Incremental Updates to Scenes Illuminated by Area Light Sources

Y. Chrysanthou and M. Slater

Department of Computer Science,
UCL University of London,
Gower Street, London WC1E 6BT, UK.

Abstract: An object space algorithm for computing shadows in dynamic scenes illuminated by area light sources is presented. A mesh with the shadow boundaries as well as other discontinuities in the illumination function, is built in a pre-processing stage and updated on-line after any interaction resulting in a change in the scene geometry. The mesh on each polygon is a 2D BSP tree stored in a winged edge data structure. To accelerate the mesh construction a number of new ideas are employed: sorting of the polygons in respect to the area source, the shadow overlap cube, BSP tree merging of the shadows. In addition a method for dynamically changing the BSP representation of the mesh and quickly identifying the vertices requiring intensity computations was developed. Preliminary experimental results indicate the strength and the potential of this method.

1 Introduction

Shadows greatly enhance the quality of computer generated images. They help viewers to better understand the spatial relationships between objects for example for tasks such as object placement. Even though this is particularly important for interactive applications, shadow algorithms are considered too expensive for real-time computation and are usually omitted. At best some fake [2] or point light source [7] shadows are rendered. However, in the real world most of the light sources have a non-zero area, shadows due to such area sources have soft edges, they are no longer defined by a singular sharp boundary (umbra), but also have partially lit areas (penumbra).

In this paper we present an algorithm for the computation of shadows from area light sources, that can be used in interactive applications. The new method can also be seen as a discontinuity meshing algorithm since, along with the shadow boundaries, it can also compute other (EV) discontinuities of the illumination function.

The boundaries between lit and penumbra and between penumbra and umbra areas are called the *extremal boundaries* of the shadow. The first to explicitly take in account extremal boundaries were Nishita and Nakamae [19], by using shadow volumes and considering all pairs of objects in the scene. More efficient methods based on Shadow Volume BSP trees were later presented by Campbell and Fussell [3] and Chin and Feiner [4].

Campbell and Fussell [3] noted that the illumination function has maxima, minima and discontinuities within the penumbra regions and they used sampling to

locate these. The discontinuities occur along curves in the penumbra where the visible part of the source changes qualitatively [11]. These curves are located at the intersection of the *critical surfaces* EV and EEE with the surfaces of objects in the scene. 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 [11]. The most abrupt discontinuities (value discontinuities) lie along the edges where objects touch, these are denoted by D^0.

Discontinuity Meshing was developed through 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 [14] 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 [13]. Concurrently a different 3-D algorithm was proposed by Lishinski *et al* [15] which was subsequently combined with Hierarchical Radiosity [16]. All of these methods accounted only for EV edges.

EEE surfaces were partly treated by Teller [22], 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 [8] and Stewart and Ghali [21]. Similar information can also be obtained from the *3D visibility complex* of Durand *et al* [9]. However, EEE (and non-emitter EV) surfaces are not usually included in radiosity or direct illumination solutions because their contribution is small compared to their cost.

All of the previous methods were aimed at static scenes and cannot easily be adapted to deal with interaction. The method presented here is constructed from the outset with this problem in mind. Instead of tracing each critical surface independently, the set of surfaces defined by each occluder are traced together. This makes it easier to locate the affected polygons when an object is moved (using the shadow overlap cube). Also discontinuities are added to the meshes using BSP tree merging. This allows for fast identification of the vertices requiring intensity calculations after an interaction, it also gives a fast way of finding the exact occluders of each vertex.

The method for the initial construction and illumination of the mesh is given in Section 2. Section 3 describes how to update this information when there is a change in the scene geometry. The results and conclusions are given in the two final sections.

2 Construction and Illumination of the Mesh

The method produces a DM-tree [15] on each surface, where a DM-tree is a 2-D BSP tree supported by a winged edge data structure (WEDS)[1]. It proceeds in the following steps. 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. The shadows are cast by finding the discontinuities due to an occluder, on the receivers plane and merging them with the rest of its mesh. Finally, the vertices of the mesh are illuminated. In the rest of this chapter each module is explained separately.

2.1 Ordering the Polygons from the Source

The ordering is done using a hybrid of a BSP tree and graph theoretic sorting. Ordering in respect to an area, using the standard BSP tree, is not always possible.

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 [3, 4, 10].

First we build a normal BSP tree (T') using the scene polygons that have the source 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 totally behind are irrelevant to the shadow algorithm, since we assume all polygons to be single sided. If each of the offending polygons reaches a different cell of T' then the resulting tree is an ordered list that can be walked left-to-right, assuming left is the front or 'outside' child in our BSP tree, to give the desired order. If more than one offending polygon reaches the same cell of T' then we order these using the graph theoretical approach described in [20, 17].

Evaluation and a generalisation of this method to large areas using an additional constraint are given in [5]. Note that, this tree is only relevant for the sorting and any polygon splitting does not propagate to the shadow determination phase.

2.2 The Shadow Overlap Cube

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

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) and umbra (USV) shadow volumes and internal discontinuities (ISD). The "projection" of P_i onto the cube is the intersection of the cube sides with the PSV of P_i. 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 associated with P_i. Then P_i only needs to be compared against the critical surfaces of these polygons.

Alternatively instead of using a regular grid, the penumbra intersections could be represented with 2-D BSP trees. These could then be merged together to produce the intersection information.

2.3 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.

First we apply an additional verification test on the shadow relation: the penumbra vertices are projected onto R's plane and the area they define is compared against R, Figure 2. 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 3, and they are joined to make the DM-tree of discontinuities from O, Figure 4. 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 5.

The construction of the single-tree proceeds in three steps. First, the penumbra subtree is constructed by connecting the penumbra vertices, Figure 2. We know that

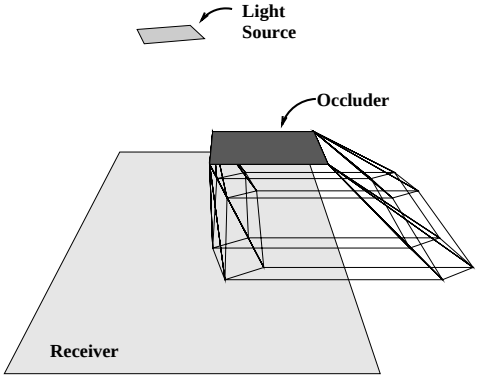

Fig. 1. The source, the receiver and the occluder with the complete set of EV planes.

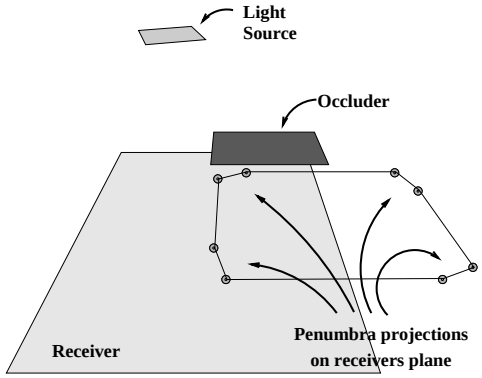

Fig. 2. Penumbra vertices are cast on receiver's plane and checked for intersection.

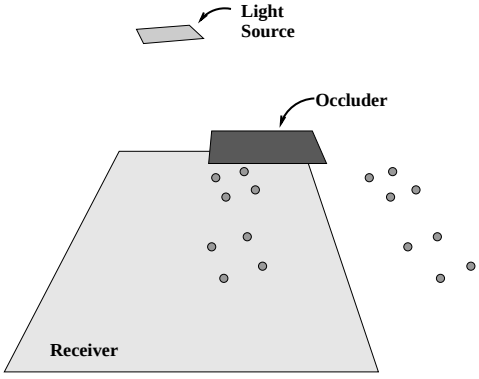

Fig. 3. When an intersection is established the rest of the vertices are also projected.

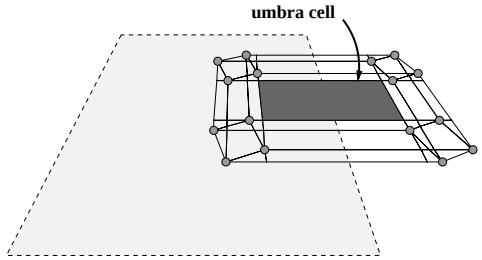

Fig. 4. The single-tree is built using adjacency information in the shadow planes.

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 ISD 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. (For more detail see [5]).

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 [18] with some modifications to allow for sub-hyperplanes not spanning the entire subspace in which they reside. Nodes with such a property are the boundary edges of the receiver and the penumbra nodes of the single-tree which are only expanded after they reach

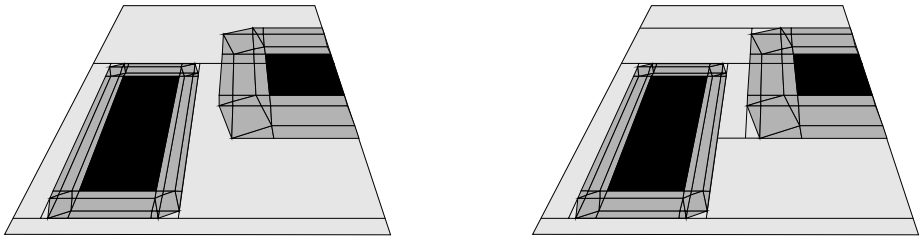

Fig. 5. The single-tree is merged into the total-tree of the face, clipping anything outside and adding construction edges to any penumbra edge not spanning its subspace.

a cell, Figure 5.

2.4 Computing Illumination Intensities on the Vertices

For each vertex we can rapidly extract information on whether it is lit, in umbra or in penumbra, and also exactly which occluders block each penumbra vertex before we begin to illuminate it.

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 6. 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 6). In cases where a D^0 vertex is covered by the penumbra caused by a 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. Since we ignored EEE events, some vertices classified as penumbral might actually be completely occluded. For these vertices there will be no visible part of the source.

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

3 Dynamic Modifications

The main purpose of this research is to build a shadow algorithm that can take advantage of the spatio-temporal coherence in interactive applications and allow for

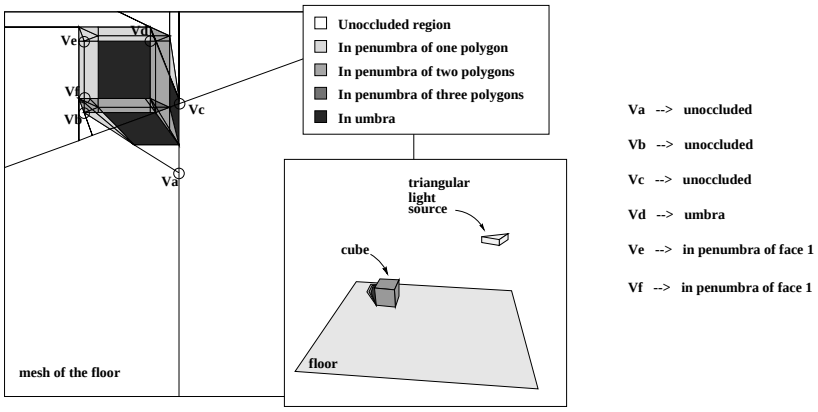

Fig. 6. Possible classifications of a vertex during illumination.

the necessary modifications to be performed in a fraction of the normal construction time. This is made possible due to a combination of certain aspects of the algorithm. First, the shadow overlap cube significantly localises the operations performed to only a small superset of the affected polygons. Space subdivision schemes have been used in previous DM solutions ([8]) but since they trace each discontinuity surface independently, they fail to identify all polygons concerned during scan-conversion (small polygons fully in umbra or penumbra are only found on a separate step [10]).

Second, the use of BSP tree merging for adding the discontinuities which induces an explicit classification of the cells and provides a means for identifying the concerned vertices during interaction. For example in Figure 7, as the moving object comes near object-a, the new discontinuities are computed. A traditional method can find the newly created vertices and examine them for illumination, but not so easily 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 modelled by two operations: deletion and/or addition of objects. After each transformation we illuminate the relevant vertices.

3.1 Removing an Object

To remove the polygons of an object we first delete their entries from the shadow overlap cube either by scan-converting them again (slower) or by using lists of the grid elements they go through stored at the initial scan-conversion (more space consuming). Then we remove the polygons from the BSP tree using the method described in [6].

Each polygon holds reference to the receiver polygons upon which it has cast a shadow during the construction of the mesh. When removing the polygons of an object, the receivers are collected into to a 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:

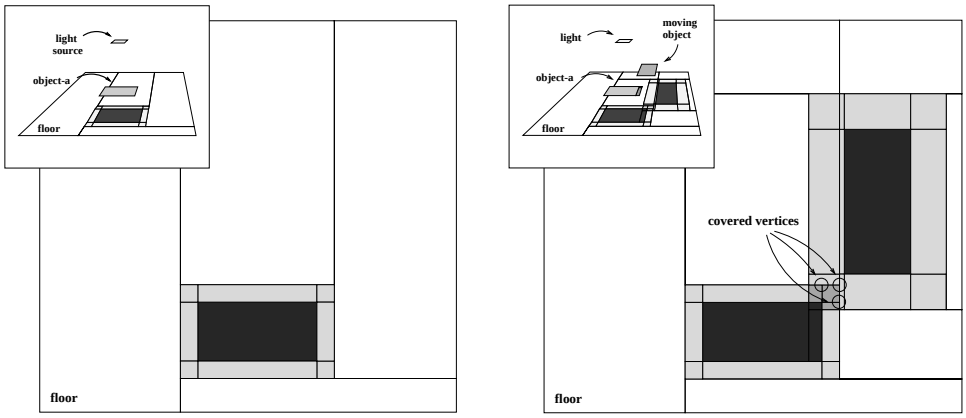

Fig. 7. Merging allows for easy identification of the vertices with changed intensity when a polygon is added or deleted

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.
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.

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 in two parts, removing it will result in two unconnected subtrees that will have to put together. Of course 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 is an optimisation of BSP merging that takes advantage of the fact that they are defined in separate subspaces.

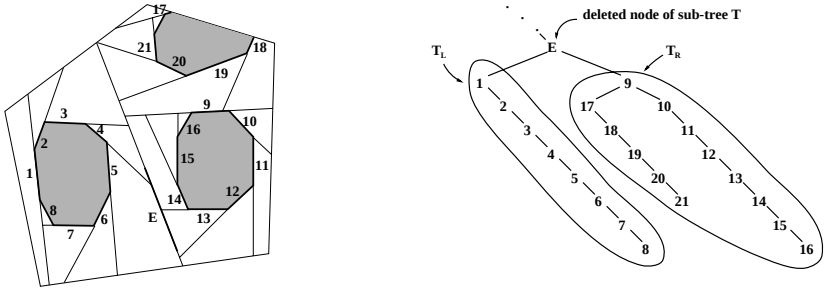

Fig. 8. Discontinuities on a polygon and its tree representation

To delete the marked node of edge E that has two non-empty subtrees (T_L and T_R), Figure 8, the following three steps are performed:

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.

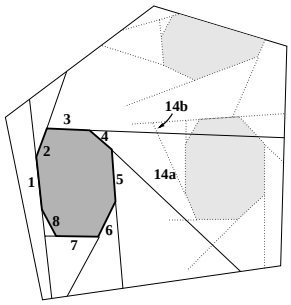

Fig. 9. T_L is expanded to the whole subspace

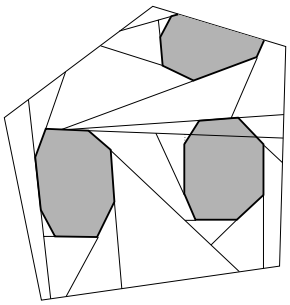

Fig. 10. T_R added to T_L to form one tree

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. The estimation of the cost we use 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 penumbra edges in b.

In Figure 8, $E[cost_{T_L}] < E[cost_{T_R}]$ 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 9. 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.

Step 3 T_R is inserted into T_L to form a unified tree, as in Figure 10. 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 9. The top part, 14b, does not contain any part of a discontinuity so it will be removed.

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 be at or near the leaves. A more detail description and analysis of the performance of this method is given in [5].

3.2 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 ordering tree which is traversed to get the new front-to-back order. The new polygons are added to the shadow overlap cube and the other faces sharing tiles with them are found. Shadows are cast between the later and the new polygons.

3.3 Illumination of Vertices

During the deletion and/or addition of objects, the new vertices created and the existing ones witch have a modified visibility of the source are identified and stored into a list. At the end of the frame they are illuminated using the method described in Section 2.4.

4 Results

4.1 Statistics for Initial Construction of the DM

The algorithm was implemented in C on a SUN SparcStation 20, 75MHz and 160MB of RAM. At the time of writing it has only been tested on four small office and cube scenes ranging from 92 to 184 polygons. The time taken for the construction and illumination of the mesh was between 0.77 and 2.24 seconds. This does not include any further triangulation which, as in any DM algorithm would be required. In our method, further triangulation would be accelerated because of the occlusion classification of each mesh element.

The time taken to generate the shadow volumes and process the shadow overlap cube are insignificant (less than 4% of total). Constructing the single DM-trees and merging them to the total DM-tree of the receivers takes between 30 and 45 % while most of the remaining time was taken by the illumination of the vertices.

Following the discussion of Section 2.4 one might have expected the illumination to be less expensive than that found. In fact the efficiency of the illumination step was apparent in the results by the fact that for all scenes, the average number of source/occluder comparisons per vertex was between 1.3 and 2.1. One of the reasons that the percentage of the illumination time is so high 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 11 and Figure 12.

To give an example of how source size influences the performance, we run the *15 cubes* scene (15 randomly placed and oriented cubes in a room) with a source 5 times smaller than the original. The resulting mesh is less complex with fewer vertices and in particular much fewer penumbra vertices, the illumination time went down from 1.10s to 0.35s (with the average number of source/occluder comparisons 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.

Larger scale experiments are required before we can have any conclusive evidence on the performance of the method. However, these preliminary results compare very favourably with results reported for other algorithms and indicate the potential of this method.

4.2 Evaluation of the Incremental Modifications

Selected objects from each scene were moved interactively to test the response of the method. These objects ranged in size from 5 to 30 % of the total scene geometry. The time taken to delete the moving object from the mesh was never greater than 4% of rebuilding the whole mesh, while usually being closer to 1%. 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. The time taken to add an object back as a proportion of the total rebuilding time is equivalent (statistically) to the proportion of the object's geometry to the total scene geometry.

5 Conclusion and Further Work

In this paper a fast discontinuity meshing algorithm has been presented. It uses a shadow overlap cube along with the ordering produced by an augmented BSP tree for identifying potential shadow relations between model polygons.

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.

Due to the structured creation of the DM and the new method for updating the DM-tree dynamically, 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.

One attribute of dynamic environments is that the attention of the user is distracted by the movement so some visual 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 speedup 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. In fact when simulating a realistic scene with a large number of polygons we would probably not want to use all the discontinuities, even for static scenes. Some selection method could be employed [12].

An issue that remains to be 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.

Aknowledgments

This work is funded by the European ACTS Project AC040, COVEN - Collaborative Virtual Environments.

References

1. B. G. Baumgart. Geometric modeling for computer vision. AIM-249, STA -CS-74-463, CS Dept, Stanford U., October 1974.
2. J. F. Blinn. Jim Blinn's Corner: Me and my (fake) shadow. *IEEE Computer Graphics & Applications*, 8(1):82–86, January 1988.
3. A. T. Campbell. *Modelling Global Diffuse Illumination for Image Synthesis*. PhD thesis, Department of Computer Science, University of Texas at Austin, December 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. *Shadow Computation for 3D Interaction and Animation*. PhD thesis, Queen Mary and Westfield College, University of London, February 1996.
6. Y. Chrysanthou and M. Slater. Dynamic changes to scenes represented as BSP trees. *Computer graphics Forum, (Eurographics 92)*, 11(3):321–332, 1992.
7. 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.
8. 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.
9. F. Durand, G. Drettakis, and C. Puech. The 3d visibility complex: A new approach to the problems of accurate visibility. In *Proceedings of the Seventh Eurographics Workshop on Rendering*, pages 245–256, Porto, Portugal, June 1996.
10. N. Gatenby and W. T. Hewitt. Optimizing discontinuity meshing radiosity. In *Fifth Eurographics Workshop on Rendering*, pages 249–258, Darmstadt, Germany, June 1994.
11. 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.
12. S. Hardt and S. Teller. High-fidelity radiosity rendering at interactive rates. In *Proceedings of the Seventh Eurographics Workshop on Rendering*, pages 71–80, Porto, Portugal, June 1996.
13. P. Heckbert. Discontinuity meshing for radiosity. *Third Eurographics Workshop on Rendering*, pages 203–226, May 1992.
14. P. Heckbert. Radiosity in flatland. *Computer graphics Forum, (Eurographics 92)*, 11(3):181–192, 1992.
15. D. Lischinski, F. Tampieri, and D. P. Greenberg. Discontinuity meshingfor accurate radiosity. *IEEE Computer Graphics & Applications*, 12(6):25–39, November 1992.
16. 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.
17. B. F. Naylor. *A Priori Based Techniques for Determining Visibility Priority for 3-D Scenes*. PhD thesis, University of Texas at Dallas, May 1981.
18. B. F. Naylor, J. Amandatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. *ACM Computer Graphics*, 24(4):115–124, 1990.

19. 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.
20. 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.
21. 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.
22. 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.

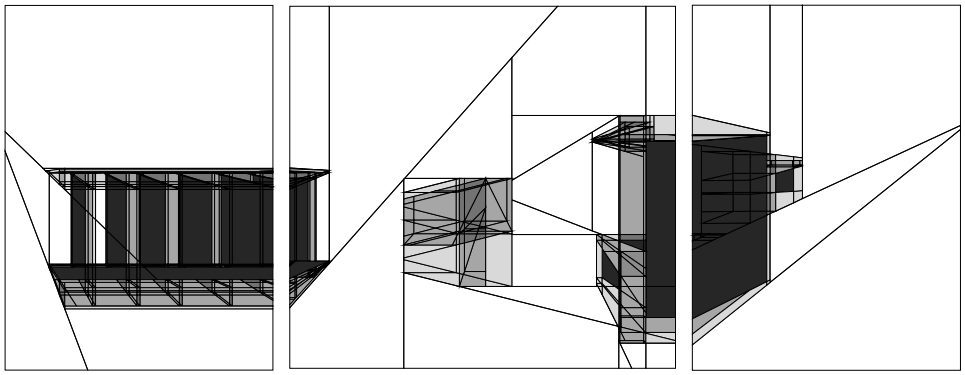

Fig. 11. The mesh of the left wall (left), the floor (middle) and the right wall (right) for *officeA*, 114 polygons

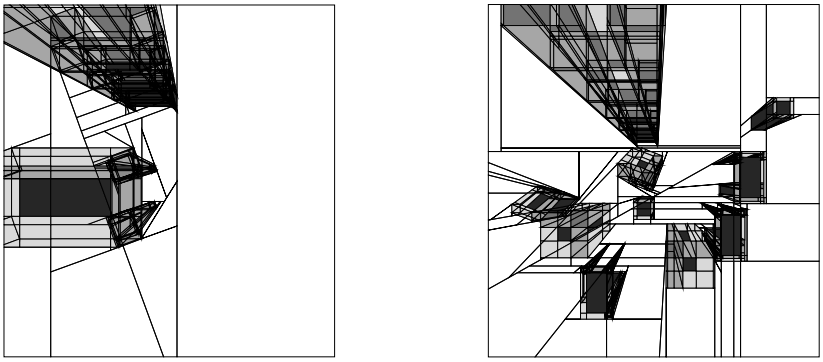

Fig. 12. The mesh on the floor from two objects and a rotated source (left) and the *officecubes* scene (right), 92 and 184 polygons respectively