme significantly localizes the operations performed to only a small superset of the affected polygons. Drettakis [9] also uses a (voxel based) space subdivision scheme but as his algorithm traces each discontinuity surface independently, it fails to identify all polygons concerned during scan-conversion (small polygons fully in umbra or penumbra are only found on a separate step).

2. The use of BSP tree merging for adding the discontinuities from an occluder to a receiver polygon. This induces an explicit classification of the cells which provides a means for identifying the concerned vertices during interaction. For example in Figure 9, as the second object moves in the discontinuities due to this are found. A traditional method can find the newly created vertices and pass them for illumination, but not the covered vertices. Our method identifies these as they fall in an IN cell of the added single DM-tree.

Without loss of generality we will assume that any change in the scene data can be modeled by two operations: deletion and/or addition of objects. After each transformation we illuminate the relevant vertices.

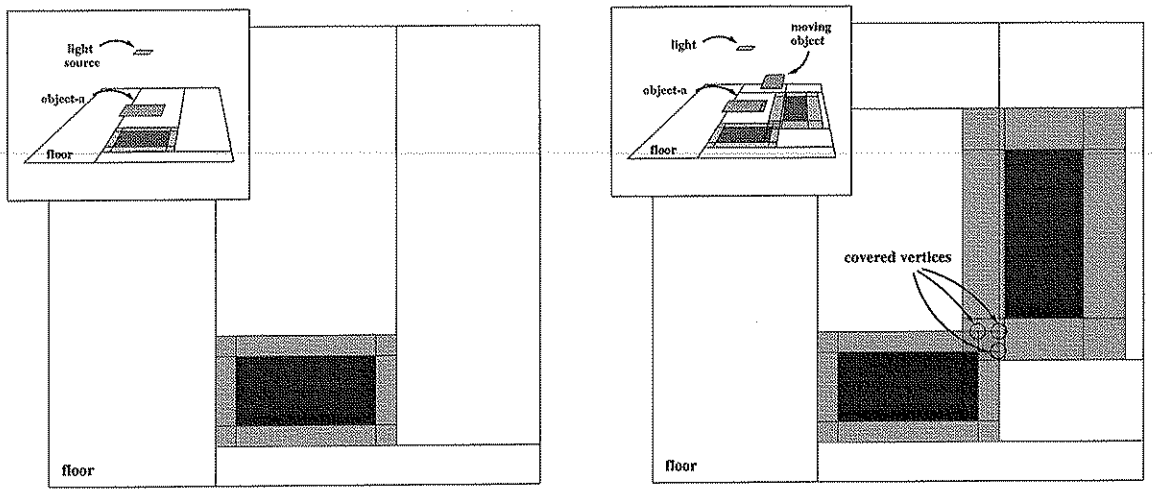


Figure 9: Merging allows for easy identification of the vertices with changed intensity when a polygon is added or deleted

4.1 Removing an Object

To remove the polygons of on object we first delete their entries from the tiling cube either by scan-converting them again (slow) or by using lists of the grid elements they go throught stored at the initial scan-conversion (more space consuming). Then we remove the polygons from the BSP tree using the method described in [5].

Each polygon holds a list of references to the receiver polygons upon which it has cast a shadow during the construction of the mesh. When removing an object polygon, its receivers are added to a list called *invalidDMT-list*. The polygons added to this list contain information in their DM-trees generated by the moving object which must be removed. So after removing all object polygons the DM-tree of each polygon in the list is traversed and scanned for two things:

1. Subtrees marked as completely covered by a polygon of the removed object. The cells of such subtrees are visited and any reference to the object in question is deleted (every vertex of such a cell is added to the illumination list).

2. Nodes holding discontinuity edges due to polygons in the removed object. These edges and their nodes are removed using the method described below. As the subdivision defined by these discontinuities is removed, any references to the object in the remaining cells must also be removed.

4.1.1 Deleting Edges from the DM-tree

When removing an edge from the DM-tree we are faced with the classical problem of removing nodes from a BSP tree. As a DM-tree node subdivides the polygon and the discontinuities further, removing it will result in two unconnected subtrees that will have to put together. Of source if one or both of the subtrees is empty then this is trivial, but in general both subtrees may be non-empty. The solution we suggest here for putting the

two subtrees together, is an optimization of BSP merging that takes advantage of the fact that they are defined in separate subspaces.

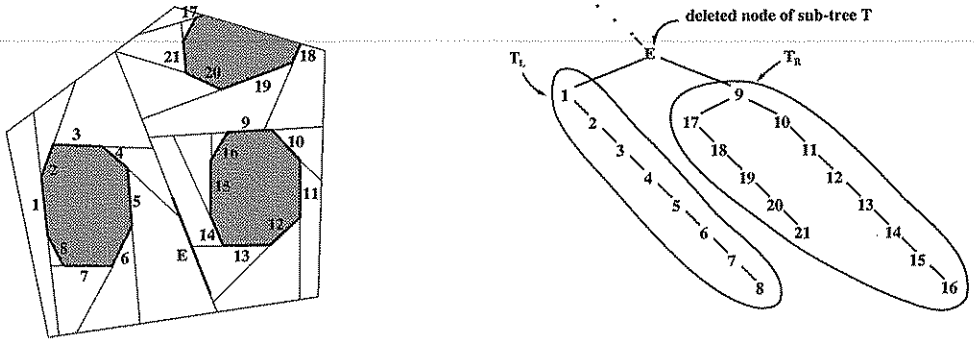


Figure 10: A 2-D scene and its tree representation

We will explain the method using the example in Figure 10. To delete the marked node of edge E that has two non-empty subtrees (T_L and T_R) the following three steps are performed:

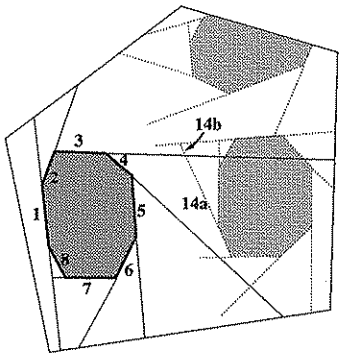


Figure 11: T_L is expanded to the whole subspace

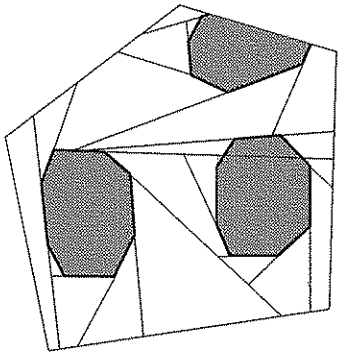


Figure 12: T_R is inserted into T_L to form one tree

Step 1 The edge at the marked node is removed from the WEDS. For this we need to traverse E from end to end. The edges at its endpoints are joined together (these are boundary edges of the cell defined at the parent node) while anything else touching the edge is marked as *dangling*, and may need to be expanded later. The number of *dangling edges* on each side of E is counted. In our example the dangling edges would be {3, 4} on the left and {9, 12, 13, 19} on the right.

Step 2 One of the subtrees is chosen to form the basis for the merging. The choice is based on the expected cost of inserting each subtree. An adequate estimation of the cost is: $E[cost_a] = D_a * size_b$, where $a, b \in$ {front-subtree, back-subtree} and $a \neq b$, D_a = number of dangling edges in a and $size_b$ = number of unmarked nodes in b, (for the DM-tree we can use as size the number of unmarked penumbra edges).

In Figure 10 $E[cost_{T_L}] < E[cost_{T_R}]$ so T_L is selected as basis for the merging. T_L is then traversed from top to bottom and the dangling edges are extended to span the whole of the cell defined at the removed node. This creates a convex partitioning of the cell

(Figure 11). To extend the dangling edges, the boundary of the cell defined at the parent node of E is traversed by always following edges from nodes that are ancestors of E. This is important since the edges of T_R are still in the WEDS but at the moment they should be ignored (Figure 11).

Step 3 T_R is inserted into T_L to form a unified tree, as in Figure 12. The important thing here is that only the nodes that were expanded in step 2 ($\{3, 4\}$) may possibly split T_R, as they are the only ones that intersect T_R's subspace. For the rest of the nodes in T_L that will be encountered ($\{1, 2, 5\}$), classifying one point on T_R will suffice. When the merging is finished the dangling edges of T_R must be extended to span the whole of their subspace. Meanwhile, at partitioning, the subtrees created are condensed to avoid unnecessary fragmentation of homogeneous regions. An example of where the condensation can take place is edge 14 in Figure 11. The top part, 14b, does not contain any part of a discontinuity and so it will be removed.

This method is particularly fast if one of the subtrees has only a few dangling edges. An example of an extreme case can be seen in Figure 13. Here our algorithm can detect the left side of E having no intersection (no dangling edges recorded when deleting E). Hence T_R is inserted in T_L as a point.

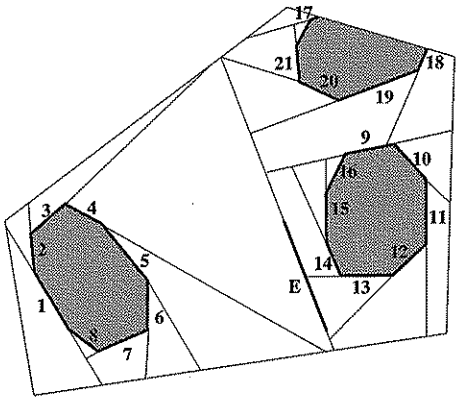


Figure 13: When E is removed, T_R can be inserted in T_L as a point because none of edges of T_L touch E

Also, if an object is transformed for more than one frame then the merging is only relevant for the first. In subsequent frames the nodes due to this object will at or near the leaves.

A more detail description and analysis of the performance of this method is given in [28].

4.2 Adding an Object

Adding an object to the scene requires similar steps to the initial construction of the mesh but only involves the polygons of the added object. First the polygons are added to the scene BSP which is the traversed to get the new front-to-back order. The new polygons are added to the tiling cube and the other faces sharing tiles with them are found. Shadows

are cast between the later and the new polygons.

4.3 Illumination of Vertices

After the deletion and/or addition of objects, new vertices will be created and some of the existing ones will have changed visibility, due to objects covering or uncovering them. However most of the vertices will remain unaffected and it would be extremely wasteful to recalculate the illumination for all of them. Instead a list is maintained during the deletion or addition of objects, which holds the relevant vertices. The vertices added to this list include the following:

During deletion of an object:

1. Existing vertices on cells that have one or more of their occluders removed from their occluder-list.

2. New vertices created during deletion of DM-nodes, either by extending dangling edges or by partitioning during merging of subtrees after deletion of a node.
   In fact it is not essential to recalculate the illumination value of these vertices from scratch, since the shadow information remains the same. Their value could be determined by interpolation from the end-points of the edge they partition, but we recalculate them for greater accuracy.

During addition of an object:

1. Existing vertices covered by added polygons.

2. All vertices on the mesh of added polygons.

3. Any other new vertex created by the discontinuities caused by the added polygons on the existing.

The illumination of the vertices is done in the same way as described in Section 3.4.

4.4 Optimization

One attribute of dynamic environments is that the attention of the user is distracted by the movement so a lot more imperfections can go unnoticed. In cases where the performance of the algorithm is not sufficient such as when the dynamic objects are large or moving over complex parts of the scene, a speed up can be obtained by using only extremal discontinuities for the dynamic objects (umbra and penumbra). We can return back to the full algorithm on release of the object.

5 Results

The main purpose of the method describe in this paper is for calculating the mesh in dynamic scenes. To evaluate the performance of the incremental updates we compute the

time taken for updating the DM after an object transformation has occured, and compare it against the time it takes to rebuild the whole DM from scratch. However, for this argument to be valid we have to show that the time for constructing the mesh is at least competitive with other DM methods.

5.1 Statistics for Initial Construction of the DM

The algorithm is written in C and implemented on a SUN SparcStation 20, 75MHz, Model 71 with 160M of RAM. Four different scenes were used in the experiments. The first (*15 cubes*) consists of 15 randomly placed cubes (92 polygons), the second (*officeA*) of a desk, a bookcase, a computer and a large polyhedral cursor (114 polygons) and the third (*officeB*) of two desks, one of them raised above the floor, and a bookcase (128 polygons). For the last scene we used three desks a bookcase and six randomly placed cubes (184 polygons).

	15 cubes		officeA		officeB		officecubes		15 cubes*	
	sec	%	sec	%	sec	%	sec	%	sec	%
total	1.70	100	1.23	100	1.77	100	2.24	100	0.77	100
build mesh	0.57	33	0.62	50	0.60	34	1.07	48	0.41	53
make SVs	0.02		0.03		0.03		0.04		0.02	
add to TC	0.06		0.02		0.05		0.12		0.04	
single-DM	0.16		0.22		0.17		0.29		0.14	
merge DMs	0.32		0.33		0.34		0.60		0.20	
misc	0.02		0.02		0.01		0.02		0.01	
illuminate	1.10	65	0.58	47	1.15	64	1.14	51	0.35	45
misc	0.03	2	0.03	3	0.02	2	0.03	1	0.01	2

Table 1: Analytical times for the construction of the mesh

The timings for these scenes are shown in Table 1. For each scene under the column labeled *sec* we show the absolute time, and under % the percentage of time for each routine. The first row shows the total time for each scene. Next we have the time for constructing the mesh, which is analyzed further in the following five rows. *make SVs* is the time to construct the shadow volumes (create the EV critical surfaces) which is not significant. *add to TC* is the time taken by the shadow tiling, to find the shadow relations and test for potential obstruction. This row provides evidence of the efficiency of our subdivision system. It helps to identify and process almost only the related pairs of polygons, and yet it takes an amount of time never exceeding the 4% of the total computation. The next two rows show the times for building the single DM-trees (*single-DM*) and for merging them to the total DM-tree of the receiver (*merge DMs*). These are both significant values taking up to almost a half of the total processing in certain scenes (e.g. officeA). *misc* refers to various secondary routines of the mesh construction, including the building of the BSP tree.

The row labeled *illuminate* shows the illumination computation. Following the discussion of Section 3.4 one might have expected this to be less expensive than what we

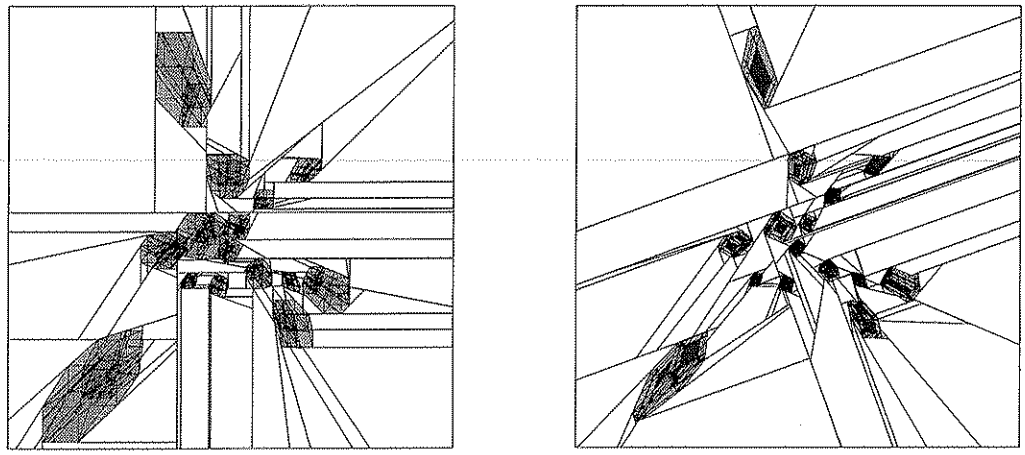
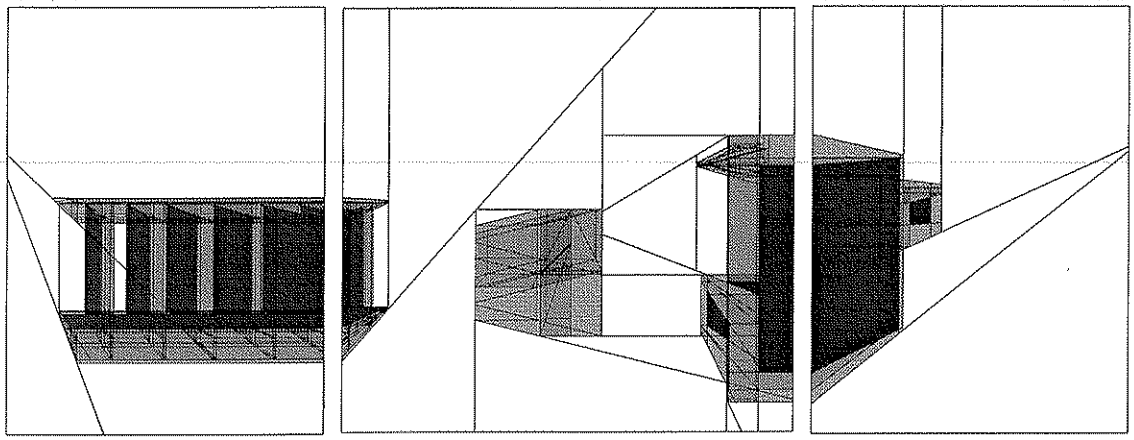


Figure 14: The mesh of *15 cubes* scene from (a) a large light source and (b) a source 5 times smaller

recorded here. In fact efficiency of the illumination step was apparent in the results by the fact that for all scenes, the average number of source/occluder comparissons per vertex was between 1.3 and 2.1. One of the reasons that the percentage of the illumination time is so hight is the matching efficiency of the rest of the operations, another is the size of the source. In all the scenes used here the light source is very large, this can verified by the width of the penumbras in the meshes shown in Figure 15 and Figure 17.

To give an example of how source size influences the performance, we run the *15 cubes* scene with a source 5 times smaller than the original, we called this *15 cubes**. The resulting mesh is less complex (Figure 14), with fewer vertices and in particular much fewer penumbra vertices, the illumination time drops vertically from 1.10s to 0.35s (with the average number of source/occluder comparissons per vertex dropping to 1.0). The construction time also drops since there are fewer intersecting edges in the mesh, but it does not decrease as much since the number of shadow relations remain almost unchanged. This dramatic difference in time indicates that the algorithm is output sensitive, meaning that the amount of computation depends more on the resulting mesh than in the number and geometry of input polygons.

One of the problems reported by other researchers [26] is that the use of DM-tree creates badly shaped cells with excessive subdivision. This is mainly due to the construction edges added to the discontinuities for forming the binary subdivision. One of the benefits of our methods is that without any user intervention this problem is very limited. In Figure 15 we have the mesh resulting from *officeA* on the left wall, the floor and the right wall, in that order. Here we can see almost no extra subdivision than the necessary. Of course this scene is well suited for our example, since the objects are rectangular with sides parallel to a rectangular source (apart from the cursor), however this pattern is present in all our experiments. In Figure 17(a) the source is rotated so that it is not parallel to anything and in Figure 17(b), which shows the mesh on the floor from *officecubes*, some objects are randomly placed. In both of these again the subdivision is small.



(a) (b) (c)

Figure 15: The mesh of (a) the left wall, (b) the floor and (c) the right wall for *officeA*

Another common problem of the existing DM-algorithms is the time complexity. In general this is more than linear. The only other work that reports close to linear growth is that of Drettakis [9] but even there the slope is steep. To give a rough idea of the growth rate of our method, we run a set of experiments using the cube scenes. We computed the construction/illumination times for scenes consisting of one to fifteen cubes, in steps of one. The results are shown in the graph of Figure 16, not only do we have linear growth but also the marginal cost of adding each extra object is almost the same throughout the range of the data.

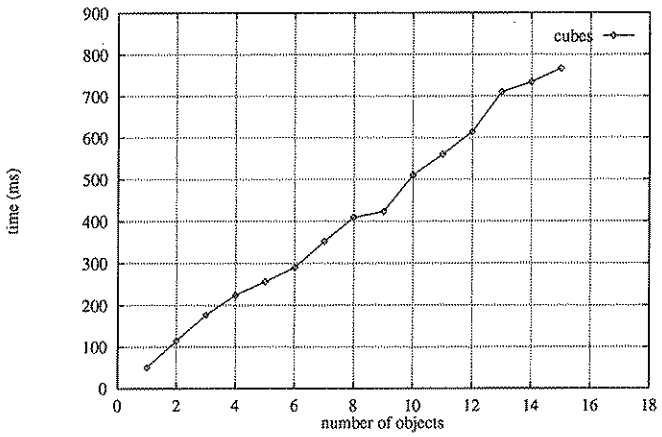
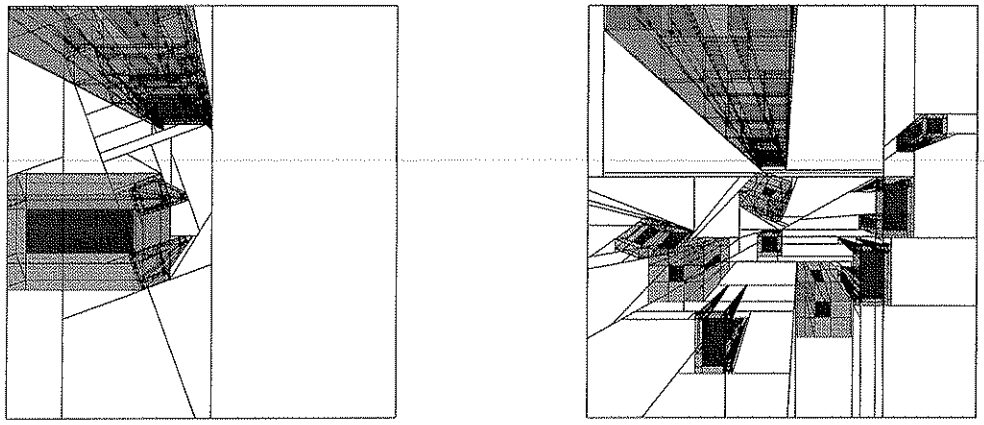


Figure 16: Time against number of objects for the cube scenes

One of the reasons we have such a reduced growth in this particular experiment is that the cubes are randomly placed without much overlap, as seen from the light source. This is optimal for the space subdivision used.

Larger scale experiments are required before we can have any conclusive evidence on the performance of the method. However, by comparing the present results with the results reported by other algorithms [18, 9, 10, 25], and especially the rate of growth, we



(a) (b)

Figure 17: The mesh on the floor from (a) two objects and a rotated source and (b) the *officecubes* scene

speculate that this method could be up to an order of magnitude faster.

5.2 Evaluation of the Incremental Modifications

Having established that our method is at least as good as the existing DM methods, we can now continue with the evaluation of dynamic scenes.

Selected objects from each scene were moved, the results are given in Table 2. For each scene we show the objects that moved followed by the translation times (under *transformation*). The transformation is broken into its two components the deletion of the object (*delete*) which includes removal from the tiling and from the DM-trees of the objects receivers, and the addition (*add*) of the object back to the scene. In the next two columns we give the times for rebuilding the scene mesh without the moved object (*rebuild without*) and the total time to rebuild the whole mesh, including the object (*rebuild total*). The values of the latter are taken from Table 1. Finally under *del as %rebuild* we give the percentage of time used for removing the object against rebuilding the mesh with out it.

As already stated the addition of a object in the mesh is performed using the same method as for the initial creation of the mesh. So the time taken to add the object (*add*) should be similar to the difference of the columns *rebuild total* and *rebuild without*. This is approximately the case. The important row in this table is the last one.

The values given under *transformation* are the average over ten small steps of continuous modification. Depending on the positions of the discontinuities in the mesh, the first deletion of certain objects may take longer than the rest, see Section 4.

6 Summary

In this paper a fast discontinuity meshing algorithm has been presented. A spatial subdivision based on the tiling cube, along with the ordering produced by an augmented BSP tree, were used for identifying potential shadow relations between model polygons.

scene	object moved	tranformation delete (s)	add (s)	rebuild without (s)	rebuild total (s)	del as %rebuild
15 cubes	cube 3	0.02	0.10	1.62	1.70	1
	cube 5	0.03	0.11	1.58	1.70	2
officeA	cursor	0.01	0.09	1.13	1.23	1
	bookcase	0.02	1.02	0.45	1.23	4
officeB	desk1	0.02	0.19	1.52	1.77	1
	bookcase	0.02	1.20	0.59	1.77	3
officecubes	cube 1	0.02	0.11	2.12	2.24	1
	cube 3	0.02	0.11	2.15	2.24	1

Table 2: Timings for mesh computation after transforming objects in the scene

While traditional DM algorithms trace each discontinuity surface separately through the model, our algorithm traces the whole set of surfaces (shadow) from an occluder together. The intersections of this set of surfaces with the plane of each receiver are found and they are connected together to form a DM-tree which is merged into the DM-tree of the receiver. This process has several advantages over previous methods, such as: reduced time complexity, increased accuracy and explicit classification of each resulting mesh cell in respect to its occluders leading to faster illumination calculations.

Due to the structured creation of the DM, incremental updates are made possible. The shadow information for moving objects can be computed using only a fraction of the computation required to compute the whole shadow information.

An issue that remains un-addressed, is that of further subdivision. In our implementation the triangulation method used for interaction is very straight forward. For every cell that changes, even in the slightest, its whole triangulation is recalculated. A better approach is required.

The DM-method presented, currently only computes direct illumination. An interesting direction for future work would be to extend this method to a full Radiosity solution ([15, 19]).

References

[1] J. F. Blinn. Jim blinn's corner: Me and my (fake) shadow. *IEEE Computer Graphics & Applications*, 8(1):82–86, January 1988.

[2] A. T. Campbell. *Modelling Global Diffuse Illumination for Image Synthesis.* PhD thesis, Department of Computer Science, University of Texas at Austin, December 1991.

[3] A. T. Campbell, III and D. S. Fussell. An analytic approach to illumination with area light sources. Technical Report R-91-25, Dept. of Computer Sciences, Univ. of Texas at Austin, August 1991.

[4] N. Chin and S. Feiner. Fast object-precision shadow generation for area light sources using BSP trees. In *ACM Computer Graphics (Symp. on Interactive 3D Graphics)*, pages 21–30, 1992.

[5] Y. Chrysanthou and M. Slater. Dynamic changes to scenes represented as BSP trees. *Computer graphics Forum, (Eurographics 92)*, 11(3):321–332, 1992.

[6] Y. Chrysanthou and M. Slater. Shadow Volume BSP trees for fast computation of shadows in dynamic scenes. In *Proceedings of the ACM Symposium of Interactive 3D Graphics*, pages 45–50, Monterrey, California, March 1995.

[7] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In Hank Christiansen, editor, *ACM Computer Graphics*, volume 18, pages 137–145, July 1984.

[8] F. Crow. Shadow algorithms for computer graphics. *ACM Computer Graphics*, 11(2):242–247, 1977.

[9] G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using backprojection. In Andrew Glassner, editor, *ACM Computer Graphics*, pages 223–230, July 1994.

[10] N. Gatenby. *Incorporating Hierarchical Radiodity into Discontinuity Meshing Radiosity.* Ph.D. thesis, University of Manchester, Manchester, UK, 1995.

[11] N. Gatenby and W. T. Hewitt. Optimizing discontinuity meshing radiosity. In *Fifth Eurographics Workshop on Rendering*, pages 249–258, Darmstadt, Germany, June 1994.

[12] Z. Gigus and J. Malik. Computing the aspect graph for the line drawings of polyhedral objects. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(2):113–133, February 1990.

[13] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. Modeling the interaction of light between diffuse surfaces. In Hank Christiansen, editor, *ACM Computer Graphics*, volume 18, pages 213–222, July 1984.

[14] E. A. Haines and D. P. Greenberg. The light buffer: a shadow testing accelerator. *IEEE Computer Graphics & Applications*, 6(9):6–16, 1986.

[15] P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. In Thomas W. Sederberg, editor, *ACM Computer Graphics*, volume 25, pages 197–206, July 1991.

[16] P. Heckbert. Discontinuity meshing for radiosity. *Third Eurographics Workshop on Rendering*, pages 203–226, May 1992.

[17] P. Heckbert. Radiosity in flatland. *Computer graphics Forum, (Eurographics 92)*, 11(3):181–192, 1992.

[18] D. Lischinski, F. Tampieri, and D. P. Greenberg. Discontinuity meshingfor accurate radiosity. *IEEE Computer Graphics & Applications*, 12(6):25–39, November 1992.

[19] D. Lischinski, F. Tampieri, and D. P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. In James T. Kajiya, editor, *ACM Computer Graphics*, volume 27, pages 199–208, August 1993.

[20] B. F. Naylor. *A Priori Based Techniques for Determining Visibility Priority for 3-D Scenes.* PhD thesis, University of Texas at Dallas, May 1981.

[21] B. F. Naylor, J. Amandatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. *ACM Computer Graphics*, 24(4):115–124, 1990.

[22] T. Nishita and E. Nakamae. Half-tone representation of 3-D objects illuminated by area sources or polyhedron sources. *Proc. COMPSAC 83: The IEEE Computer Society's Seventh Internat. Computer Software and Applications Conf.*, pages 237–242, November 1983.

[23] R. Schumacker, B. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. Technical Report AFHRL-TR-69-14, NTIS AD700375, U.S. Air Force Human Resources Lab., Air Force Systems Command, Brooks AFB, TX,, September 1969.

[24] M. Slater. A comparison of three shadow volume algorithms. *The Visual Computer*, 9(1):25–38, October 1992.

[25] A. J. Stewart and S. Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In Andrew Glassner, editor, *ACM Computer Graphics*, pages 231–238. ACM SIGGRAPH, July 1994.

[26] F. Tampieri. *Discontinuity Meshing for Radiosity Image Synthesis.* Ph.D. thesis, Cornell University, Ithaca, NY, 1993.

[27] S. J. Teller. Computing the antipenumbra of an area light source. In Edwin E. Catmull, editor, *ACM Computer Graphics*, volume 26, pages 139–148, July 1992.

[28] xxxxxxxxx. *Shadow Computation for 3D Interaction and Animation.* PhD thesis, xxxxxxxxxx, December 1995.