# The Visibility Skeleton:
# A Powerful And Efficient Multi-Purpose Global Visibility Tool

Frédo Durand, George Drettakis and Claude Puech

iMAGIS[†]- GRAVIR/IMAG - INRIA

## Abstract

Many problems in computer graphics and computer vision require accurate *global* visibility information. Previous approaches have typically been complicated to implement and numerically unstable, and often too expensive in storage or computation. The Visibility Skeleton is a new powerful utility which can efficiently and accurately answer visibility queries for the entire scene. The Visibility Skeleton is a *multi-purpose* tool, which can solve numerous different problems. A simple construction algorithm is presented which only requires the use of well known computer graphics algorithmic components such as ray-casting and line/plane intersections. We provide an exhaustive catalogue of visual events which completely encode all possible visibility changes of a polygonal scene into a graph structure. The nodes of the graph are extremal stabbing lines, and the arcs are critical line swaths. Our implementation demonstrates the construction of the Visibility Skeleton for scenes of over a thousand polygons. We also show its use to compute exact visible boundaries of a vertex with respect to any polygon in the scene, the computation of global or on-the-fly discontinuity meshes by considering any scene polygon as a source, as well as the extraction of the exact blocker list between any polygon pair. The algorithm is shown to be manageable for the scenes tested, both in storage and in computation time. To address the potential complexity problems for large scenes, on-demand or lazy contruction is presented, its implementation showing encouraging first results.

**Keywords:** Visibility, Global Visibility, Extremal Stabbing Lines, Aspect Graph, Global Illumination, Form Factor Calculation, Discontinuity Meshing, View Calculation.

## 1 INTRODUCTION

Ever since the early days of computer graphics, the problems of determining visibility have been central to most computations required to generate synthetic images. Initially the problems addressed concerned the determination of visibility of a scene with respect to a given point of view. With the advent of interactive walkthrough systems and lighting calculations, the need for *global* visibility queries has become much more common. Many examples of such requirements exist, and are not limited to the domain of computer graphics. When walking through a complex building, real-time visualization algorithms require the information of which objects are visible to limit the number of primitives rendered, and thus achieve better frame rates. In global illumination computations, the dominant part of any calculation concerns the determination of the proportion of light leaving surface $s$ and arriving at surface $r$. This determination depends heavily on the relative occlusion of the two objects, requiring the calculation of which parts of $s$ are visible from $r$. All such applications need detailed data structures which completely encode global visibility information; previous approaches have fallen short of this goal.

### 1.1 Motivation

The goal of the research presented here is to show that it is possible to construct a data structure encompassing all global visibility information and to show that our new structure is useful for a number of different applications. We expect the structure we present to be of capital importance for any application which requires detailed visibility information: the calculation and maintenance of the view around a point in a scene, the calculation of exact form-factors between vertices and surfaces, the computation of discontinuity meshes between any two pairs of objects in a scene as well as applications in other domains such as aspect graph calculations for computer vision etc.

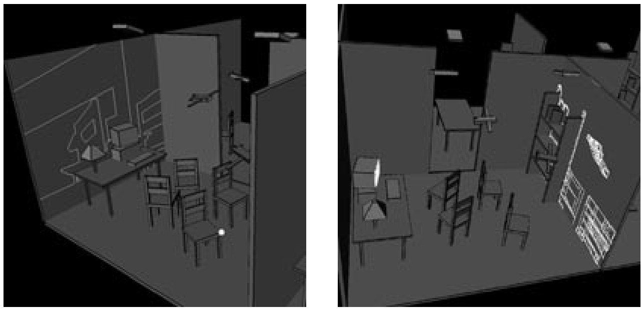
Previous algorithms have been unable to provide efficient and robust data structures which can answer global visibility queries for typical graphics scenes. In what follows we present a new data structure which can provide *exact* global visibility information. Our structure, called the *Visibility Skeleton*, is easy to build, since its construction is based exclusively on standard computer graphics algorithms, i.e., ray casting and line-plane intersections. It is a *multi-purpose* tool, since it can be used to solve numerous different problems which require global visibility information; and finally it is well-adapted to *on-demand* or *lazy* construction, due to the locality of the construction algorithm and the data structure itself. This is particularly important in the case of complex geometries.

The central component of the Visibility Skeleton are *critical lines* and *extremal stabbing lines*, which, as will be explained in detail in what follows, are the foci of all visibility changes in a scene. All modifications of visibility in a polygonal scene can be described by these critical lines, and a set of *line swaths* which are necessarily adjacent to these lines. In this paper we present the construction of the Skeleton, and the implementations of several applications. As an example, consider Fig. 1(a), which is a scene of 1500 polygons. After the construction of the skeleton, many different queries can be answered efficiently. We show the view from the green selected point to the left wall which only required 1.4 ms to compute; in Fig. 1(b), the complete discontinuity mesh on the right wall is generated by considering the screen of the computer as an emitter which required 8.1 ms.

After a brief overview of previous work (Section 1.2), we will provide a complete description of all possible nodes, and all the adjacent line swaths in Section 2. In Section 3 the construction algorithm and the actual data structure are described in detail. The results of our implementation are then presented in Section 4, giving the complete construction of the Visibility Skeleton for a suite of test

---

[†] iMAGIS is a joint research project of CNRS/INRIA/UJF/INPG. iMAGIS/GRAVIR, BP 53, F-38041 Grenoble Cedex 09 France. E-mail: `{Frederic.Durand|George.Drettakis|Claude.Puech}@imag.fr` `http://www-imagis.imag.fr/`



(a) (b)

Figure 1: (a) Exact computation of the part of the left wall as seen by the green vertex. (b) Complete discontinuity mesh on the right wall when considering the computer screen as source.

scenes. We show how the Skeleton is then used to provide exact point-to-surface visibility information for any vertex in the scene, to calculate the complete discontinuity mesh between any two surfaces in the scene, extract exact blocker lists between two objects, and compute all visibility interactions of one object with all other objects in a scene, which could be used for dynamic illumination updates in scenes with moving objects. Section 5 addresses the issues arising when treating more complex scenes, and in particular we present a first attempt at on-demand construction. The results of the implementation show that this allows significant speedup compared to the complete algorithm. In Section 6 we sketch how the structure can be extended to environments in which objects move, as well as other potential extensions, and we conclude.

### 1.2 Previous Work

Many researchers in computer graphics, computational geometry and computer vision have addressed the issue of calculating global visibility. We present here a quick overview of closely related previous work, which is of course far from exhaustive.

Interest in visibility structures in computer graphics was expressed by Teller [26], when presenting an algorithm for the calculation of anti-penumbra. This work was in part inspired by the wealth of research in computer vision related to the *aspect graph* (e.g., [21, 10, 9]). The work of Teller is closely related to the development of discontinuity meshing algorithms (pioneered by [14, 17]). These algorithms lead to structures closely resembling the aspect graph which contain visibility information (*backprojections*) with respect to a light source [5, 24]. Discontinuity meshes have been used in computer graphics to calculate visibility and improve meshing for global illumination calculations [18, 6]. Nonetheless, these structures have always been severely limited by their inability to treat visibility between objects other than the primary light sources. This is caused by the fact that the calculation of the discontinuity mesh with respect to a source is expensive and prone to numerical robustness problems.

An alternative approach to calculating visibility between two patches for global illumination has been proposed by Teller and Hanrahan [27]. In this work a conservative algorithm is presented which answers queries concerning visibility between any two patches in the scene but does not provide exact visibility information. In addition, this approach provides tight blocker lists of potential occluders between a patch pair. Information on the potential occluders between a patch pair is central in the design of any refinement strategy for hierarchical radiosity [12]. The ability to determine analytic visibility information between two arbitrary patches would render practical the error bound refinement strategy of [16], which requires this information.

In computational geometry, the problem of visibility has been extensively studied in two dimensions. The visibility complex [22] provides all the information necessary to compute global visibility. This was successfully used in a 2D study of the problem applied to radiosity [19]. A similar structure in 3D, called the *asp*, has been presented in computer vision by Plantinga and Dyer [21], to allow the computation of aspect graphs. This structure provides the information necessary to compute exact visibility information. A related, but more efficient structure called the 3D visibility complex [7] has been proposed. Both structures have remained at the theoretical level for the full 3D perspective case which is the only case of interest for 3D computer graphics, despite partial implementations of orthographic and other limited cases for the *asp* [21]. Other related work in a computational geometry framework can be found in [15, 20].

Moreover, most of the work done on static visibility does not easily extend to dynamic environments. Most of the time, motion volumes enclosing all the positions of the moving objects are built [3, 8, 23].

## 2 THE VISIBILITY SKELETON

The new structure we will present addresses many of the shortcomings of previous work in global visibility. As mentioned earlier, the emphasis is on the development of a multi-purpose tool which can be easily used to resolve many different visibility problems, a structure which is easy and stable to build and which lends itself to on-demand construction and dynamic updates.

In what follows, we will consider only the case of polygonal scenes.

### 2.1 Visual Events

In previous global visibility algorithms, in particular those relating to aspect graph computations (e.g., [21, 10, 9]), and to antipenumbra [26] or discontinuity meshing [5, 24], visibility changes have been characterized by *critical lines sets* or *line swaths* and by *extremal stabbing lines.*

Following [20] and [26], we define an extremal stabbing line to be incident on four polygon edges. There are several types of extremal stabbing lines, including vertex-vertex (or $VV$) lines, vertex-edge-edge (or $VEE$) lines, and quadruple edge (or $E4$) lines. As explained in Section 2.3.1, we will also consider here extremal lines associated to faces of polyhedral objects.

A *swath* is the surface swept by extremal stabbing lines when they are moved after relaxing exactly one of the four edge constraints defining the line. The swath can either be planar (if the line remains tight on a vertex) or a regulus, whose three generator lines embed three polygon edges.

We call *generator elements* the vertices and edges participating in the definition of an extremal stabbing line.

We start with an example: after traversing an $EV$ line swath from left to right as shown in Figure 2(a), the vertex as seen from the observer will lie upon the polygon adjacent to the edge and no longer upon the floor. This is a visibility change (often called visibility event). The topology of the view is modified whenever the vertex and the edge are aligned, that is, when there is a line from the eye going through both $e$ and $v$.

This $EV$ line swath is a one dimensional ($1D$) set of lines, passing through the vertex $v$ and the edge $e_1$, thus it has one degree of freedom (varying for example over the edge $e$). When two such $EV$ surfaces meet as in Figure 2(b) a unique line is defined by the intersection of the two planes defined by the $EV$ surfaces. This line is an extremal stabbing line; it has zero degrees of freedom.

In what follows we will develop the concepts necessary to avoid any direct treatment of the line swaths themselves since sets of lines

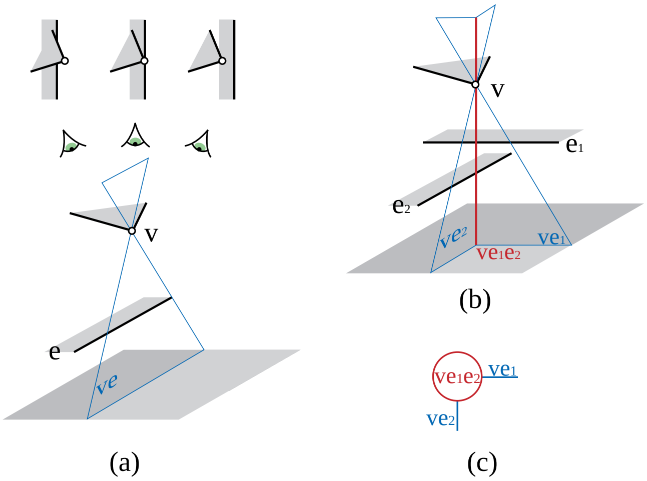


Figure 2: (a) While the eye traverses the line swath $VE$, the vertex $v$ passes over the edge $e$. (b) Two line swaths meet at an extremal stabbing line (c) and induce a graph structure

or the surfaces described by these sets are difficult to handle, in part because they can be ruled quadrics. All computations will be performed by line – or ray – casting in the scene.

We will be using the extremal stabbing lines to encode all visibility information, by storing a list of all line swaths adjacent to each extremal stabbing line. In our first example of Figure 2(b), the $VEE$ line $ve_1e_2$ is adjacent to the two $1D$ elements $ve_1$ and $ve_2$ described above; i.e., the swaths $ve_1$ and $ve_2$. Additional adjacencies for the $VEE$ line $ve_1e_2$ are implied by the interaction of $ve_2$ and $e_1$ (Fig 3(a)).

To complete the adjacencies of a $VEE$ line, we need to consider the $EEE$ line swaths related to the edges $e_4$ and $e_2$, and the two edges $e_4$ and $e_3$ which are adjacent to the vertex $v$ (Fig. 3(b) and (c)).

The simple construction shown above introduces the fundamental idea of the Visibility Skeleton: by determining all the appropriate extremal stabbing lines in the scene, and by attaching all adjacent line swaths, we can completely describe all possible visibility relationships in a 3D scene. They will be encoded in a graph structure as shown on Fig.3, to be explained in Section 2.3.2. Consider the example shown in Fig.3(a): The node associated to extremal stabbing line $ve_1e_2$ is adjacent in the graph structure to the arcs associated with line swaths $ve_1$, $ve_1{}'$ and $ve_2$.

## 2.2 The 3D Visibility Complex, the Asp and the Visibility Skeleton

The *Visibility Complex* [7], is a structure which also contains all relevant visibility information for a 3 dimensional scene. It is also based on the adjacencies between visibility events and considers sets of maximal free segments of the scene (these are lines limited by intersections with objects).

The zero and one-dimensional components of the visibility complex are in effect the same as those introduced above, which we will be using for the construction of the Visibility Skeleton. Similar constructions were presented (but not implemented to our knowledge for the complete perspective case) for the *asp* structure [21] for aspect graph construction.

In both cases, higher dimensional line sets are built. For the visibility complex in particular, faces of 2, 3 and 4 dimensions are considered. For example, the set of lines tangent to two objects has 2 degrees of freedom, those tangent to one object 3 degrees of freedom, etc. (see [7] for details).

These sets and their adjacencies could theoretically be useful for some specific queries such as view computation or dynamic updates, for example in some specific worst cases such as scenes composed of grids aligned and slightly rotated. In such cases, almost all objects occlude each other and the high number of line swaths and extremal stabbing lines makes the grouping of lines into higher dimensional sets worthwhile.

The *Visibility Complex* and *asp* are intricate data-structures with complicated construction algorithms since they require the construction of a $4D$ subdivision. In addition they are difficult to traverse due to the multiple levels of adjacencies. Our approach is different: we have developed a data structure which is easy to implement and easy to use.

These facts also explain the name *Visibility Skeleton*, since our new structure can be thought of as the skeleton of the complete Visibility Complex.

## 2.3 Catalogue of Visual Events and their Adjacencies

The Visibility Skeleton is a graph structure. The nodes of the graph are the extremal stabbing lines and the arcs correspond to line swaths. In this section (and in Appendix 7.1) we present an exhaustive list of all possible types of arcs and nodes of the Visibility Skeleton.

### 2.3.1 1D Elements: Arcs of the Visibility Skeleton

In Figure 4, we see the four possible types of $1D$ elements: an $EV$ line swath (shown in blue), an $EEE$ line swath (shown in purple) and two line swaths relating a polygonal face ($F$) to one of its vertices ($Fv$) or an edge of another polygon($FE$) (both are shown in blue). In the upper part of the figure we show the view (with changes in visibility), as seen from a viewpoint located above the scene and, from left to right in front of, on, or behind the line swath.

Note that the interaction of an edge $e$ and a vertex $v$ can correspond to many $ve$ arcs of the skeleton. These arcs are separated by nodes. Consider, for example, arcs $ve_1$ and $ve_1{}'$ adjacent to node $ve_1e_2$ in Fig. 3(a).

### 2.3.2 0D Elements: Nodes of the Visibility Skeleton

As explained in Section 2.1, two line swaths which meet define an extremal stabbing line, which in the Visibility Skeleton is the node at which the arcs meet. This section presents a list of the configurations creating nodes and their corresponding adjacencies. A figure is given in each case.

The simplest node corresponds to the interaction of two vertices shown in Figure 5(a).

The interaction of a vertex $v$ and two edges $e_1$ and $e_2$ can result in two configurations, depending on the relative position of the vertex with respect to the edges. The first node was presented previously in Figure 3 and the second is shown in Figure 5(b).

The interaction of four edges is presented in Figure 6, together with the six corresponding adjacent $EEE$ arcs. Face related nodes are given in detail in the appendix: $EFE$, $FEE$, $FF$, $E$ and $Fvv$ (see Fig. 18 to 19).

# 3 DATA STRUCTURE AND CONSTRUCTION ALGORITHM

Given the catalogues of nodes and arcs presented in the previous section, we can present the details of a suitable data structure to rep-

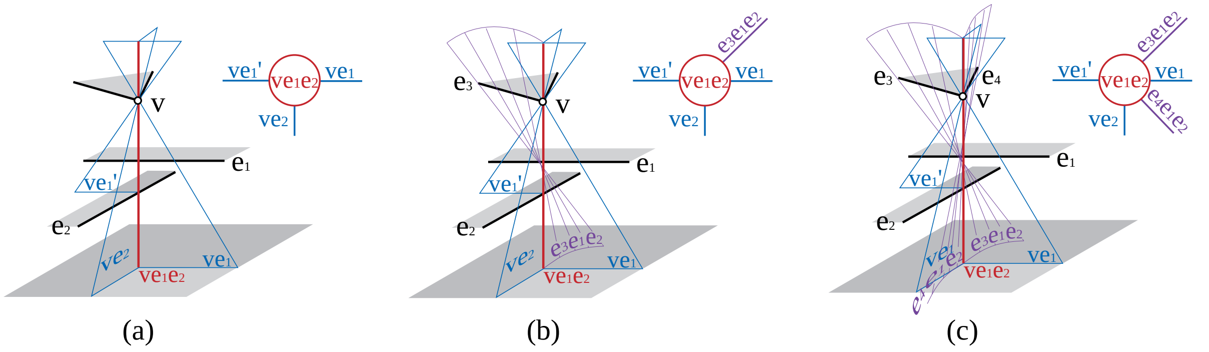


Figure 3: (a) An additional $EV$ line swath is adjacent to the extremal stabbing line, (b) (c) and two $EEE$ line swaths

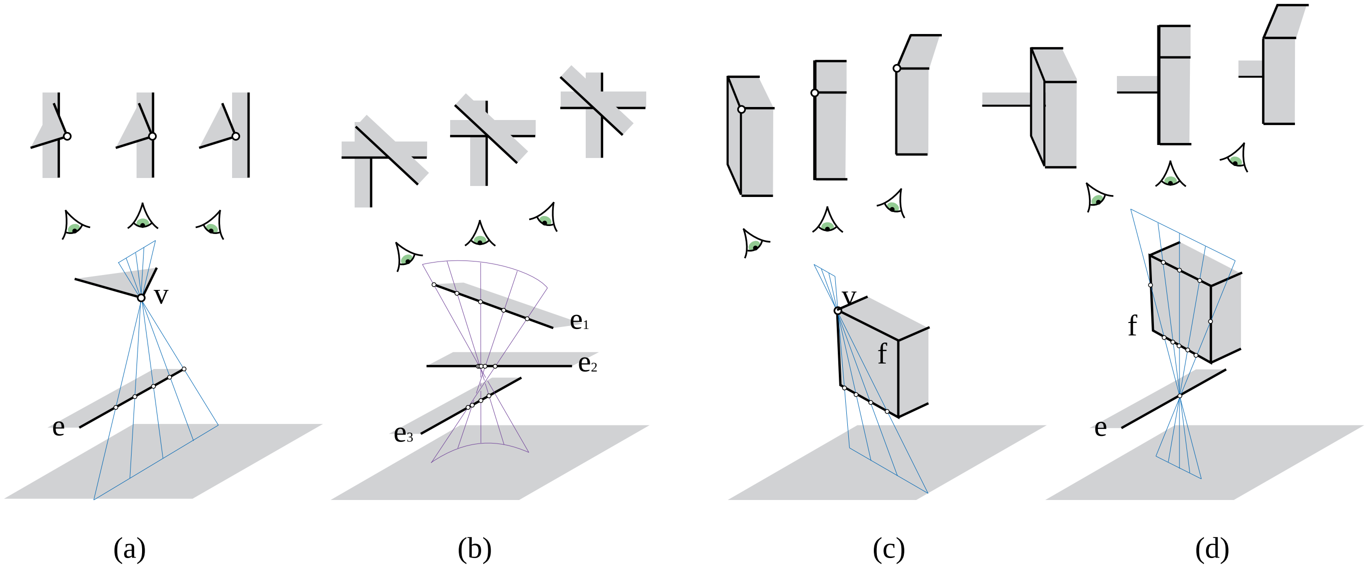


Figure 4: (a) Same as Fig. 1(a). (b) In front of the $EEE$ line swath the edge $e_2$ is visible, on the swath the edges meet at a point and behind $e_2$ is hidden. (c) In front of the $FV$ we see the front side of $F$, on the swath we see a line and behind we see the other side of $F$. (d) The $FE$ swath is similar to the $FV$ case.

resent the Visibility Skeleton graph structure, as well as the algorithm to construct it.

**Preliminaries:** Our scene model provides the adjacencies between vertices, edges and faces. Before processing the scene, we traverse all vertices, edges and faces, and assign a unique number to each. This allows us to index these elements easily. In addition, we consider all edges to be uniquely oriented. This operation is arbitrary (i.e., the orientation does not depend on the normal of one of the two faces attached to the edge), and facilitates consistency in the calculations we will be performing.

## 3.1 Data Structure

The simplest element of the structure is the node. The $Node$ structure contains a list of arcs, and pointers to the polygonal faces $F_{up}$ and $F_{down}$ (possibly void) which block the corresponding extremal stabbing line at its endpoints $P_{up}$ and $P_{down}$.

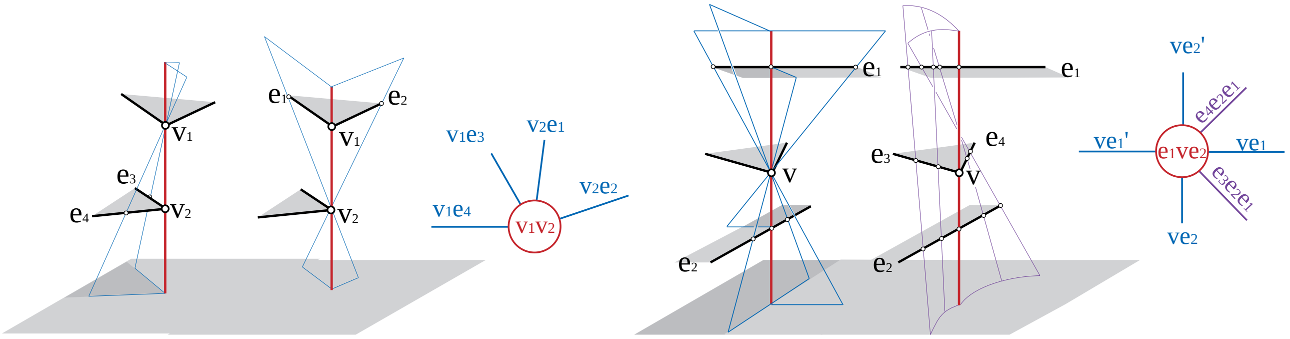
The structure for an $Arc$ is visualized in the Fig.7(a). The arc represented here (swath shown in blue) is an $EV$ line set. There are two adjacent nodes $N_{start}$, $N_{end}$, represented as red lines. All the adjacency information is stored with the arc. Details of the structures $Node$ and $Arc$ are given in Fig. 7(b).

To access the arc and node information, we maintain arrays of balanced binary search trees corresponding to the different type of swaths considered. For example, we maintain an array $ev$ of trees of $EV$ arcs (see Fig.7(b)). These arrays are indexed by the unique identifiers of the endpoints of the arcs. These can be faces, vertices or edges (if the swath is interior, that is if the lines traverse the polyhedron).

This array structure allows us to efficiently query the arc information when inserting new nodes and when performing visibility queries. The balanced binary search tree used to implement the query structure is ordered by the identifiers of the generators and by the value of $t_{start}$.

## 3.2 Finding Nodes

Before presenting the actual construction of each type of node, we briefly discuss the issue of “local visibility”. As has been presented in other work (e.g., [10]), for any edge adjacent to two faces of a polyhedron, the negative half-space of a polygonal face is locally invisible. Thus when considering interactions of an edge $e$, we do not need to process any other edge $e'$ which is “behind” the faces adjacent to $e$. This results in the culling of a large number of potential events.



(a) (b)

Figure 5: (a) A $VV$ node $v_1v_2$ is adjacent to four $EV$ arcs defined by a vertex and an edges of the other vertex: $v_1e_3$ and $v_1e_4$ and $e_1v_2$ and $e_2v_2$. (b) An $EVE$ node $e_1ve_2$: each edge defines two $EV$ arcs with $v$ depending on the polygon at the extremity, and to two $EEE$ defined by $e_1$, $e_2$ and the two edges adjacent to $v$.

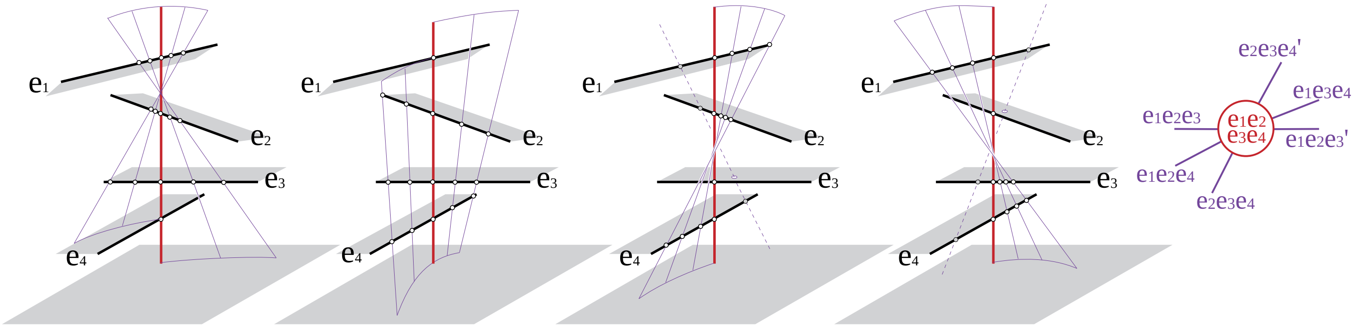
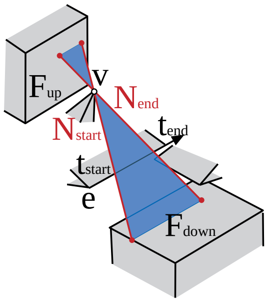


Figure 6: An $E4$ node is adjacent to six $EEE$ arcs.



```
class Node {
    List<Arcs> adjacentArcs
    Face $F_{up}$, $F_{down}$
    Point3D $P_{up}$, $P_{down}$
}

class Arc {
    Node $N_{start}$, $N_{end}$
    float $t_{start}$, $t_{end}$
    Face $F_{up}$, $F_{down}$
}
class EV : child of Arc {
    Edge $e$
    Vertex $v$
    }
class VisibilitySkeleton {
    tree<EV> ev[$F_{up}$][$F_{down}$]
    tree<EEE> eee[$F_{up}$][$F_{down}$]
    ...
}
```

(a) (b)

Figure 7: Basic Visibility Skeleton Structure.

### 3.2.1 Trivial Nodes

The simplest nodes are the $VV$, $Fvv$ and $Fe$ nodes. For these, we simply loop over the appropriate scene elements (vertices, edges and faces). The appropriate lines are then intersected with the scene using a traditional ray-caster to determine if there is an occluding object between the related scene elements, in which case no extremal stabbing line is reported. Otherwise it gives the elements and points at the extremities of the lines, and thus the appropriate location in the overall arc tree array.

### 3.2.2 VEE and EEEE Nodes

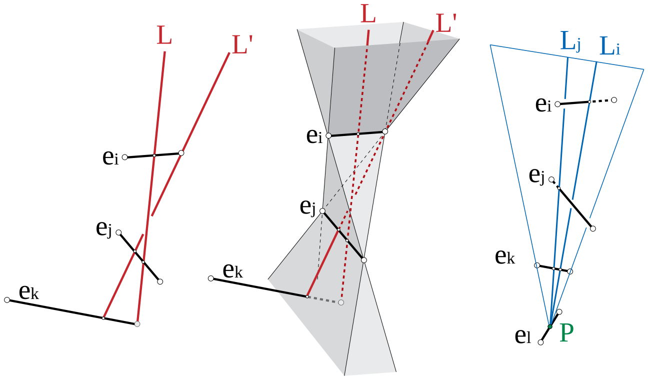
We consider two edges of the scene $e_i$ and $e_j$. All the lines going through two segments are within an extended tetrahedron (or double wedge) shown in Fig. 8, defined by four planes. Each one of these planes is defined by one of the edges and an endpoint of the other.

To determine the vertices of the scene which can potentially generate a $VEE$ or $EVE$ stabbing line, we need only consider vertices within the wedge. If a vertex of the scene is inside the double wedge, there is a potential $VEE$ or $EVE$ event.

We next consider a third edge $e_k$ of the scene. If $e_k$ cuts a plane of the wedge, a $VEE$ or $EVE$ node is created. If edge $e_k$ of the scene intersects the plane of the double wedge defined by edge $e_i$ and vertex $v$ of $e_j$, there is a $ve_ie_k$ or $e_ive_k$ event (Fig. 8(a)).

We next proceed to the definition of the $E4$ nodes. The intersections of $e_k$ and the planes of the double wedge *restrict* the third edge $e_k$. To compute a line going through $e_i$, $e_j$, $e_k$ we need only consider the restriction of $e_k$ to the double wedge defined by $e_i$ and $e_j$. This process is re-applied to restrict a fourth edge $e_l$ by the wedge of $e_i$ and $e_j$, by that of $e_i$ and $e_k$ and by that of $e_j$ and $e_k$. This multiple restriction process eliminates a large number of candidates.

Once the restriction is completed, we have two $EEE$ line sets, those passing through $e_l$, $e_i$ and $e_j$ and those passing through $e_l$, $e_i$ and $e_k$. A simple binary search is applied to find the point on $e_l$ (if it exists) which defines the $E4$ node. We perform this search for a point $P$ of $e_l$ by searching for the root of the angle formed by the two lines defined by the intersection of the plane $(P, e_i)$ with $e_j$ and with $e_k$. This is shown on Fig.8(c).



(a) (b) (c)

Figure 8: (a) (b) VEE enumeration and $EEE$ restriction. (c) $E4$ computation: find the root of the angle of the lines going through $e_je_ke_l$ and that through $e_le_ke_i$.

A more robust algorithm such as the one given in [28] could be used, but the simpler algorithm presented here seems to perform well in practice. This is true mainly because we are not searching for infinite stabbing lines, but for restricted edge line segments. The potential $VEE$ and $E4$ enumeration algorithm is given in Fig. 9.

We have developed an acceleration scheme to avoid the enumeration of all the triples of edges. For each pair of edges, we reject very quickly most of the third potential edges using a regular grid. Instead of checking if each cell of the grid intersects the extended tetrahedron, we use the projection on the three axis-aligned planes. For each such plane, we project the extended tetrahedron (which gives us an hourglass shape), and we perform the actual edge-tetrahedron intersection only for the edges contained in the cells whose three projections intersect the three pixelized hourglasses.

### 3.2.3 Non-Trivial Face Nodes

To calculate the non-trivial face-related nodes, we start by intersecting the plane of each face $f_1$ with every edge of the scene. For edges intersecting the face we attempt to create an $FvE$ node (Fig.18).

For each pair of intersections, we search for a $FEE$ node. To do this we determine if the line joining the two intersections intersects the face $f_1$. The last operation required is the verification of the existence of an $FF$ node. This case occurs if the faces adjacent to the edge of the intersection cause an $FF$. The construction for the $FEE$ and $FF$ nodes is described in Fig. 10 (a).

## 3.3 Creating the Arcs

The creation of the arcs of the Visibility Skeleton is performed simultaneously with the detection of the nodes. When inserting a new node, we create all the adjacent arcs from the corresponding catalogue presented in Section 2.3.2. For each of these arcs $a$ we calculate the arc parameter $t$ corresponding to the node to be inserted, and proceed as explained in Fig.12. We then access the list of arcs in the Skeleton with the same extremities (thus in the same list of the array) and which have the same generator elements (*vertices and edges*) as the arc $a$. If the value of $t$ indicates that the node is contained in the arc, we determine whether this node is the start of the end node of the arc. This is explained in more detail in the following paragraph. If this position is already occupied we split the arc, else we assign the node the corresponding extremity of the arc. This process is summarized in Fig. 11.

We have seen above that each time an arc adjacent to a node is considered, we have to know if it is its *start node* or its *end node*. In some cases this operation is trivial, for example for a $v_1v_2$ node and one if its adjacent $v_1e$ arcs, we simply determine if $v_2$ is the starting vertex of $e$. In other cases, this can be more involved, especially for the $E4$ case. This case and the necessary criteria for the other cases are summarized in Table 2 in the Appendix.

In Fig. 12, we illustrate the construction algorithm. Initially a trivial $vv_e$ node is created. The second node identified is $vfe$, which is adjacent the arc $ve$. Thus the arc $ve$ is adjacent to both $vv_e$ and $vfe$. The third node to be created is $vee_3$. When this node is inserted, we realize that the start node for $ve$ already exists, and we thus split the $ve$ arc. This splitting operation will leave the end of the $ve$ arc connected to $vv_e$ undefined. The final insertion shown is $ve_2e$ which will fill an undefined node previously generated.

# 4 IMPLEMENTATION AND FIRST APPLICATIONS

We have completed a first implementation of the data structure described. We have run the system on a set of test scenes, with varying visibility properties. In its current form, we have successfully computed the Visibility Skeleton for scenes up to 1500 polygons.

In what follows we first present Visibility Skeleton construction statistics for the different test scenes used. We then proceed to demonstrate the flexible nature of our construction, by presenting the use of our data structure to efficiently answer several different global visibility queries.

## 4.1 Implementation and Construction Statistics

Our current implementation requires convex polyhedra as input. However, this is not a limitation of the approach since we use polyhedral adjacencies simply for convenience when performing local visibility tests.

We treat touching objects by detecting this occurrence and slightly modifying the ray-casting operation. We also reject coplanar edge triples. Other degeneracies such as intersecting edges are not yet treated by the current implementation.

We present statistics on the size of our structure and construction time in Table 1. Evidently, these tests can only be taken as an indication of the asymptotic behavior of our algorithm. As such, we see that our test suite indicates quadratic growth of the memory requirements and super-quadratic growth of the running time. In particular, for the test suite used, the running time increases with $n^{2.4}$ on average, where $n$ is the number of polygons.

The $VEE$ nodes are the most numerous. There are approximately a hundred times fewer $E4$ nodes, even though theoretically there should be an order of magnitude more.

We believe that the memory requirements could be greatly decreased by an improved implementation of the arrays of trees. Currently, a large percentage of the memory required is used by these arrays (e.g. for scene (d) of Table 1., the arrays need 53.7Mb out of a total 135Mb). Since these arrays are very sparse (e.g. 99.3% empty for scene (d)), it is clear that storage requirements can be greatly reduced.

In the case of densely occluded scenes, the memory requirements grow at a slower rate, on average much closer to linear than quadratic with respect to the number of polygons. As an example, we replicated scene (a) 2, 4 and 8 times, thus resulting in isolated rooms containing a single chair each. The memory requirements (excluding the quadratic cost of the arrays) are 1.2Mb, 2.8Mb, 8.6Mb and 17.3Mb, for respectively 78, 150, 300 and 600 polygons.

The theoretical upper bounds are very pessimistic, $O(n^4)$ in size because every edge quadruple can have two lines going through it

```
potential VEE and EEEE enumeration
{
foreach edge $e_i$ from 1 to $n$
    foreach edge $e_j$ from $i+1$ to $n$ locally visible
        foreach edge $e_k$ from $j+1$ to $n$ locally visible
            compute the EEE restrictions $e_i e_j e_k$
        foreach edge $e_k$ from $j+1$ to $n$ locally visible
            foreach segment of its restrictions
                foreach edge $e_l$ from $k+1$ to $n$ locally visible
                foreach segment of its restrictions
                    search for E4
}
```

(a)

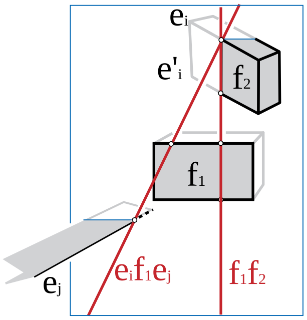
```
EEE restriction
{
foreach of the 4 planes
    compute the intersection $inter$ with the line of the edge
        if it $inter$ on the edge
            propose a VEE
            restrict the edge
foreach of the edge endpoints $s_{ep}$
    if $s_{ep}$ is inside the double wedge
        propose a VEE
        restrict the edge
}
```

(b)

Figure 9: Enumeration of Potential VEE and E4 Nodes.

```
Find Face Nodes {
foreach face $f_1$ of the scene
    foreach edge $e$ of the scene
        compute the intersection of the edge $e$ with the plane of $f_1$
        foreach intersection $P_i$
            create a $FvE$
            foreach intersection $P_j$
                if $(P_i P_j)$ intersects $f_1$
                    create $FEE$
            foreach of the 2 faces $f_2$ adjacent to the edge of Pi
                find $P_j$ the intersection of a second edge of $f_2$ with $f_1$
                if $(P_i P_j)$ intersects $f_1$
                    create $FF$
}
```

(a)



(b)

Figure 10: Finding Face Nodes.

[28], and $O(n^5)$ in time because such potential extremal stabbing lines have to be ray-cast with the whole scene. But such bounds occur only in uncommon worst case scenes such as grids aligned and rotated or infinite lines. It is clear that our construction algorithm would be very inefficient for such cases. More efficient construction algorithms are possible, but these approaches suffer from all the problems described previously in Section 2.2.

In what concerns the robustness of the computation, previous aspect graph and discontinuity meshing algorithms depend heavily on the construction of the arrangement (of the mesh or aspect graph "cells"), as the algorithm progresses. In the construction presented here, this is not the case since all operations are completely local. Since we perform ray-casting and line-plane intersections, the number of potential numerical problems is limited. Degeneracies can occasionally cause some problems, but due to the locality, this does not effect the construction of the Skeleton elsewhere. More efficient sweep-based algorithms are particularly sensitive to such instabilities, since an error in one position in space can render the rest of the construction completely incorrect and inconsistent.

## 4.2 Point-to-Area Form-Factor for Vertices

The calculation of point-to-area form factors has become central in many radiosity calculations. In most radiosity systems, point-to-area calculations are used to approximate area-to-area calculations [4, 2], and in others the actually point-to-area value is computed at the vertices [29].

In both theoretical [16] and experimental [6] studies, previous research has shown that error of the visibility calculation is a predominant source of inaccuracies. This is typically the case when ray casting is used. Lischinski et al. [16] have developed a very promising approach to bounding the error committed during light transfer for hierarchical radiosity. For it to be useful for general environments, access is required to the exact visibility information between a point on one element with respect to the polygon face it is linked to. This information is inherently global, since a pair of linked elements can contain *any* two surface elements of the scene.

The Visibility Skeleton in its initial form can answer this query exactly and efficiently for the original vertices of the input scene.

To calculate the view of a polygonal face from a vertex $v$, with respect to a face $f$, we first access all the $EV$ arcs of the skeleton related to the face $f$. This is simply the traversal of the line of our global two-dimensional array of arcs, indexed by $f$. For each entry of this list (many of which are empty), we search for the $EV$ arcs related to $v$. These $EV$ arcs are exactly the visible boundary of $f$ seen from $v$.

An example is shown in Fig. 13(a) and (b) For scene (b), containing 312 and 1488 polygons, the extraction of the point-to-area boundary takes respectively 1.2 ms and 1.5 ms (all query time are given without displaying the result).

## 4.3 Global and On-The-Fly Discontinuity Meshing

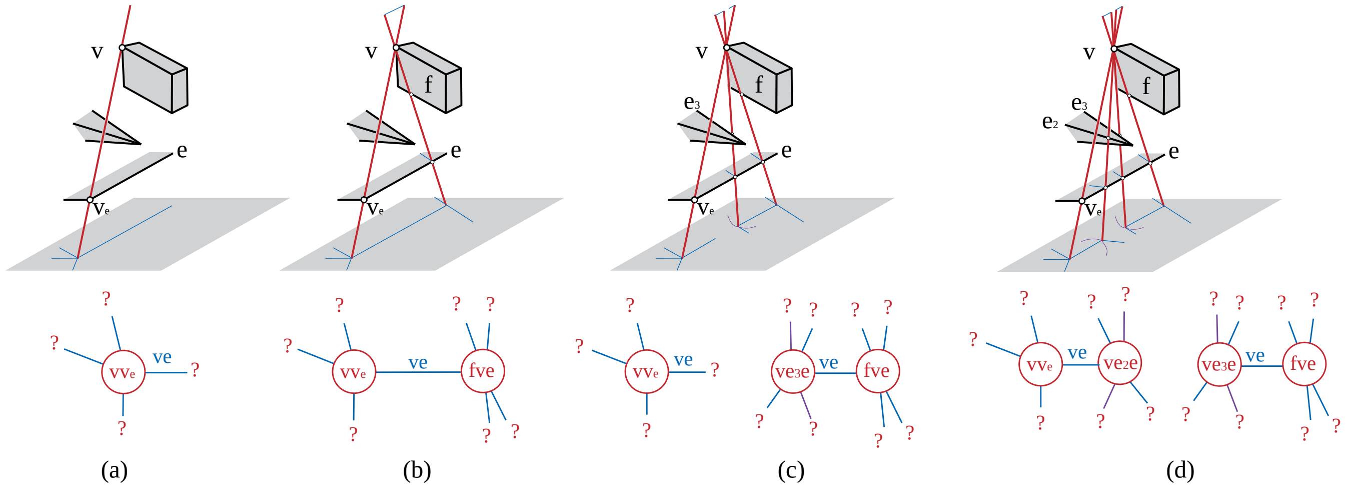
In radiosity calculations, it is often very beneficial to subdivide the mesh of a surface by following some [14, 18], or all [6] of the discontinuity surfaces between two surfaces which exchange energy. The partial [14, 17] or complete [5, 24] construction of such meshes has in the past been restricted to the discontinuity mesh between a source (which is typically a small polygon) and the receivers (which are the larger polygons of the scene). For all other interactions between surfaces of scenes, the algorithmic complexity and the inherent robustness problems related to the construction of these structures has not permitted their use [25].

For many secondary transfers in an environment, the construc-

```
Creation of a Visibility Skeleton Node
{
    foreach adjacent arc $n$
        compute $t$
        foreach arc $a$ with same extremities and same generators
            if $a \rightarrow t_{start} < t < a \rightarrow t_{end}$
            AddNodeToArc($n$, $a$)
        if no arc found
            create new Arc
}
```

```
AddNodeToArc(Node $n$, Arc $a$)
{
    $pos$ = decideStartOrEnd($n$, $a$)
    if $pos$ in $a$ undefined
        set $pos$ to $n$
    else
        split $a$ into two parts
}
```

Figure 11: Node Creation



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

Figure 12: Example of node insertions: (a) Insertion node $vv_e$. (b) Insertion of node $f_ve$. Arc $ve$ has now two ending nodes. (c) Insertion of node $ve_3e$. Arc $ve$ is split. (c) Insertion of node $ve_4e$, the two arcs $ve$ have their actual adjacent nodes.

tion of a global discontinuity mesh (i.e., from any surface (emitting/reflecting) to any other receiving surface in a scene), can aid in the accuracy of the global visibility computation. This was shown in the discontinuity driven subdivision used by Hardt and Teller [13]. In their case, the discontinuity surfaces are simply intersected with the scene polygons, and thus visibility on the line swath is not computed. With the Visibility Skeleton, the complete global discontinuity mesh between two surfaces can be efficiently computed.

To efficiently perform this query, we add an additional two-dimensional array $DM(i,\ j)$, storing all the arcs from face $f_i$ to $f_j$. Insertion into this array of lists and well as subsequent access is performed in constant time. To extract the discontinuity mesh between to surfaces $f_i$ and $f_j$ we simply access the entry $DM(i,\ j)$, and traverse the corresponding list. In Fig. 14(a), the complete discontinuity mesh between the source and the floor is extracted in 28.6 ms. The mesh caused by the small lamp on the table in Fig 14(b) was extracted in 1.3 ms (note that the arrangement is not built).

The resulting information is a set of arcs. These arcs can be used as in Hardt and Teller to guide subdivision, or to construct the arrangement of the discontinuity mesh on-the-fly, to be used as in [6] for the construction of a subdivision which follows the discontinuities. The adjacency information available in the Skeleton arcs and nodes should permit a robust construction of the mesh arrangement.

## 4.4 Exact Blocker Lists, Occlusion Detection and Efficient Initial Linking

When considering the interaction between two surfaces, it is often the case that we wish to have access to the exact list of blocker surfaces hiding one surface from the other. This is useful in the context of blocker list maintenance approaches such as that presented by Teller and Hanrahan [27].

The Visibility Skeleton can again answer this query exactly and efficiently. In particular, we use the global array $DM(i,j)$, and we traverse the related arcs. All the polygons related to the intervening arcs are blockers. It is important to note that this solution results in the *exact* blocker list, in contrast with all previous methods. Consider the example shown in Fig. 13(c) where we compute the occluders between the left ceiling lamp and the floor in 4 ms.

The *shaft* structure [11] would report all objects on the table though they are hidden by the table. In this case the Visibility Skeleton reports the exact set of blockers.

When constructing the Visibility Skeleton, we compute all the mutually visible objects of the scene: if two object see each other, there will be at least one extremal stabbing line which touches them or their edges and vertices. This is fundamental for hierarchical radiosity algorithms since it avoids the consideration of the interaction of mutually visible objects in the initial linking stage.

Similarly, the Skeleton allows for the detection of the occlusions caused by an object. This can be very useful for the case of a moving object $m$ allowing the detection of the form factors to be recomputed. To detect if the form factor $F_{ij}$ has to be recomputed we perform a query similar to the discontinuity mesh between two polygons: we traverse $DM(i,j)$ and search for an arc caused by an element (vertex, edge or face) of $m$. This gives us the limits of occlusions of $m$ between $f_i$ and $f_j$. Moreover, by considering all the arcs of the skeleton, we report all the form factors to be recomputed, and not a superset. Fig 14(c) shows the occlusions caused by the body of

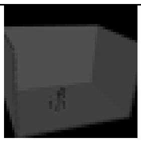
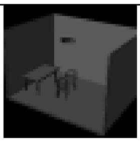
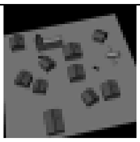
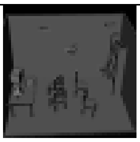
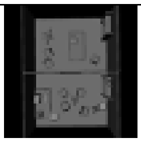
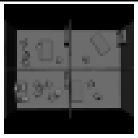
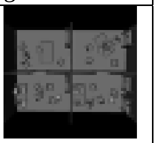
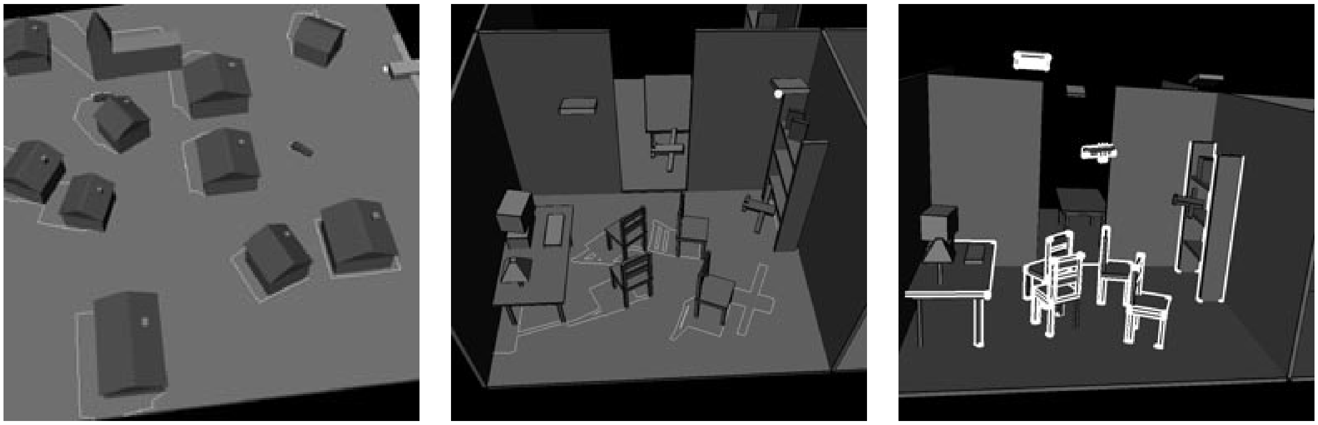
| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| Scene |  |  |  |  |  |  |  |
| Polygons | 84 | 168 | 312 | 432 | 756 | 1056 | 1488 |
| Nodes ($*10^3$) | 7 | 37 | 69 | 199 | 445 | 753 | 1266 |
| Arcs ($*10^3$) | 16 | 91 | 165 | 476 | 1074 | 1836 | 3087 |
| Construction | 1 s 71 s | 12 s 74 | 37 s 07 | 1 min 39 s | 5 min 36 s | 14 min 36 s | 31 min 59 |
| Memory (Mb) | 1.8 | 9 | 21 | 55 | 135 | 242 | 416 |

Table 1: Construction statistics (all times on a 195Mhz R10000 SGI Onyx 2). Storage is scene dependent and can be greatly reduced.



(a) (b) (c)

Figure 13: (a) Part of the scene visible from a vertex of the airplane. (b) Part of the floor seen by a vertex of the right-hand light source. (c) List of occluding blockers between the left light source and the floor. Note that the objects on the table that are invisible from the floor are not reported as blockers.

the plane between the screen and the right wall. This computation required 1.3 ms.

## 5 DEALING WITH SPATIAL COMPLEXITY: ON-DEMAND CONSTRUCTION

We propose here an on-demand or *lazy* scheme to compute visibility information only where and when needed. For example, if we want the discontinuity mesh between two surfaces, we just need to compute the arcs of the complex related to these two faces, and for this we only need to detect the nodes between these two faces.

The key for this approach is the locality of the Visibility Skeleton construction algorithm. We only compute the nodes of the complex where needed. The fact that some arcs might have missing nodes causes no problem since no queries will be made on them. Later on, other queries can appropriately link the missing nodes with those arcs.

Two problems must be solved: determination of what is to be computed, and determination of what has already been computed.

We propose two approaches: a source driven computation, and an adaptive subdivision of ray-space in the spirit of [1].

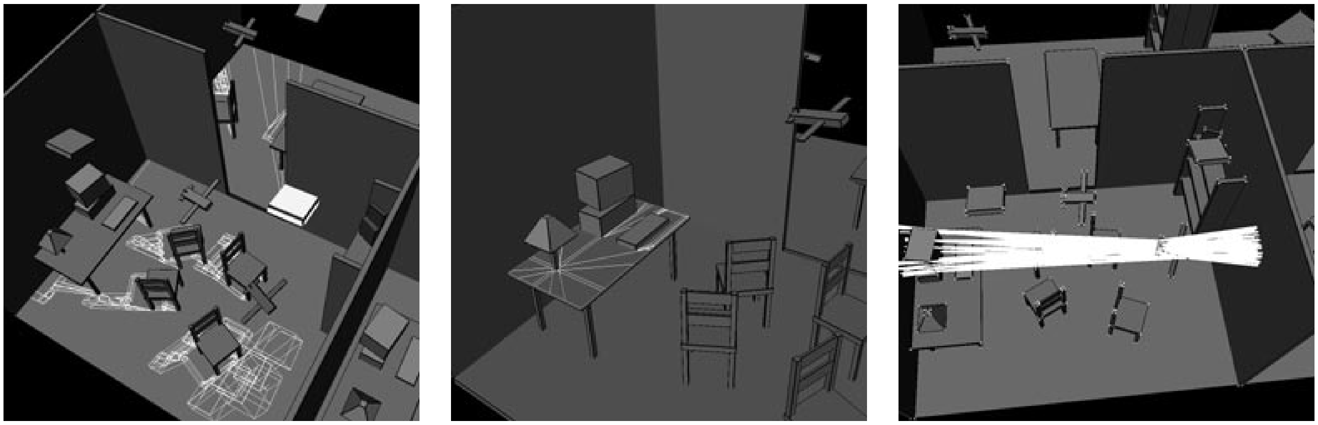
In the context of global illumination, the information related to “sources” (emitters or reflectors) is crucial. Thus the part of the visibility skeleton we compute in an on-demand construction is related to lines cutting the sources. The event detection has to be modified: every time a double wedge or a face does not cut the source, the pair of edges or the face is discarded, and if a potential node is detected, the ray-casting is performed only if the corresponding critical line cuts the source.

We use our grid-acceleration scheme here too: for each first edge, an edge pair is formed only for the edges that lie inside the hourglass defined by the source and the first edge.

When considering many sources one after the other, we also have to detect nodes already computed. If the sources are small, it is not worth rejecting double wedges, and only the final ray-casting and node insertion can be avoided (in our implementation they account for a third of the running time). We can perform a “final computation” if we want all the nodes that have not yet been computed: we just test before ray-casting if the critical line cuts one of the sources.

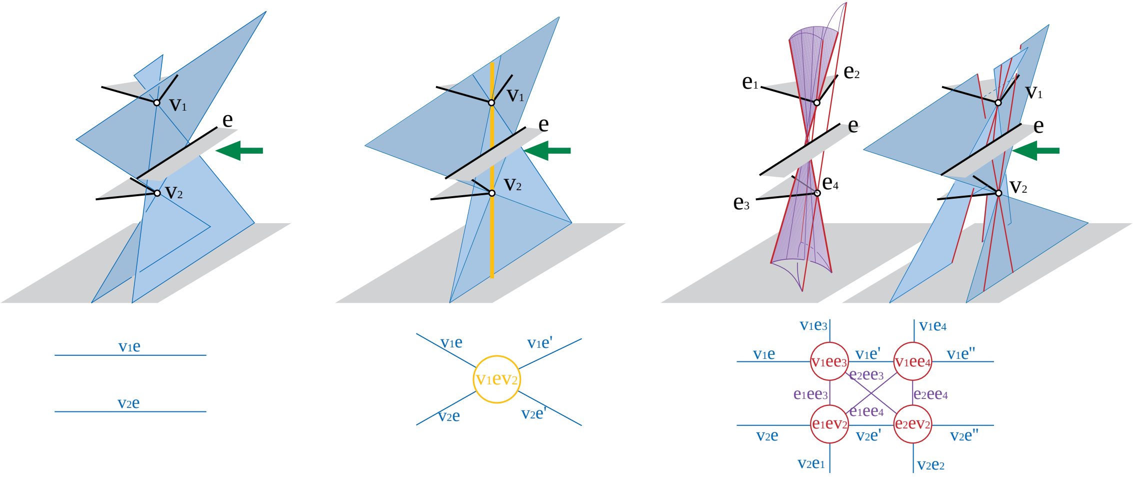
For scene (g) of Table 1, the part of the Visibility Skeleton with respect to one of the sources is computed in 4 min. 15 s. instead of 31 min. 59 s. for the entire scene.

When the number of sources becomes large, most of the time would be spent in checking if lines intersect the sources or if they have already been subdivided. If we need visibility information only between two objects, not between an object and the whole scene, we propose the use of ray classification of [1] together with the notions of dual space of [7] to build the visibility skeleton only where and when needed. The idea (which is not currently implemented) is to parameterize the lines of the 3D space (which is a set in 4D space), for example by their direction and projection on a plane or by their intersections with two parallel planes. We then perform a subdivision of the space of lines with a simple scheme (e.g., grid, hierarchical subdivision) and compute the nodes of the complex located inside a given cell of this subdivision.



(a) (b) (c)

Figure 14: (a)The complete discontinuity mesh with respect to the right source. (b) Discontinuity mesh between the lamp and the table. (c) Limits of the occlusions caused by a part of the plane between the computer screen and the right wall.



(a) (b) (c)

Figure 15: The edge moving from right to left causes a $VEV$ temporal visibility event which is the meeting of two $EV$ with the the two same extremities and with a common element (here the edge $e$). Four nodes are created, the $EV$ arcs are split into three parts and eight arcs are created. These events and the topological visibility changes are local in the visibility skeleton.

## 6 CONCLUSIONS AND FUTURE WORK

We have presented a new data structure, called the *Visibility Skeleton*, which encodes all global visibility information for polygonal scenes. The data structure is a graph, whose nodes are the extremal stabbing lines generated by the interaction of edges and vertices in the scene. These lines can be found using standard computer graphics algorithms, notably ray-casting and line-plane intersections. The arcs of the graph are critical line sets or swaths which are adjacent to nodes. The key idea for simplicity was to treat the nodes and deduce the arcs using the full catalogues of all possible nodes and adjacent arcs we have presented for polygonal scenes. A full construction algorithm was then given, detailing insertion of nodes and arcs into the Skeleton.

We presented an implementation of the construction algorithm and several applications. In particular, we have used the Skeleton to calculate the visible boundary of a polygonal face with respect to a scene vertex, the discontinuity mesh between any two polygons of the scene, the exact list of blockers between any two polygons, as well as the complete list of all interactions of a polygon with all other polygons of the scene.

The implementation shows that despite unfavorable asymptotic complexity bounds, the algorithm is manageable for the test suite used, both in storage and in computation time. In addition, we have developed and implemented a first approach to on-demand or lazy construction which opens the way to hierarchical and progressive construction techniques for the Skeleton.

The use of our implemented system shows the great wealth of information provided by the Visibility Skeleton. Only a few of the many potential applications were presented here, and we believe that there are many computer graphics (and potentially computer vision) domains which can exploit the capacities of the Skeleton.

In future work many issues remain to be investigated. From a theoretical point of view, the most challenging problems are the development of a hierarchical approach so that the Visibility Skeleton can be used for very complex scenes as well as the resolution of all theoretical issues for the treatment of dynamic scenes. Some of the problems for the dynamic solution are sketched in Fig 15. Adapting the algorithm to curved objects requires the enumeration of all relevant events and definitely has many applications.

Finally the field of applications must be extended: exact point to area form-factor from any point on a face, aspect graph construction, and incorporation into a global illumination algorithm.

## Acknowledgements

We would like to thank Jean-Dominique Gascuel for his AVL-tree code and Seth Teller for the very fruitful discussions we had and for all the suggestions he gave on conservative, lazy and practical approaches.

# 7 Appendix

## 7.1 Complete Catalogue of FACE Adjacencies

Face related events are adjacent to $FE$ elements $Fv$ elements as well as $EEE$ arcs when two non-coplanar edges are involved.

The interaction of a face with two edges is shown in Fig. 16, the interaction of a face a vertex and an edge is shown in Fig. 18 and finally the interaction of two faces is shown in Fig. 17.

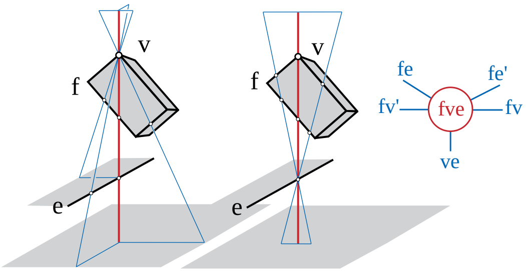


Figure 18: A $FvE$ node.

## 7.2 Details of the Construction to find the Orientation of Arcs

Finding the correct extremity of an arc when inserting a node is crucial for the construction algorithm to function correctly. We present here the most complex case, which is the insertion of an $E4$ node.

Consider the node $e_1e_2e_3e_4$ shown in Fig. 19, and the adjacent arc $e_1e_2e_3$. The question that needs to be answered is whether the node $e_1e_2e_3e_4$ is the start or the end node of this arc. To answer this query, we examine the movement of the line $l$ going through $e_1$, $e_2$ and $e_3$, when moving on $e_1$. The side of $e_4$ to which we move will determine whether we are a start or an end node.

Consider the infinitesimal motion $d\vec{\epsilon_1}$ on $e_1$. The corresponding point of $e_3$ on the $EEE$ will lie on the intersection of the plane defined by $e_2$ and the defining point on $e_1$. The motion of $d\vec{\epsilon_1}$ on $e_1$ corresponds to a rotation of $\alpha = \frac{\vec{\epsilon_1}.\vec{n}}{d_1}$ of the plane around $e_2$. Symmetrically, this rotation corresponds to the motion $d\vec{\epsilon_3}$ on $e_3$ and we have $\alpha = \frac{d\vec{\epsilon_3}.\vec{n}}{d_3}$, by angle equality. Thus, $d\vec{\epsilon_3} = \vec{e_3}\frac{d_3 d\vec{\epsilon_1}.\vec{n}}{d_1\vec{e_3}.\vec{n}}$.

Now we want to obtain $d\vec{\epsilon_4}$, the infinitesimal motion of the line going through the three edges around $e_4$. We consider the line as being defined by its origin on $e_1$ and by its unnormalized direction vector $\vec{dir}$ from $e_1$ to $e_3$. For the motion $d\vec{\epsilon_1}$ of the origin, the direction vector of moves by $d\vec{\epsilon_3} - d\vec{\epsilon_1}$, and thus $d\vec{\epsilon_4} = d\vec{\epsilon_1} + \frac{d_4}{d_3-d_1}(d\vec{\epsilon_3} - \vec{e_1})$.

The sign of $(\vec{\epsilon_4} \times \vec{e_4}).\vec{node}$ determines on which side of $e_4$ the line $l$ will move.

The adjacencies also depend on the face related to the edges which are visible from the other edges. The other cases are simpler and summarized in Table 2.

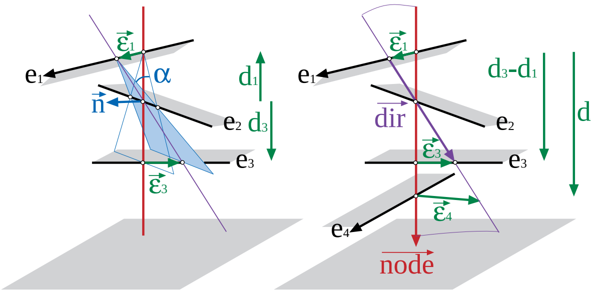


Figure 19: Determining the direction of an E4 node insertion.

| Node | Adjacent arc | Start or End Criterion |
|---|---|---|
| $v_1v_2$ | $v_1e_3$ | $v_1 == startV(e)$ |
| $ve_1e_2$ | $ve_2$<br>$e_3e_1e_2$<br>$e_2e_1e_3$ | $(\vec{e_2} \times \vec{e_1}).\vec{node} > 0$<br>$v == startV(e_3)$<br>$\vec{n} = \vec{normal}(v, \vec{e_2})$<br>$\vec{e_3}.\vec{n} * \vec{e_1}.\vec{n} > 0$ |
| $e_1ve_2$ | $ve_2$<br>$e_2e_1e_3$ | $(\vec{e_1} \times \vec{e_2}).\vec{node} > 0$<br>$\vec{n} = \vec{normal}(v, \vec{e_2})$<br>$\vec{e_3}.\vec{n} * \vec{e_2}.\vec{n} > 0$ |
| $e_1e_2e_3e_4$ | $e_1e_2e_3$ | $\vec{n} = \vec{normal}(\vec{e_2}, \vec{node});$<br>$\vec{\epsilon_3} = \vec{e_3}\frac{d_3\vec{e_1}.\vec{n}}{d_1\vec{e_3}.\vec{n}}$<br>$\vec{\epsilon_4} = \vec{e_1} + \frac{d_4}{d_3-d_1}(\vec{\epsilon_3} - \vec{e_1})$<br>$(\vec{\epsilon_4} \times \vec{e_4}).\vec{node} > 0$ |
| $e_1fe_2$ | $fe_1$<br>$e_1e_{f1}e_2$ | $\vec{e_2}.\vec{normal}(f) > 0$<br>$\vec{e_1}.\vec{normal}(f) > 0$ |
| $fe_1e_2$ | $fe_2$<br>$e_2e_1e_{f1}$ | $\vec{e_1}.\vec{normal}(f) > 0$<br>$\vec{n} = \vec{normal}(\vec{node}, \vec{e_1})$<br>$\vec{n}.\vec{e_2} * \vec{n}.\vec{e_{f1}} > 0$ |
| $fve$ | $fv$<br>$ve$ | $\vec{e}.\vec{normal}(f) > 0$<br>$\vec{e}.\vec{normal}(f) > 0$ |

Table 2: for each arc adjacent to a created node, there is a criterion that tells if it is a start node or an ending node.

## References

[1] James Arvo and David B. Kirk. Fast ray tracing by ray classification. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 55–64, July 1987.

[2] Daniel R. Baum, Holly E. Rushmeier, and James M. Winget. Improving radiosity solutions through the use of analytically determined form-factors. *Computer Graphics*, 23(3):325–334, July 1989. Proceedings SIGGRAPH '89 in Boston, USA.

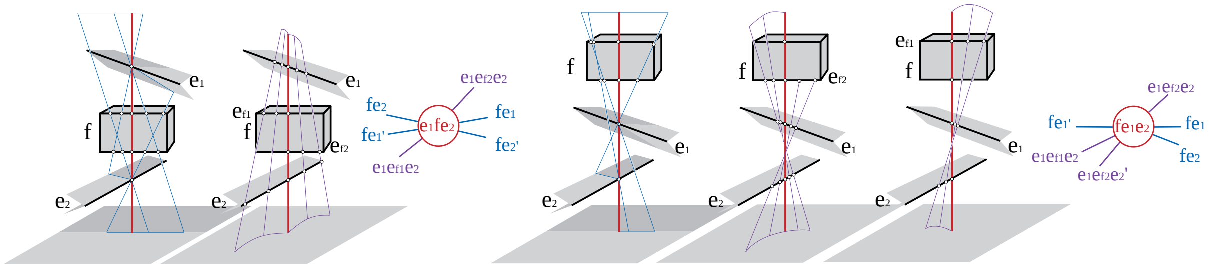
[3] Daniel R. Baum, John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. The back-buffer algorithm : an extension of the radiosity method to dynamic environments. *The Visual Computer*, 2:298–306, 1986.

[4] Michael F. Cohen and Donald P. Greenberg. The hemi-cube : A radiosity solution for complex environments. *Computer Graphics*, 19(3):31–40, July 1985. Proceedings SIGGRAPH '85 in San Francisco (USA).

[5] George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using back projection. In Andrew Glassner, editor, *SIGGRAPH 94 Conference Proceedings (Orlando, FL)*, Annual Conference Series, pages 223–230. ACM SIGGRAPH, July 1994.

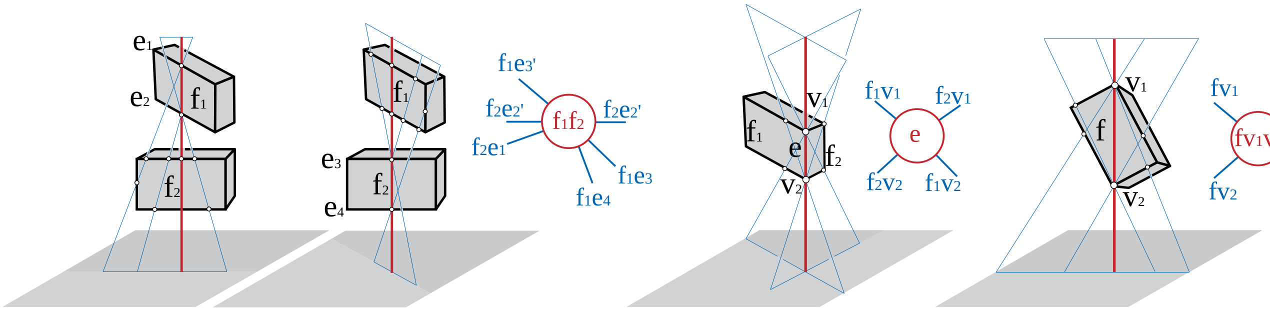
[6] George Drettakis and Francois Sillion. Accurate visibility and meshing calculations for hierarchical radiosity. In X. Pueyo and P. Shröder, editors, *Rendering Techniques '96*, pages 269–279. Springer Verlag, June 1996. Proc. 7th EG Workshop on Rendering in Porto.

[7] Frédo Durand, George Drettakis, and Claude Puech. The 3d visibility complex, a new approach to the problems of accurate visibility. In X. Pueyo and P. Shröder, editors, *Rendering Techniques '96*, pages 245–257. Springer Verlag, June 1996.



(a) (b)

Figure 16: An $EFE$ node.



(a) (b) (c)

Figure 17: (a) A $FF$ node, (b) an $Fe$ node and (c) and $Fvv$ node.

Proc. 7th EG Workshop on Rendering in Porto.

[8] David W. George, François Sillion, and Donald P. Greenberg. Radiosity redistribution for dynamic environments. *IEEE Computer Graphics and Applications*, 10(4), July 1990.

[9] Ziv Gigus, John Canny, and Raimund Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Trans. on Pat. Matching & Mach. Intelligence*, 13(6), June 1991.

[10] Ziv Gigus and Jitendra Malik. Computing the aspect graph for the line drawings of polyhedral objects. *IEEE Trans. on Pat. Matching & Mach. Intelligence*, 12(2), February 1990.

[11] Eric A. Haines. Shaft culling for efficient ray-traced radiosity. In Brunet and Jansen, editors, *Photorealistic Rendering in Comp. Graphics*, pages 122–138. Springer Verlag, 1993. Proc. 2nd EG Workshop on Rendering (Barcelona, 1991).

[12] Pat Hanrahan, David Saltzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206, August 1991. SIGGRAPH '91 Las Vegas.

[13] Stephen Hardt and Seth Teller. High-fidelity radiosity rendering at interactive rates. In X. Pueyo and P. Shröder, editors, *Rendering Techniques '96*. Springer Verlag, June 1996. Proc. 7th EG Workshop on Rendering in Porto.

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

[15] Michael Mc Kenna and Joseph O'Rourke. Arrangements of lines in space: A data structure with applications. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 371–380, 1988.

[16] Dani Lischinski, Brian Smits, and Donald P. Greenberg. Bounds and error estimates for radiosity. In Andrew S. Glassner, editor, *SIGGRAPH 94 Conference Proceedings (Orlando, FL)*, Annual Conference Series, pages 67–74. ACM SIGGRAPH, July 1994.

[17] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, November 1992.

[18] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. In Jim Kajiya, editor, *SIGGRAPH 93 Conference Proceedings (Anaheim, CA)*, Annual Conference Series, pages 199–208. ACM SIGGRAPH, August 1993.

[19] Rachel Orti, Stéphane Rivière, Frédo Durand, and Claude Puech. Radiosity for dynamic scenes in flatland with the visibility complex. In Jarek Rossignac and François Sillion, editors, *Computer Graphics Forum (Proc. of Eurographics '96)*, volume 16, pages 237–249, Poitiers, France, September 1996.

[20] M. Pellegrini. Stabbing and ray shooting in 3-dimensional space. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 177–186, 1990.

[21] H. Plantinga and C. R. Dyer. Visibility, occlusion, and the aspect graph. *Internat. J. Comput. Vision*, 5(2):137–160, 1990.

[22] M. Pocchiola and G. Vegter. The visibility complex. 1996. special issue devoted to ACM-SoCG'93.

[23] Erin Shaw. Hierarchical radiosity for dynamic environments. Master's thesis, Cornell University, Ithaca, NY, August 1994.

[24] A. James Stewart and Sherif Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In Andrew Glassner, editor, *SIGGRAPH 94 Conference Proceedings (Orlando, FL)*, Annual Conference Series, pages 231–238. ACM SIGGRAPH, July 1994.

[25] Filippo Tampieri. *Discontinuity Meshing for Radiosity Image Synthesis*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, 1993. PhD Thesis.

[26] Seth J. Teller. Computing the antipenumbra of an area light source. *Computer Graphics*, 26(4):139–148, July 1992. Proc. SIGGRAPH '92 in Chicago.

[27] Seth J. Teller and Patrick M. Hanrahan. Global visibility algorithms for illumination computations. In J. Kajiya, editor, *SIGGRAPH 93 Conf. Proc. (Anaheim)*, Annual Conf. Series, pages 239–246. ACM SIGGRAPH, August 1993.

[28] Seth J. Teller and Michael E. Hohmeyer. Computing the lines piercing four lines. Technical report, CS Dpt. UC Berkeley, 1991.

[29] John R. Wallace, Kells A. Elmquist, and Eric A. Haines. A ray tracing algorithm for progressive radiosity. *Computer Graphics*, 23(3):315–324, July 1989. Proceedings SIGGRAPH '89 in Boston.