Interactive High-Quality Soft Shadows in Scenes with Moving Objects

Céline Loscos, George Drettakis

EUROGRAPHICS '97 / D. Fellner and L. Szirmay-Kalos
(Guest Editors)

Volume 16, (*1997*), *Number 3*

Interactive High-Quality Soft Shadows in Scenes with Moving Objects

Céline Loscos and George Drettakis[†]

iMAGIS[‡]/GRAVIR-INRIA,
BP 53, F-38041 Grenoble Cedex 9, FRANCE.

Abstract

Interactive rendering of soft shadows (or penumbra) in scenes with moving objects is a challenging problem. High quality walkthrough rendering of static scenes with penumbra can be achieved using pre-calculated discontinuity meshes, which provide a triangulation well adapted to penumbral boundaries, and backprojections which provide exact illumination computation at vertices very efficiently. However, recomputation of the complete mesh and backprojection structures at each frame is prohibitively expensive in environments with changing geometry. This recomputation would in any case be wasteful: only a limited part of these structures actually needs to be recalculated. We present a novel algorithm which uses spatial coherence of movement as well as the rich visibility information existing in the discontinuity mesh to avoid unnecessary recomputation after object motion. In particular we isolate all modifications required for the update of the discontinuity mesh by using an augmented spatial subdivision structure and we restrict intersections of discontinuity surfaces with the scene. In addition, we develop an algorithm which identifies visibility changes by exploiting information contained in the planar discontinuity mesh of each scene polygon, obviating the need for many expensive searches in 3D space. A full implementation of the algorithm is presented, which allows interactive updates of high-quality soft shadows for scenes of moderate complexity. The algorithm can also be directly applied to global illumination.

Keywords: Illumination, soft shadows, incremental update, discontinuity meshing, backprojection, dynamic scenes.

1. Introduction

High quality rendering for scenes lit by *area* light sources is an important component of any lighting system. Such display is typically performed using ray-casting to successfully render the soft shadows or *penumbra* [1]. An alternative approach is the use of discontinuity meshing with backprojections. The *discontinuity mesh* provides an initial decomposition of the scene which is used to create a subdivision into simple polygons, whose edges are well adapted to the penumbra contours and the discontinuities of illumination in the interior of partially shaded regions [2, 3]. The computation of the full discontinuity mesh (capturing all illumination discontinuities due to the light source) permits the calculation of *backprojections* [2, 4]. The backprojection structure encodes exact visibility of any point in the scene with respect to the light source, thus providing *exact* illumination (irradiance) values at the vertices of the subdivision and at any point in the penumbra. Very high quality rendering of soft shadows can be achieved in this manner, using a polygonal decomposition on a graphics hardware pipeline. We are therefore able to interactively visualise scenes with accurate soft shadows on graphics workstations as long as the objects in the scene do not move. If the geometry changes, existing algorithms require the complete recomputation of the discontinuity mesh and the backprojections, which is prohibitively expensive, and definitely precludes user interaction.

In this paper we present an algorithm which allows interactive rendering of high quality shadows for scenes where objects move, which we call *dynamic* scenes. Our new algorithm is based on discontinuity meshing and backprojections,

† *E-mail:* {*Celine.Loscos* | *George.Drettakis*}*@imag.fr*

‡ iMAGIS is a joint research project of CNRS/INRIA/UJF/INPG.

thus providing accurate soft shadows for interactive display. To achieve interactive update rates for dynamic scenes, the algorithm exploits spatial coherence of the required modifications to the data structures related to shadows and the local nature of changes in the discontinuity mesh. This locality is encoded in the rich structure of the discontinuity mesh, which permits us to identify the visibility events by simply examining the planar discontinuity mesh on the polygons.

This novel algorithm is useful in several contexts. Since primary illumination is dominant in many situations, high quality direct lighting with soft shadows can be used as a standalone interactive visualisation program offering a much higher level of realism compared to traditional point-source interactive lighting systems. In addition the algorithm can be used as a first interactive design phase before a global illumination solution, for object placement and general modeling in a scene. Although we treat only direct illumination, this approach can be applied in the context of global illumination. Our method thus opens an interesting avenue of research for combined discontinuity meshing/hierarchical radiosity approaches such as those previous presented [5, 6], in the context of dynamic scenes.

The strategy adopted to achieve interactive display of soft shadows with moving objects is based on two main components: (i) intelligent data structures which localise and thus accelerate access to changing visibility information and (ii) an efficient update algorithm which takes into account both spatial coherence and visibility information contained in the mesh. After presenting related previous work in Section 2, we present the data structures used in Section 3 and the incremental shadow update algorithm is described in Section 4. We next present the results of the implementation in Section 5. In Section 6 we discuss future work and conclude.

2. Previous work

2.1. Illumination in Dynamic Scenes

Most previous work in illumination for dynamic environments has concentrated on global solutions. Some research has been performed in ray-tracing (e.g., [7]), which is specifically related to the view-dependent nature of ray-tracing, and is thus unsuitable for rendering approaches based on interactive visualisation using current graphics hardware.

The output of radiosity algorithms was used very early on with graphics hardware, permitting realistic interactive walkthroughs albeit with the restriction to static environments. The first attempt to remove this restriction was the approach of Baum [8], in which motion was predetermined and the region of space affected was preprocessed to accelerate the calculation of form-factors for each frame.

More involved approaches, based on the progressive refinement radiosity algorithm were presented by George et al. [9], and also Chen et al. [10]. In these approaches moving shadows were treated by re-shooting energy to remove them from their previous positions, shooting negative energy to reinstate them elsewhere. Modifications in the environment had to be ordered by a queue due to the nature of progressive refinement. Special attention was paid to efficiently treating shadows due to direct illumination. A more involved data structure for maintaining shadow form-factor lists has been presented [11] for progressive refinement radiosity.

A first approach for hierarchical radiosity has been presented [12]. A similar approach was presented by Shaw [13]. In this work, a “motion volume” was used to identify the links affected by the displacement of an object.

2.2. Discontinuity Meshing for High-Quality Illumination

For polygonal scenes lit by area sources, discontinuity meshing [14, 15, 2, 4], was introduced to improve the quality of rendering for scenes containing soft shadows. To create the discontinuity mesh with respect to a source, *discontinuity surfaces* are cast into the environment. These surfaces are the interaction of an edge and a vertex (*EV* surface) or three edges (*EEE* surface [16, 17]). The reader unfamiliar with discontinuity surfaces is strongly encouraged to refer to the appropriate references [16, 2, 4]. Algorithms which treat all such events [16, 2, 4], can then incrementally compute the *backprojection* data structure, which encodes all visibility information with respect to the source.

A first approach for dynamic environments rendering using discontinuity meshing with BSP trees was developed for point light sources by Chrysanthou and Slater [18].

Worral et al. have presented a new approach for area sources [19]. In their method, illumination is computed on a triangulated discontinuity mesh in the context of a progressive-refinement radiosity method. The discontinuity mesh vertices are updated by taking into account certain visibility changes. Triangular mesh coherence is maintained and radiance values are updated for each triangle of the mesh by shooting the irradiance difference compared to the previous mesh. An interesting criterion is introduced, determining whether a change in visibility occurs in the mesh. This approach is limited to *EV* discontinuity surfaces, with vertex *V* on the source. Moreover, the focus of Worral et al.’s work is the update of the triangulation, whose cost is minimal compared to the casting of discontinuity surfaces, especially in complex environments. It is important to note that the approach presented in [19], computes an incomplete mesh, since *EEE* and other important discontinuity surfaces are ignored. As a consequence, backprojections cannot be computed. Visibility must thus either be approximated (e.g., by ray-casting), or be calculated by clipping the entire scene against the source, which is extremely expensive. Such visibility computation typically dominates the computation time [5].

In contrast, our new approach is totally different, since the complete discontinuity mesh and backprojections are incrementally updated at each frame. The exact visible part of the source can thus be determined very cheaply at any point in the penumbra, without a visibility calculation, since this information is encoded with the backprojections [2]. Thus at every frame, we have exact (analytical) irradiance values for all the vertices in the mesh. Before presenting the complete algorithm, we describe important data structures used in the algorithm.

3. Data Structures for Efficient Update

In this section we present the data structures used to accelerate the update of shadows in dynamic scenes. In what follows we define as *static* the edges and vertices which belong to static objects, i.e. objects which do not move. The object that moves will be referred to as the *dynamic* object. We define as *dynamic* edges and vertices which belong to the dynamic object. As a consequence, we call *dynamic discontinuity surfaces*, the *EV* or *EEE* surfaces which are defined by at least one edge or vertex of the dynamic object, and *static discontinuity surfaces* those which are defined entirely by static edges or vertices. Finally, a *mesh* edge or *mesh* vertex, is a two-dimensional edge or vertex which is part of the planar discontinuity mesh calculated on each scene polygon. We use a *winged-edge* data structure used to store the mesh and access it efficiently [2]. The deletion of mesh edges can thus be performed locally and rapidly, as well as the incremental update of backprojections.

The three data structures used to localise and thus accelerate access to information which modifies visibility and thus shadow calculations at each frame, are the following: (a) discontinuity surface storage in the spatial subdivision structure, (b) the motion volume and (c) intersection lists for modified discontinuity surfaces.

3.1. Storage of the Discontinuity Surfaces in the Spatial Subdivision

The scene is decomposed into a regular grid [20], used for efficient casting of discontinuity surfaces [2]. Each voxel contains the list of polygons that cut it. In addition to this we add the list of discontinuity surfaces which intersect the voxel. This list is created on-the-fly, during the propagation of discontinuity surfaces. An example of this list is shown in Fig. 1.

The lists of discontinuity surfaces associated with each voxel allow the rapid identification of all visibility events affecting an area of space, by simply traversing the corresponding voxels. As a consequence, we can perform efficient incremental updates in the region of a moving object.

The storage overhead of the lists is small (between 65 and 300 Kb) for the test scenes presented in the results (see Section 5), which use a moderately-sized grid (15x15x15).

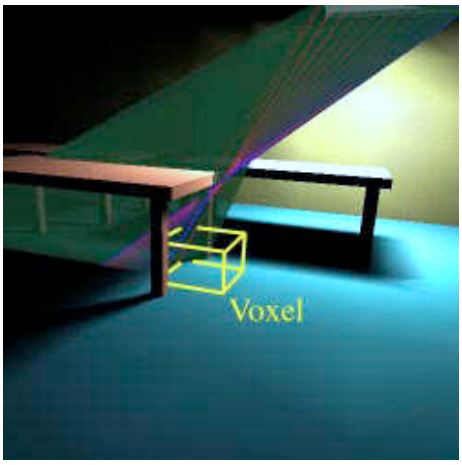


Figure 1: *Discontinuity surfaces in a voxel (see also colour section).*

3.2. Construction of a Motion Volume

The region of space for which visibility is affected by the motion of an object is entirely limited by the *maximal* (i.e. delimiting the frontier between light and penumbra) edge-vertex (*EV*) discontinuity surfaces defined by the light source and the polyhedron of the moving object for the initial and the final position. In addition there is no change in visibility in the region of space between the dynamic object and the light. As an illustration see Fig. 2, where the maximal discontinuity surfaces are shown as the dark grey surfaces, containing all interior discontinuity surfaces, such as that shaded in light grey.

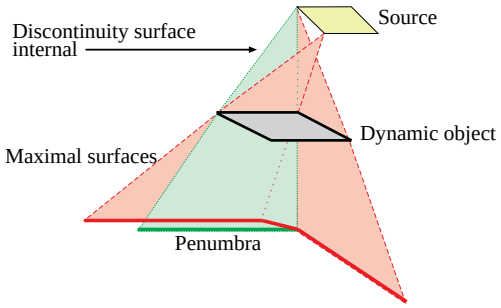


Figure 2: *The maximal surfaces are shown in dark grey and the interior surfaces in light grey. Notice that the maximal surfaces encompass all the others.*

Given this property, we can define a simplified approximation to the exact volume in space affected by the motion which we call a *motion volume*. This volume is delimited by a plane parallel to the source above the uppermost side (i.e. closest to the source) of the dynamic object, a plane parallel to the source plane which is completely outside the

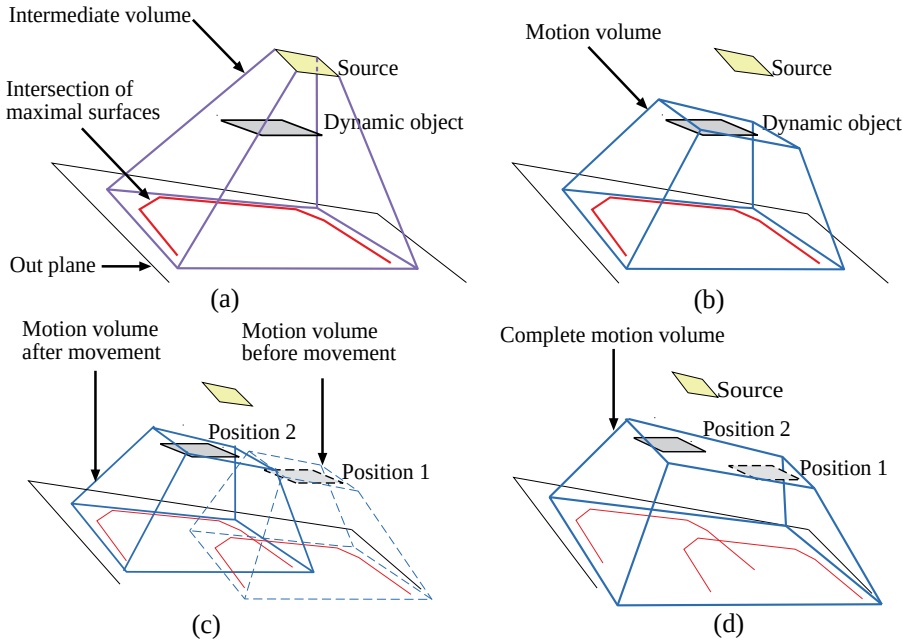


Figure 3: *Motion Volume construction: (a) The maximal surfaces of the dynamic object are intersected with the outplane: a plane parallel to the source, and tangent to the bounding volume of the scene. The 2D bounding box of the contour of the maximal surfaces (dark grey segment on the outplane) is found. A four-plane volume is then constructed with the 2D bounding box of the source. (b) The volume is cut by a plane parallel to the source above the object, (c) Volumes for position before and after the move (d) Complete motion volume.*

scene, and four planes surrounding the maximal surfaces (see Fig. 3(a)-(b)).

In our current implementation three volumes are created. One for the first position of the object, one for the final position, and one that is the bounding volume of the two previous volumes (see Fig. 3(c)-(d)). Since we consider a small discrete motion at each frame, we currently use the bounding volume as the motion volume for updates. For larger displacements, the use of the two independent volumes would be more appropriate since their intersections would be small or inexistant. Otherwise the bounding volume would include too much unchanged space.

Note that this construction does not limit the dynamic object motion in any way. At any frame, the previous and current positions are available, and thus the user may interact freely with the dynamic object. Given the construction of the bounding volume, this motion can be of any type (translation, rotation), a scale operation, or a discrete curved trajectory.

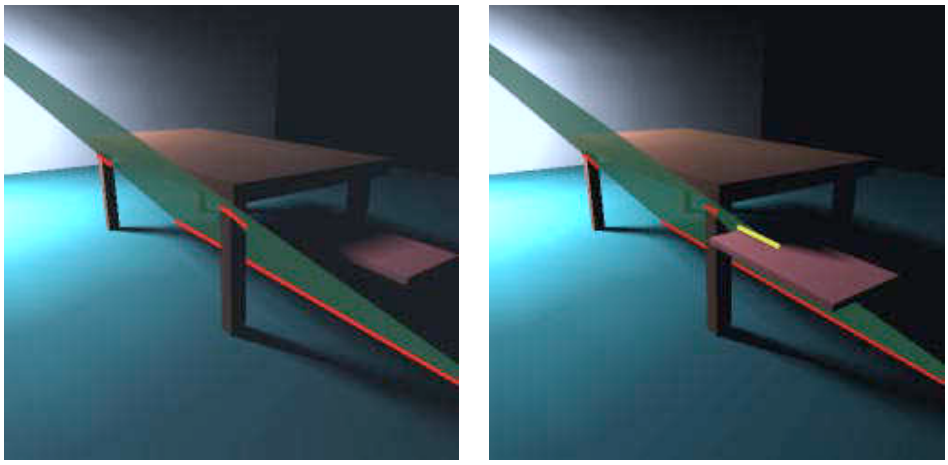
3.3. Storage of Intersection Information with the Discontinuity Surfaces

The casting time for the discontinuity surfaces is mainly concentrated in the testing and intersection parts of the casting operation. Due to the richness of information already in the mesh, we can avoid a large part of this cost by storing some additional information with the discontinuity surfaces.

Consider the case of movement shown in Fig. 4(a) and (b), corresponding respectively to the position of the dynamic object before and after the move. We know that the only possible change in visibility for surfaces such as those shown in Fig. 4 can be caused by the dynamic object. As a consequence we do not need to search or intersect the discontinuity surface with any other object in the environment at each frame. If the dynamic object were moving away from the discontinuity surface after Fig. 4(a) it is evident that we would not need to recompute the intersection of the discontinuity surface with the environment, nor recompute the visibility on the surface.

The intersections of the polygons with the discontinuity surface are stored *before* visibility processing. An example is shown in Fig. 5(a). Notice that after visibility processing, which occurs as a 2D operation in the plane of the discontinuity surface (or 2-D parametric space for EEE), the intersections are changed, resulting in the final mesh edges inserted in the discontinuity mesh (e.g., two mesh edges for the floor - see Fig. 5(b)).

This list is stored with the discontinuity surface. For example the list i_1, i_2, i_3, i_4 in Fig. 5(a) is stored with the EV surface shown. When treating a static discontinuity surface at



(a) (b)

Figure 4: *Dynamic object motion (a) the dynamic object (floating parallelepiped) does not cut the static EV discontinuity surface, (b) the object moves forward and cuts the discontinuity surface.*

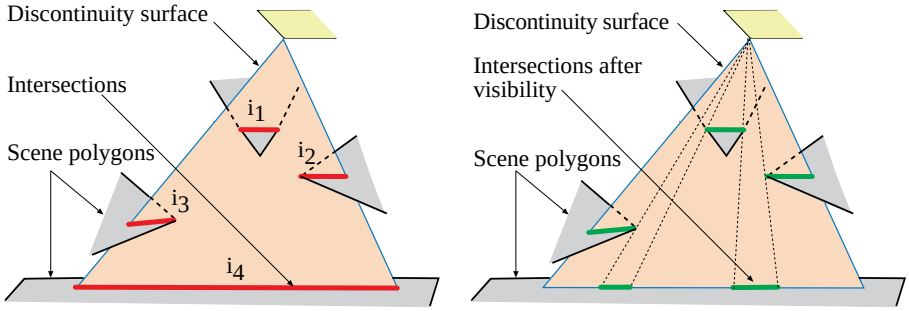


Figure 5: *Intersection information storage (a) the intersections i_1, i_2, i_3, i_4 (in dark grey) are stored with the discontinuity surface before visibility computation. (b) the actual intersections (in light grey) after the visibility computation performed in the plane of the discontinuity surface.*

a given frame, we only perform a new intersection with the dynamic object. We thus avoid the cost of searching for and performing intersections with all the other (static) objects in the scene.

3.4. Input Scenes

As shall be seen later, we will be identifying visibility changes based on information in the mesh (see Section 4.3). In order to find all visibility changes, input scenes need to be closed environments. This ensures that all discontinuity surfaces have intersections with at least one scene polygon at any time. This guarantees that all the information required can be found in the mesh.

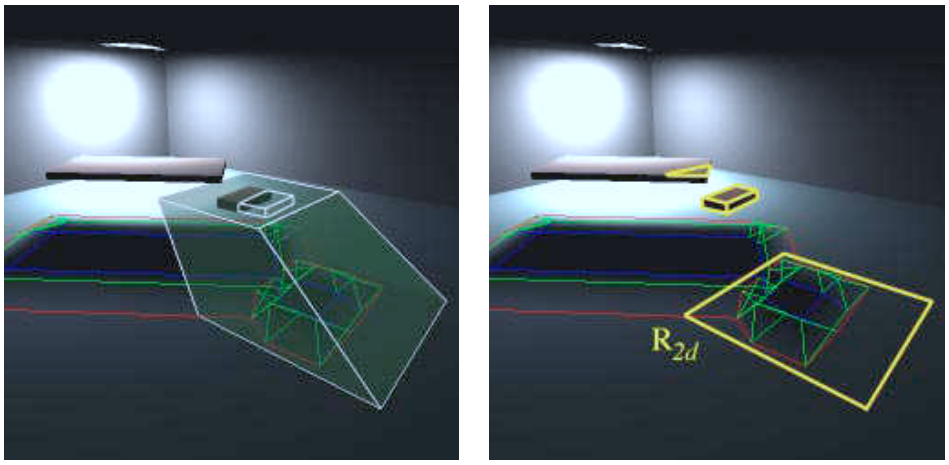
Considering only such scenes is not a strong restriction, since open environments can easily be changed by enclosing the scene in a box.

In addition, we suppose that the area source cannot move since the entire mesh would have to be updated. Techniques such as the Visibility Skeleton [21] are probably more appropriate for this type of update, and will undoubtedly lead to efficient discontinuity mesh and backprojection algorithms for moving sources (see also Section 6).

4. Update Algorithm

Given the storage of discontinuity surfaces in the spatial subdivision structure, the creation of the motion volume and the storage of intersections with the discontinuity surfaces, we can now present the machinery required to perform efficient updates of the discontinuity mesh and backprojections.

The shadow update algorithm needs to perform the following steps: (a) identify the volume of space modified, and collect related discontinuity surfaces which need to be updated (function *findChangedSpaceAndDS*); (b) identify and process the region modified on each input polygon; (c) identify the visibility changes for each modified discontinuity surface (function *findAndProcessVisibilityChanges*); (d) cleanup the



(a) (b)

Figure 6: *(a) Intersection of motion volume with polygons (see also colour section), (b) modified regions R_{2d}, (in white).*

parts of the mesh which are invalid within each region; (e) update the mesh, and finally (f) update the shadows and the illumination.

In this manner we will have performed the necessary updates in the parts of the discontinuity mesh affected, and thus the soft shadows will correspond to the new position of the object. Both spatial coherence using the motion volume, and the information in the mesh are used to identify potential changes in visibility. Note that after these updates, the discontinuity mesh and backprojections are entirely recomputed, and the values of irradiance in the penumbra correct. We examine each step of the algorithm in detail.

4.1. Identification of Affected Discontinuity Surfaces and Mesh Region

We first identify (using the grid) all discontinuity surfaces and polygons contained in the motion volume. The discontinuity surfaces concerned are inserted into a list DS_d for the dynamic surfaces, and DS_s for the static surfaces. The intersection R_{2d} of the volume with each polygon is then computed, as shown in Fig. 6(a). The intersections R_{2d} of the volume with the polygons concerned are outlined in Fig. 6(b) in white. This two-dimensional polygon R_{2d} is in effect the modified region for each input polygon.

4.2. Processing of Mesh Edges in Modified Regions

Due to the winged-edge data structure used to store the mesh, we can efficiently identify the mesh edges which are modified. In particular, we find the mesh face containing a corner of R_{2d} and search all neighbouring faces recursively until no mesh edges crossing or contained in R_{2d} can be found.

For each mesh edge we identify those which need to be deleted. All dynamic mesh edges will be removed, as well as the static mesh edges for which a change in visibility occurs, with respect to the dynamic object. More precisely *shouldDeleteStatic*(*e*) is true only if the discontinuity surface associated to the edge *e* intersects the dynamic object at its initial or its final position. The corresponding static discontinuity surfaces are marked as changed. This process is summarised in Fig. 7, and detailed in what follows.

```
processModifiedEdges() {
  foreach input polygon p
    Poly2d R2d = modified region of p
    foreach mesh edge e in R2d
      if e is dynamic
        add e to dynEdgeDelList
      else if shouldDeleteStatic( e )
          add e to statEdgeDelList
}
```

Figure 7: *Modified Mesh Edge Processing*

After processing the edges in the modified regions, we have two lists *dynEdgeDelList* and *statEdgeDelList* which are the mesh edges to be removed when the information they contain is no longer needed.

4.3. Finding and Processing the Visibility Changes in the Modified Regions

Recall that the routine *findChangedSpaceAndDS*() returns two lists which give us all the discontinuity surfaces passing through the motion volume: DS_s for the static discontinuity surfaces and DS_d containing the dynamic discontinuity surfaces.

For each surface, we identify the related visibility changes and perform the appropriate updates required to reflect the dynamic object motion. The process is summarised in Fig. 8.

4.3.1. Static Discontinuity Surfaces

For each static discontinuity surface which is on the list DS_s and has been marked changed, we compute new intersections with the polygons of the dynamic object, if such intersections exist. Note that a static discontinuity surface may intersect the dynamic object in its upper part, between an object vertex and the source edge, resulting in no mesh edges because of the object occlusion. Therefore a discontinuity surface may interact with the dynamic object without being detected by the previous mesh traversal. The use of the DS_s list is thus very important because it avoids the cost of an object-space search. With this list, we are able to consider such surfaces.

We then modify the intersection list of the discontinuity surface by either adding, deleting or modifying the information encapsulating the intersections of the surface with dynamic object.

```
findAndProcessVisibilityChanges() {
   processStaticSurfaces(DS_s)
   processDynamicSurfaces(DS_d)
}
processDynamicSurfaces(list DS_d) {
   foreach surface ds in DS_d
      if ds is EV
         processEV( ds )
   ...
}
```

Figure 8: *Finding and Processing Visibility Changes*

4.3.2. Dynamic Discontinuity Surfaces

For dynamic discontinuity surfaces, we can easily see that their intersections with the scene polygons always change. In addition, the motion of the dynamic object can result in a change in the visibility configuration of each surface with respect to the static objects (new intersections, disappearance of intersections etc.). Much relevant information is contained in the mesh, and most notably is related to the static mesh edges. We thus avoid the cost of the search of intersections for each dynamic *EV* surface, involving an expensive traversal of many objects in the scene.

The treatment of *EV* surfaces was inspired by Worrall et al. [19] who analyze the intersection of two edges of a mesh and decide whether a change in visibility occurs. In their work, a change occurs if the two corresponding discontinuity surfaces of the mesh edges share the same source vertex. We extend this idea to all types of *EV* edges and present a novel solution for the case of dynamic *EEE* surfaces.

EV Surfaces: Consider the example given in Fig. 9: the dynamic object (the small object on the right), has a dynamic *EV* surface related to the source vertex *V*. Initially it does not cut the static (larger) object. When moving inwards, the dynamic *EV* surface will intersect the corner of the static object. This can be detected of course in three-dimensions, but this

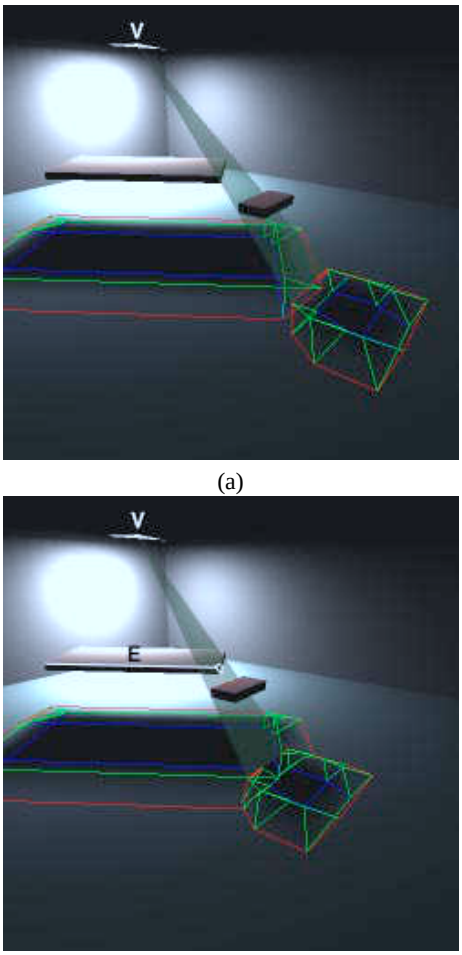
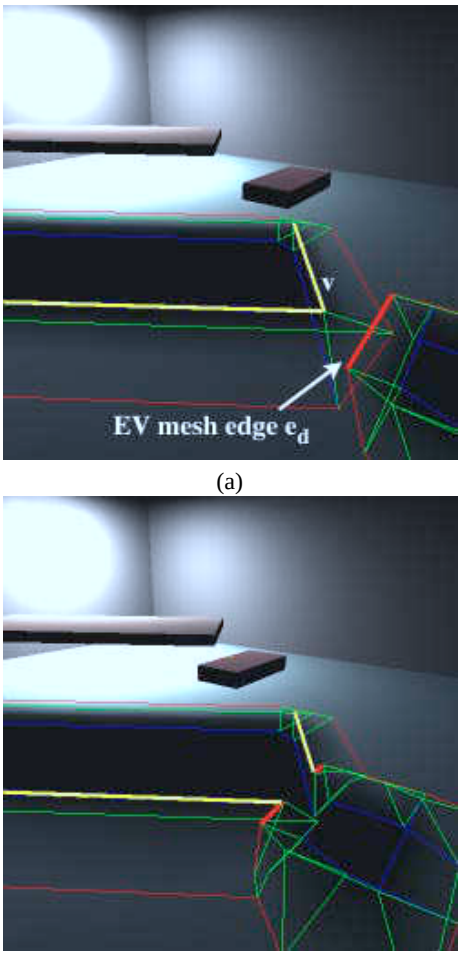


Figure 9: *The small object is dynamic, moving towards the larger object. Dynamic EV discontinuity surface: (a) before the move there is no intersection with the static object, (b) after the move the dynamic surface intersects the static object.*

would imply a costly search in space. Instead, we can directly identify this change in the discontinuity mesh. Consider the mesh edge e_d corresponding to the *EV* surface, shown in grey in Fig. 10(a). Due to the motion, the edge e_d will traverse the mesh vertex *v* (Fig. 10). The mesh vertex *v* is due the crossing of the mesh edges (in white), caused by two static *EV* discontinuity surfaces, due to the *same* source vertex *V* (Fig. 9(b)). Because of this traversal of a mesh edge generated by the same source vertex, we know that there is a visibility change concerning the dynamic *EV* surface, and that it is due to the static object in question.

To determine all such traversals, we need to perform a search in the mesh related to each dynamic discontinuity surface. For each dynamic discontinuity surface, we have stored the list of intersections with the polygons of the scene, for



(a)

(b)

Figure 10: *Dynamic discontinuity surfaces treatment: (a) the EV discontinuity surface of Fig. 9 results in edge e_d in the mesh. In (b) we see its new position. The modified search region for EV is defined by the two positions of e_d. Since the vertex v is crossed, a visibility change has occurred.*

the *previous* position of the dynamic object. We will thus traverse this intersection list, and for each polygon which was intersected, we will find the region defined by the intersection points of the surface with the polygon, before and *after* the move. These correspond to the endpoints of e_d before (Fig. 9(a)) and after (Fig. 9(b)) the move. Within this region, we identify all static mesh edges. We again use the adjacency information of the winged-edge data structure to access these mesh edges rapidly. This is the reason why we do not remove any mesh edges before this step in the algorithm.

We then test to see if the conditions for a change in visibility are satisfied: that is whether the vertex V_s or the edge E_s of the corresponding static *EV* are the same as the edge *E* or vertex *V* of the dynamic discontinuity surface *evDs*. This process is summarised in Fig. 11 for the case of a EV_{src} surface (with vertex *V* on the source and edge *E* on the dynamic object). The $E_{src}V$ (edge on source, vertex on dynamic object) case is treated similarly, by considering the equality of the generating edge *E* with E_s as well as the two vertices defining the edge.

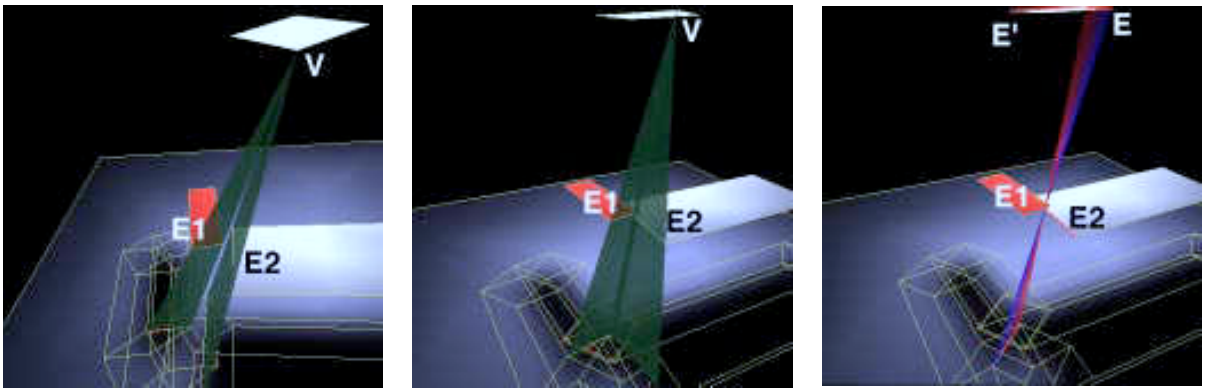
```
processEV( DiscSurface evDs ) {
   updateDStoNewPosition( evDs )
   foreach intersection oldDsi of evDs
      newDsi = evDs →
         computeNewIntersection(dsi → polygon())
      Poly2d p2d_ds =
            findRegionAffected(oldDsi, newDsi )
      switch (evDs →type() )
         case : EV_src
         findVisibilityChange(p2d_ds, evDs )
         ...
}
findVisibilityChangeEV_src
   (Poly2d p2d, DiscSurface evDs ) {
   Edge E = evDs →edge(), E_s
   Vertex V = evDs →vertex(), V_s
   foreach mesh edge e_m in p2d
      DiscSurface ds_s = e_m →getDiscSurface()
      E_s = ds_s →edge()
      V_s = ds_s →vertex()
      if V_s == V {
         P = polygon containing edge E_s
         updateIntersection(evDs, P)
         checkForEEE()
      }
}
```

Figure 11: *Finding Visibility Changes for Dynamic Discontinuity Surfaces*

If a visibility change is identified, the dynamic discontinuity surface is intersected with the corresponding static object, and its intersection list is updated. The same process could be applied to non-emitter *EV* surfaces.

EEE Surfaces. For *EEE* surfaces an algorithm which finds all visibility modifications from the mesh is much more involved, due to the complications implied by their curved nature. Nonetheless, we are capable of determining when a *EEE* surface will be created, maintained or destroyed, in particular for the case in which the discontinuity surface has one edge on the source (this is the most common type of *EEE* surface). This allows us to avoid a costly search for *EEE* surfaces related to the source, which is otherwise required at each frame.

To understand this, consider the two *EV* surfaces in Fig. 12(a), created by a source polygon edge, a polygon edge on the dynamic object and a static polygon edge. When the dynamic object moves, the surfaces will intersect (Fig. 12(b)), and thus two *EEE* surfaces will be created. One such surface is shown in Fig. 12(c) with edges E of the



(a) (b) (c)

Figure 12: *(a) Two EV surfaces which do not intersect (b) Intersection of the EV surfaces (c) One of the two EEE surfaces created. Figures replicated in the colour section.*

source (adjacent to V), E1 and E2. The second EEE surface is built with E1, E2 and the second edge E' adjacent to V. These changes can be determined easily. When testing EV_{src} changes we check to see if the dynamic surface crosses a static surface generated by the same vertex. The creation of the two EEE surfaces is performed by the routine *checkForEEE* (see Fig. 11).

4.4. Edge Cleanup, Mesh and Illumination Update

After processing all mesh edges in the modified region and identifying potential visibility changes we no longer need the mesh edges which will be modified. We thus traverse the lists *dynEdgeDelList* and *statEdgeDelList* and remove the edges from the mesh within the modified region. The winged edge data-structure allows us to perform all removal-insertion operations efficiently and locally within the mesh. Adjacency information is accessed directly from the edge pointers stored in the aforementioned lists. After removal, we are ready to perform the visibility updates required for the static and dynamic surfaces which require them.

We first visit every modified surface of DS_s and DS_d, and perform a two-dimensional visibility operation on the discontinuity surface. This operation is a fast sweep algorithm which processes the intersection information stored with the discontinuity surface. Recall that this information always corresponds to the geometric state *before* visibility processing. This operation costs much less than a complete re-cast of a discontinuity surface which would involve a search in $3D$ and the re-intersection with the scene objects.

Once the visibility is performed, we insert the segments into the discontinuity mesh. These segments are thus correctly updated for occlusion.

We now have a discontinuity mesh which is completely up to date with respect to the new position of the dynamic object. We simply update the backprojection information in the faces which were modified. These faces were marked during the deletion and insertion of mesh edges. We incrementally traverse the faces changed and update the backprojections concerned. The same incremental algorithm as that presented in [17, 2, 4] is used. Since the number of modified faces is small, we can efficiently compute all the exact visibility information in the penumbra very efficiently.

The final step required is the update of the mesh vertex illumination values, which again is restricted to the mesh faces modified. This operation is again very efficient, since the backprojections compactly encode *complete* and *exact* visibility information. We thus rapidly compute exact irradiance values on the vertices in the penumbra which have changed.

It is important to note that the result of the update algorithm is not an approximation: at every frame, the solution is exact, and results in a mesh computed as if we were performing the entire re-computation of the discontinuity mesh and the backprojections.

5. Implementation and Results of the Update Algorithm

We have implemented the update algorithm and tested it on an Indigo2 R4400 running at 150MHz. The three test scenes are shown in Fig. 13. The scenes contain respectively 145, 307 and 475 polygons. The dynamic object is a box floating above the desk and its movement is given as four consecutive positions. We perform two different tests. The motion for Test 1 is shown in the sequence of Fig. 17, the motion for Test 2 in Fig. 18.

In Table 1 t_{TS} corresponds to the time which is required if the entire discontinuity mesh is to be recalculated at each frame. The time t_{TD} is the total time spent by our algorithm to update the mesh and the backprojections, as well as the illumination. All times are in seconds. The column *s* shows the

(a) (b) (c)

Figure 13: *(a) Scene 1 (b) Scene 2 (c) Scene 3. Figures replicated in the colour section.*

speedup (ratio between t_{TS} and t_{TD}). The additional memory overhead for the storage of the intersection lists is on average 19 Kb for Scene 1, 46 Kb for Scene 2 and 60 Kb for Scene 3, in what concerns Test 1, and 17 Kb for Test 2.

As we can see, the update times are interactive for Test 1, between 1.2 seconds per frame for the simplest scene and (containing 145 polygons), to 2.5 seconds/frame for the most complex scene containing almost 500 polygons. In addition, notice that the additional memory required is small (less than 50Kb) for Scene 2. For Scene 3 (475 polygons) speedup can reach 90 times, compared to the recomputation of the complete mesh at each frame.

The localisation of the modified space has the benefit that the cost of the update algorithm does not depend heavily on the complexity of the rest of the scene. Notice that update times seem to grow sub-linearly with respect to scene complexity (number of polygons).

In Test 2 (performed only on Scene 1), a different movement of the dynamic object is performed, with greater interaction with the other objects (see Fig. 18). The visibility complexity is thus augmented by the number of static surfaces treated and the complication of the mesh. Notice that the update takes between 1.3 and 2.7 seconds. The additional cost is thus not overwhelming.

Our implementation is definitely unoptimised, and we thus believe that significantly improved update rates could be achieved by fine-tuning.

6. Summary, Discussion and Future work

The algorithm presented here provides accurate soft shadow updates for dynamic scenes at interactive rates. We first presented data structures that provide rapid access to relevant information, by exploiting spatial coherence. These structures permit local treatment of the visibility update. The inherent structure of the discontinuity mesh was then used to find and update local changes of visibility for both static and dynamic discontinuity surfaces. *EV* and *EEE* surfaces are treated in this manner. Finally, backprojections and lighting are efficiently updated exclusively in the parts of the mesh which have changed. Thus at each frame, analytic irradiance values are computed, resulting in high quality soft shadows.

Large scenes (more than several thousand polygons) cannot be directly treated with the implementation presented here. This is mainly due to problems of numerical precision. Numerical robustness problems occur during the computation of the intersections between objects and discontinuity surfaces and also during the calculation of the arrangement of the line segments forming the winged-edge data structure. Both problems can be addressed by adopting a symbolic computation approach based on *extremal stabbing lines* as described in the context of the Visibility Skeleton (VS) [21]. All discontinuity surface/object intersection calculations can be replaced by the extremal stabbing lines, and the adjacency information available in the VS can be used to overcome the problems of the topological construction.

Other improvements of the our method should also be investigated. In particular, the use of a uniform grid, although simple to program, is definitely inefficient for more complex scenes. The use of a recursive grid or an octree type structure should provide interesting results.

Furthermore, depending on the movement of the object, we could also use individual volumes for the two positions and apply the algorithm presented for each. If the object moves only a little, we can use the bounding volume, whereas if the object moves a lot, resulting in negligible overlap between the initial and final volumes, it is probably better to use the two volumes.

More importantly, this paper opens a direction of research which will lead to an algorithm which limits the updates only to those strictly necessary. The incremental method presented for dynamic surfaces indicates a potential for such an

Pos.	t_{TS}	t_{TD}	s
Pos 1	57.67	1.36	42.4
Pos 2	57.86	1.25	46.3
Pos 3	58.13	1.24	46.9
Pos 4	58.55	1.27	46.1

Results for Test1: Scene 1 (145 polygons)

t_{TS}	t_{TD}	s
122.33	1.79	68.3
123.91	1.81	68.5
125.16	1.91	65.5
125.97	1.99	63.3

Scene 2 (307 polygons)

t_{TS}	t_{TD}	s
222.51	2.33	89.4
225.75	2.35	79.7
227.60	2.48	84.3
230.22	2.46	85.3

Scene 3 (475 polygons)

Pos.	t_{TS}	t_{TD}	s
Pos 1	58.74	1.29	45.5
Pos 2	60.53	1.91	31.7
Pos 3	61.24	2.23	27.5
Pos 4	61.69	2.66	23.2

Results for Test 2 (Scene 1)

Table 1: *Results for the Update Algorithm.*

approach. What is needed for this type of algorithm is an exhaustive classification of all events which occur in the discontinuity mesh in time, and the corresponding actions which must be taken. Optimality may thus be achieved by developing a "sweep" algorithm in time. Approaches similar to the Visibility Complex [22] or the more recent Visibility Skeleton [21] will prove useful in this direction.

Finally, the algorithm developed here should prove very useful in the context of mixed hierarchical radiosity/discontinuity meshing approaches [6]. In particular, a method similar to that described in [13] could be combined with our approach to achieve interactive updates for global illumination.

7. Acknowledgments

Thanks to Seth Teller for suggesting the treatment of static surfaces and Frédo Durand for a fruitful discussion on the treatment of dynamic surfaces.

References

1. R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing", *Computer Graphics*, **18**(4), pp. 137–147 (1984). Proceedings SIGGRAPH '84 in Minneapolis (USA).

2. G. Drettakis and E. Fiume, "A fast shadow algorithm for area light sources using back projection", in *SIGGRAPH 94 Conference Proceedings (Orlando, FL)* (A. Glassner, ed.), Annual Conference Series, pp. 223–230, ACM SIGGRAPH, (July 1994).

3. G. Drettakis and E. Fiume, "Structured penumbral irradiance computation", *IEEE Transactions on Visualization and Computer Graphics*, **2**(4), pp. 299–313 (1996).

4. A. J. Stewart and S. Ghali, "Fast computation of shadow boundaries using spatial coherence and backprojections", in *SIGGRAPH 94 Conference Proceedings (Orlando, FL)* (A. Glassner, ed.), Annual Conference Series, pp. 231–238, ACM SIGGRAPH, (July 1994).

5. D. Lischinski, F. Tampieri, and D. P. Greenberg, "Combining hierarchical radiosity and discontinuity meshing", in *SIGGRAPH 93 Conference Proceedings (Anaheim, CA)* (J. T. Kajiya, ed.), Annual Conference Series, pp. 199–208, ACM SIGGRAPH, (Aug. 1993).

6. G. Drettakis and F. Sillion, "Accurate visibility and meshing calculations for hierarchical radiosity", in *Rendering Techniques '96* (X. Pueyo and P. Shroeder, eds.), pp. 269–279, Springer Verlag, (June 1996). Proceedings of 7th Eurographics Workshop on Rendering in Porto, Portugal.

7. N. Briere and P. Poulin, "Hierarchical view dependent structures for interactive scene manipulation", in *SIGGRAPH 96 Conference Proceedings (New Orleans, LO)* (H. Rushmeier, ed.), Annual Conference Series, pp. 83–91, ACM SIGGRAPH, (Aug. 1996).

8. D. R. Baum, J. R. Wallace, M. F. Cohen, and D. P. Greenberg, "The back-buffer algorithm : an extension of the radiosity method to dynamic environments", *The Visual Computer*, **2**, pp. 298–306 (1986).

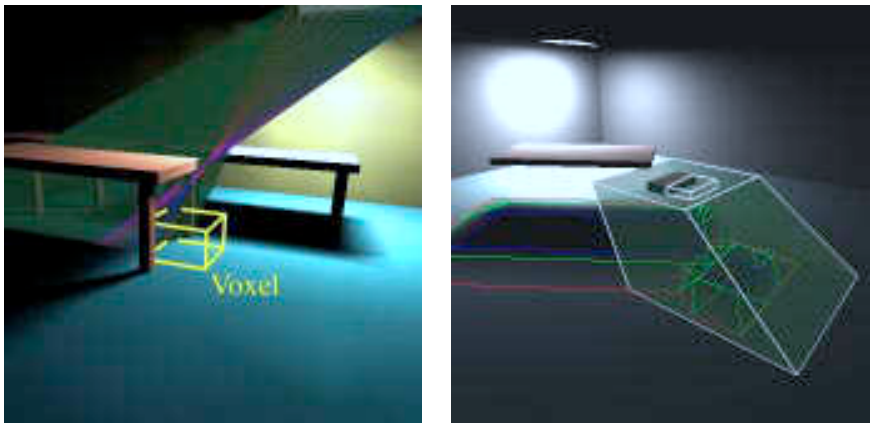
9. D. W. George, F. Sillion, and D. P. Greenberg, "Radiosity redistribution for dynamic environments", *IEEE Computer Graphics and Applications*, **10**(4), (1990).

10. S. E. Chen, "Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system", *Computer Graphics*, **24**(4), pp. 135–144 (1990). Proceedings SIGGRAPH '90 in Dallas (USA).

11. S. Müller and F. Schoffel, "Fast radiosity repropagation for interactive virtual environments using a

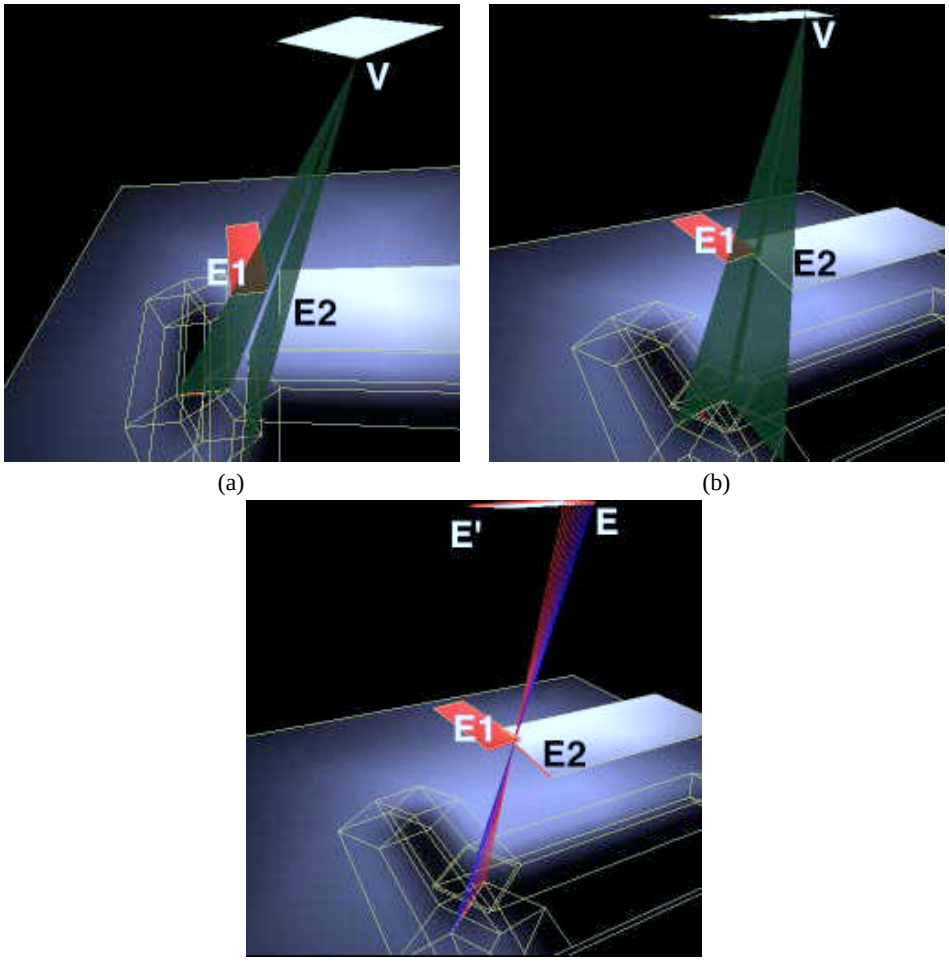
shadow-form-factor-list", in *Photorealistic Rendering Techniques* (G. Sakas, P. Shirley, and S. Müller, eds.), (Darmstadt, Germany), Springer Verlag, (June 1994). Proceedings of Fifth Eurographics Workshop on Rendering.

12. D. Forsyth, C. Yang, and K. Teo, "Efficient radiosity in dynamic environments", in *Photorealistic Rendering Techniques* (G. Sakas, P. Shirley, and S. Müller, eds.), (Darmstadt, Germany), Springer Verlag, (June 1994). Proceedings of Fifth Eurographics Workshop on Rendering.
13. E. Shaw, "Hierarchical radiosity for dynamic environments", Master's thesis, Cornell University, Ithaca, New York, (August 1994).
14. P. S. Heckbert, "Adaptive radiosity textures for bidirectional ray tracing", *Computer Graphics*, **24**(4), pp. 145–154 (1990). Proceedings SIGGRAPH '90 in Dallas (USA).
15. D. Lischinski, F. Tampieri, and D. P. Greenberg, "Discontinuity meshing for accurate radiosity", *IEEE Computer Graphics and Applications*, **12**(6), pp. 25–39 (1992).
16. S. J. Teller, "Computing the antipenumbra of an area light source", *Computer Graphics*, **26**(4), pp. 139–148 (1992). Proceedings of SIGGRAPH '92 in Chicago (USA).
17. Z. Gigus and J. Malik, "Computing the aspect graph for the line drawings of polyhedral objects", *IEEE Trans. on Pat. Matching & Mach. Intelligence*, **12**(2), (1990).
18. Y. Chrysanthou and M. Slater, "Shadow volume BSP trees for computation of shadows in dynamic scenes", in *1995 Symposium on Interactive 3D Graphics* (P. Hanrahan and J. Winget, eds.), pp. 45–50, ACM SIGGRAPH, (Apr. 1995).
19. A. Worrall, C. Willis, and D. Paddon, "Dynamic discontinuities for radiosity", in *Edugraphics + Compugraphics Proceedings* (H. P. Santo, ed.), (Alvor, Portugal '95), pp. 367–375, GRASP- Graphic Science Promotions & Publications, P.O. Box 4076, Massama, 2745 Queluz, Portugal, (Dec. 12 1995).
20. J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing", in *Eurographics '87*, pp. 3–10, North-Holland, (Aug. 1987).
21. F. Durand, G. Drettakis, and C. Puech, "The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool", in *SIGGRAPH 97 Conference Proceedings (Los Angeles, CA)* (T. Whitted, ed.), Annual Conference Series, ACM SIGGRAPH, (Aug. 1997).
22. F. Durand, G. Drettakis, and C. Puech, "The 3d visibility complex, a new approach to the problems of accura te visibility", in *Rendering Techniques '96* (X. Pueyo and P. Shroeder, eds.), pp. 245–257, Springer Verlag, (June 1996). Proceedings of 7th Eurographics Workshop on Rendering in Porto, Portugal.



(a) (b)

Figure 14: *(a) Discontinuity surfaces in a voxel, (b) Intersection of motion volume with scene polygons*



(a) (b)

(c)

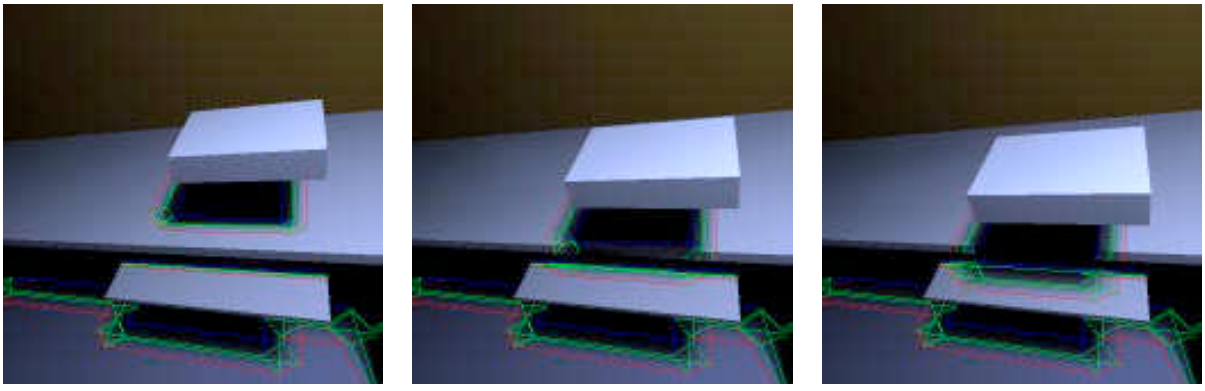
Figure 15: *(a) Two EV surfaces which do not intersect (b) Intersection of the EV surfaces (c) One of the two EEE surfaces created*

(a) (b) (c)

Figure 16: *(a) Scene 1 (b) Scene 2 (c) Scene 3*

(a) (b) (c) (d)

Figure 17: *Test 1: (a) position 1 (b) position 2 (c) position 3 (d) position 4*



(a) (b) (c)

Figure 18: *Test 2: (a) position 1 (b) position 3 (c) position 4*