Robust Epsilon Visibility

Florent Duguet, George Drettakis

Robust Epsilon Visibility

Florent Duguet and George Drettakis

REVES - INRIA Sophia-Antipolis, France, http://www-sop.inria.fr/reves/ *

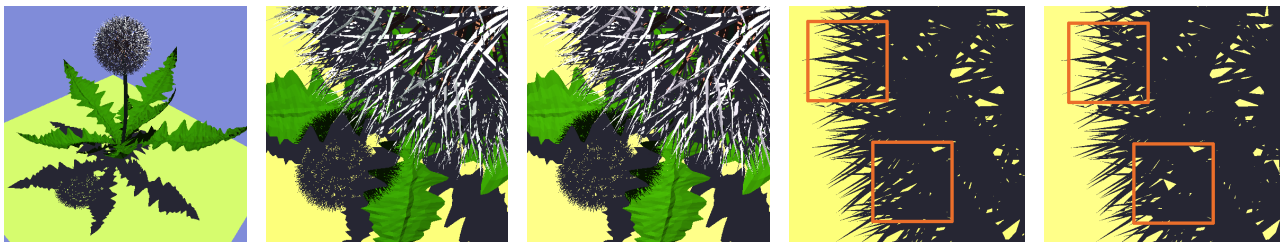

Figure 1: (Far left) A Dandelion model of 62,000 polygons. Using robust ε visibility, the output shadow mesh has 270,000 polygons for $\varepsilon = 10^{-4}$. From left to right: shadows on the leaves and ground (top view) for $\varepsilon = 10^{-4}$, and $\varepsilon = 10^{-2}$, resulting in 220,000 polygons, with no visible difference. Rightmost two images: zoom of the ground for $\varepsilon = 10^{-4}$ and $\varepsilon = 10^{-2}$; some missing features are now visible.

Abstract

Analytic visibility algorithms, for example methods which compute a subdivided mesh to represent shadows, are notoriously unrobust and hard to use in practice. We present a new method based on a generalized definition of extremal stabbing lines, which are the extremities of shadow boundaries. We treat scenes containing multiple edges or vertices in degenerate configurations, (e.g., collinear or coplanar). We introduce a robust ε method to determine whether each generalized extremal stabbing line is blocked, or is touched by these scene elements, and thus added to the line's generators. We develop robust blocker predicates for polygons which are smaller than ε. For larger ε values, small shadow features merge and eventually disappear. We can thus robustly connect generalized extremal stabbing lines in degenerate scenes to form shadow boundaries. We show that our approach is consistent, and that shadow boundary connectivity is preserved when features merge. We have implemented our algorithm, and show that we can robustly compute analytic shadow boundaries to the precision of our chosen ε threshold for non-trivial models, containing numerous degeneracies.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Visible line/surface algorithms

Keywords: Illumination, Shadow Algorithms, 3D Visibility, Robust Visibility Predicates, Epsilon Visibility.

* {Florent.Duguet|George.Drettakis}@sophia.inria.fr. The first author is also affiliated to the Ecole Nationale Supérieure des Télécommunications (ENST), Paris.

1 Introduction

Computing visibility is central for many aspects of computer graphics. From the most basic visible surface determination, to shadows and global illumination calculations, a major part of computational effort is spent on visibility. Although discrete approximations or sampling such as the z-buffer or ray-casting are often used in practice, *analytic methods* in which a continuous and precise solution is computed, can be very useful. In particular, for shadow calculations [16, 21, 6], or global illumination [10], the importance and utility of analytic visibility methods has been demonstrated. For virtual environments or games, graphics engines can now handle large polygon counts, even for low-end platforms. In certain cases, for games engines or virtual reality, texture memory may be a scarce resource, precluding its use for shadow representation. As a result, subdividing the input geometry into (partially) shadowed and lit sub-polygons using one of these methods could well be the best solution for displaying high-quality shadows for these applications. Note that such subdivision is a *view-independent* solution with shadows, with no additional shadow processing per frame. We will refer to this subdivision of the input scene as a *shadow mesh* in what follows. The shadow mesh can be created either for hard shadows (such as from a directional or point source) or for soft shadows from an area source.

In practice however, analytic methods have not been used because they suffer from robustness problems, algorithmic complexity and/or memory restrictions. These problems render them unusable for the type of scene used in virtual reality, games or other interactive applications. Such scenes are geometrically complex, and typically contain a large number of degeneracies: objects touch, edges are often aligned or coplanar etc. In addition, since floating point numbers are used both for the modeling phase and visibility computation, below a certain threshold, small features of the model or of the resulting shadows can lead geometric algorithms to fail. Object connectivity is not always given in such models and cannot always be reconstructed, and intersecting polygons are often present in the models. All of these properties can lead to robustness problems for geometric algorithms.

1.1 Motivation

To create shadow meshes, in particular for soft shadows, discontinuity meshing approaches [13, 6, 16] intersect shadow boundary

surfaces, or *swaths*, with scene geometry. This approach is inevitably unrobust for large and degenerate scenes, since numerical problems quickly lead to loss of connectivity in the shadow mesh, or result in geometrical errors due to small features.

We can observe however that shadow boundaries are delimited by *extremal stabbing lines* [23]. Two examples are shown in red in Fig. 2 (a). One way to generate shadow boundaries is to compute extremal stabbing lines, and then join them to form the actual *swaths* which constitute shadow boundaries. A simpler problem is thus solved, since in essence we only perform line- or ray-casting. This results in a stabler approach, as was shown in the Visibility Skeleton [9]. For non-degenerate scenes, the number of possible configurations of connectivity between extremal stabbing lines and swaths is finite and small. As a result, Durand et al., [9], developed a *catalog* of swaths and their neighboring lines to establish connectivity, based on a small set of non-degenerate configurations of edges, vertices and faces. The result of this construction, among other applications, is the shadow mesh.

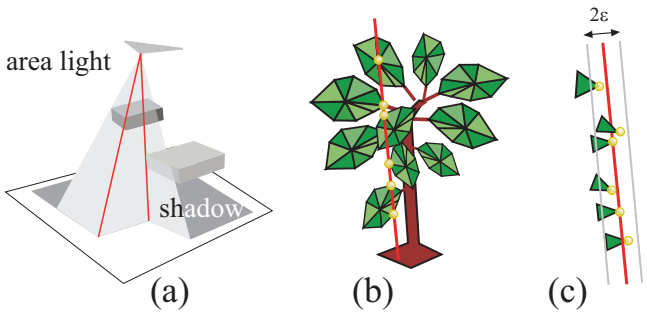

Figure 2: (a) Shadow boundaries are delimited by extremal stabbing lines (in red). (b) In complex models, stabbing lines may contain multiple collinear vertices. (c) If vertices are *almost* collinear, we choose to consider them as being on the same line.

Unfortunately, this approach will fail for complex scenes, such as those encountered in games or VR applications. Consider Fig. 2(b), where vertices of several leaves of a tree are collinear. Building shadow adjacencies based on a finite set of configurations is no longer possible, since a catalog capable of treating all cases would be infinitely big. Despite treatment of certain cases for the test scenes used in [10], a complete solution is lacking, as mentioned by the authors ([10], pages 164 – 166). This fact is also clearly demonstrated by the small size and specific (i.e., custom-built) nature of the test scenes used in all previous analytic approaches [6, 21, 16, 9, 10], in which the largest test scenes used were under 2,000 polygons. If edges or vertices are *almost* aligned or coplanar (see Fig. 2(c)) problems will appear, since resulting small shadow mesh features become sources of numerical instability, such as small, sliver triangles, edge side-test problems etc. Finally, many models are inaccurate, and may contain intersecting polygons or lack connectivity information. To use the catalog approach, expensive and numerically unstable preprocessing would have to be performed to re-facetize the input scene based on these intersections, and then to reconstruct all connectivity information around edges.

1.2 Contributions

To robustly compute shadow boundaries for real-world scenes, we develop an approach based on *generalized extremal stabbing lines*. Candidate extremal stabbing lines are proposed based on *native generators* for non-degenerate configurations [9], i.e., VV, VEE and $E4$, with V a vertex and E an edge. We attach *additional generators* to the candidate line, using ε methods to robustly determine whether a single feature (edge or vertex) is a generator or a blocker. The set of attached generators ensures that degenerate stabbing lines are correctly constructed, and allows us to robustly establish shadow boundary connectivity. We show that our ε method is consistent, by avoiding undesired propagation of ε contact and by ensuring consistent shadow-boundary connectivity.

The use of ε methods requires special attention when encountering faces (polygons), smaller than the ε threshold. We introduce robust blocker predicates both for the case when face connectivity is available, and for models lacking this information. We also treat models with intersecting polygons as input.

The resulting structure is sufficient for the generation of a shadow mesh. An interesting advantage of our methods is that when the value of ε used in the robust computation is increased, small shadow features will disappear, reducing the size of the shadow mesh.

These methods have allowed us to implement robust shadow mesh computation for soft and directional shadow boundaries for complex, real-world scenes, such as those shown in Fig. 1, 14 – 19.

2 Previous Work

Very early work in computer graphics attempted to compute shadows using polygon clipping methods [24]. These were the precursors of *analytic visibility* methods which have inspired our work.

It is beyond the scope of this paper to survey all work on visibility. An excellent survey of the whole spectrum of visibility methods in computer graphics can be found in F. Durand's Ph.D. thesis [8]. For directional and point light sources, shadow volumes [5] allow the computation of shadow boundaries. They suffer however from robustness problems and computational expense, both in the preprocess and for rendering since they are very polygon-fill intensive [17]. Various other methods have been developed, for example the methods of Campbell and Fussell [3] or Chin and Feiner [4] based on BSP-trees, which also run into robustness or memory problems due to the geometric operations and data structures involved.

Discontinuity meshing approaches were introduced by Heckbert [13] and Lischinski et al. [16]. These methods compute shadow discontinuity surfaces (line swaths), by intersecting them with the scene, with the resulting robustness problems previously discussed in Sect. 1.1. The ideas developed in the original aspect graph vision literature [14, 11], and introduced to graphics by Teller [23] have led to algorithms computing the complete discontinuity mesh [6, 21]. These include the backprojection data structure representing the visible part of the source for rapid computation of penumbral lighting. These methods suffer from the same robustness problems.

We have already mentioned the Visibility Skeleton [9], which computes extremal stabbing lines by casting them into the scene, and uses a finite catalog to establish swath connectivity. This approach was used to compute shadows, but also in a global illumination algorithm which adapts the radiosity mesh using perceptual criteria to insert discontinuities [10].

Many approximate or discrete methods have been developed for shadow computation (e.g., shadow maps [25]) and ray-tracing is evidently one of the most widespread point-sampling approaches used in practice. For soft shadows, convolution methods [20] give convincing results, which may contain some error however, depending on the configuration. All these methods use a discrete buffer, and thus resolution and resulting aliasing are the predominant problems. More recent work [1] presents an image-based approach comprising an efficient interactive solution with limited control of sampling and an expensive, view-dependent ray-tracing solution.

We adopt an ε geometry approach [18], albeit without perturbing the input data, using interval arithmetic to robustly treat degenera-

cies for our visibility computations. Interval approaches have been extensively used in computer graphics, for example for implicit functions and CSG [7] or by Snyder [19] for a variety of problems such as ray-tracing, interference detection etc. To our knowledge, ε methods have only been used to a limited extent for 3D visibility or shadow problems, notably by Bala et al. [2] for ray-tracing.

3 Generalized Visibility Events

As mentioned in the introduction, we will treat scenes containing degeneracies, such as those used in real-world applications. We first define our new framework, which is based on a general definition of extremal stabbing lines (ESL) and line swaths. This definition overcomes the limitations of a catalog of visibility events [9], since it will not depend on a finite enumeration of specific configurations. We first define the basic entities we use.

A *generator* is an edge or a vertex of the scene [1]. For edges we distinguish *silhouette edges* with respect to a given line direction, in the traditional sense of the term, i.e., that the edge is attached to a single face, or that one of the two faces connected to the edge is back-facing with respect to the line. A *silhouette vertex* is defined depending on its local blocking properties, i.e., whether it blocks an extremal stabbing line or not, depending on its neighboring faces. This is defined in detail in Section 4.1 (see also Fig. 6).

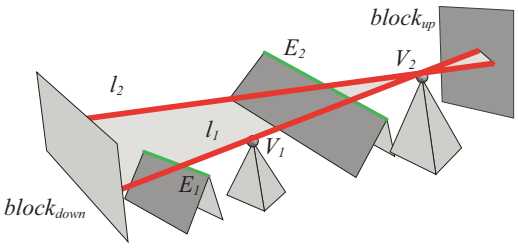

Figure 3: An extremal stabbing line l_1, with generators V_1, V_2, E_1, E_2, and up and down blockers $block_{up}$ and $block_{down}$, and a swath between ESL l_1 and ESL l_2, sharing E_2 and V_2.

A *blocker* is a face, or non-silhouette edge or vertex touched by a line.

A generalized extremal stabbing line (ESL) is a data structure, containing the corresponding line equation (also called *supporting line*), the set of associated generators, and its *up* and *down* blocker. A degenerate ESL, l_1, its generators and blockers are shown in Fig. 3.

The limits of a critical line swath, or simply *swath*, are its up and down blockers, and the start and end ESL's (see Fig. 3).

To establish shadow boundaries, we need to connect stabbing lines with swaths. Since we can no longer use a finite catalog, we will define when it is possible to connect pairs of ESL's. The intuition is that there needs to be a sufficient number of shared generators to define a planar (e.g., edge-vertex) or quadric (edge-edge-edge) swath joining the two ESL's. The process we use is procedural, and detailed in Section 4.3.

In what follows we consider visibility events between an "emitter" and a "receiver" pair. This is a logical choice for shadow computations, and allows us to define distances and sizes in a meaningful manner for our ε algorithms. This can be seen as a *lazy evaluation* of parts of the visibility skeleton [9]. In theory, the Visibility Skeleton could be computed using our algorithm, if we computed and stored all emitter-receiver pairs. This would however be impractical for any scene of reasonable size, due to the (at least) quadratic complexity in memory. Our ε approach is based on distance, defined across the shaft formed by the emitter and receiver. Thus merging operations vary depending on the chosen receiver-emitter pair; querying the entire structure would require special manipulation. However, for all the applications using the visibility skeleton to date [9, 10], (shadows, discontinuity mesh, global illumination), emitter-receiver pair computations suffice, so we do not consider this restriction significant.

[1] In contrast to [9], we do not consider faces as generators. In the context of a finite catalog, such a consideration is appropriate, since it allows a more concise presentation of possible configurations.

4 Enumeration and Validation of ESL's and Swaths

The first step in our approach is the enumeration and validation of extremal stabbing lines (ESL). We construct ESL's, associating all appropriate generators as we proceed, as well as start and end blockers, if they exist. The second step is the identification and validation of *swaths* connecting ESL pairs.

ESL's are "cast" into the scene to identify potential generators. We will use ε methods to consistently treat "almost" coplanar or "almost" collinear events, during this process. Before the enumeration and validation, we present the specifics of our ε methods.

4.1 ε methods

While casting lines, we need to identify whether a scene element, face (polygon), edge or vertex *interacts* with the line. If there is an interaction, it can be either a *blocking* interaction, i.e., we consider that the line is blocked, or can be a *generator* interaction, i.e., the line grazes the element, for example an edge, and thus we consider that the edge is a generator of the line. Consistent attribution of generators is a key to correct connectivity of shadow boundaries, since ESL's are connected by looping over generators.

We perform all calculations to determine whether an interaction exists, and its type, in floating point arithmetic, up to a precision of a pre-determined ε threshold.

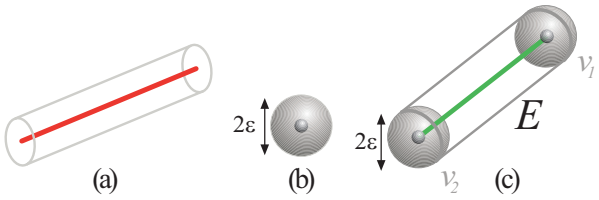

Figure 4: (a) A fat line or fat ray, (b) A fat vertex (c) A fat edge.

The ε parameter is given in the same units as the model of the scene. Thus if the scene is defined for example in meters, ε will also be defined in meters. Thus, the algorithm reacts as expected to scaling operations, i.e., if the entire scene is scaled by a coefficient of say 2, then the result will be the same in the topological sense, as if ε was divided by 2.

We use an interval arithmetic approach with floating point values for upper and lower bounds. We define ε contact in a natural manner: two elements are in ε contact, if and only if their distance is below ε. We say that an edge or vertex of the scene is in interaction with a line if and only if its distance to the line is below ε. For a vertex p and line we project the vertex onto the line to find the closest point q, and for a line and an edge, p and q are the pair of closest points of the two lines. An equivalent definition is that a

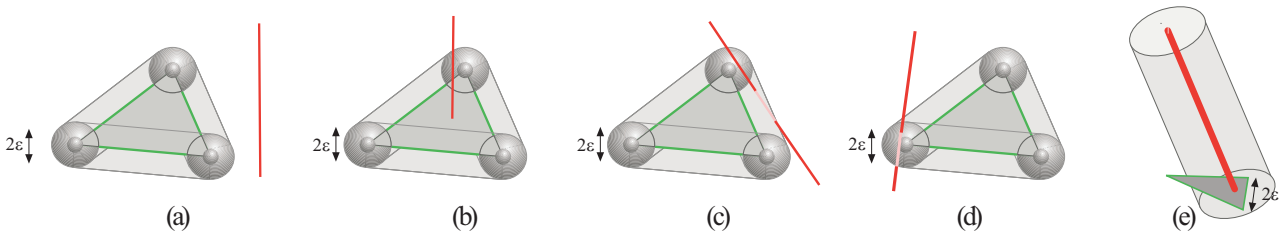

Figure 5: (a) No-stab, the line does not interact with the triangle (b) Full-stab, the line is blocked by the triangle (c) Edge-stab, the line touches the edge but nothing else (d) Vertex-stab the line touches the vertex but nothing else (e) Multi-stab: the fat line touches multiple elements.

vertex or an edge interacts with the line if it hits the associated fat element, i.e., sphere or shaft. This ε contact approach is central to our algorithm.

As a result of our ε approach, scene elements are transformed. A line is transformed into a shaft, which we call a *fat line* or *fat ray* (Fig. 4(a)). This shaft is the cylinder defined by the spheres of any two points centered on the line.

A vertex is a sphere of radius ε, which we will call a *fat vertex*, Fig. 4(b), and an edge is a cylindrical "shaft" (similar in spirit to that of [12]) between the two spheres of its extremal vertices, which we will call a *fat edge*, Fig. 4(c).

Since faces are only considered as blockers in the context of our approach, they remain as is.

There are five kinds of interactions between a line and a face, shown in Fig. 5. The first two cases, no-stab and full-stab, are handled trivially (Fig. 5(a) and (b)), since the first case results in no interaction, and the second results in the line being blocked. The third and fourth (Fig. 5(c) and (d)) are followed by a blocker/generator test described below, in order to check if the element is a blocker or a generator for the line. The last case is more complex and is the essence of our ε approach (Fig. 5(e)).

In Section 5, we introduce techniques to robustly determine whether this configuration blocks the line.

The blocker/generator tests for edges and vertices proceed as follows. Since the line may not pass exactly through the edge, for an edge-stab we form a plane Π with the edge (points A and B in Fig. 6(a)) and normal to a direction perpendicular to the line with its origin on the edge. We can always define this plane since we do not have a vertex-stab, and thus the edge is not on the line. We then compute the position of the faces attached to the edge with respect to this plane. If faces are present on both sides, the edge is a blocker, otherwise, the edge is a generator (Fig. 6(b)). This is equivalent to determining whether we have a silhouette edge with respect to the line.

For the vertex stab, let Π be a plane orthogonal to the line, for instance a plane passing through the vertex. We compute the orthogonal projections of the faces surrounding the vertex onto Π. Each face around the vertex can be seen as a slice of a pie. We then merge the slices in contact. If the slices merge into a whole pie, then the vertex is a blocker, otherwise, it is a generator (see Fig. 6(c)).

4.2 Enumeration and Validation of ESL's

We have now defined the framework for generalized visibility events and our ε visibility approach, which are the basic elements required to define our new, robust, algorithm. We start with the enumeration of non-degenerate or *generic* ESL's, that is VV, VEE and $E4$, as defined in previous work [23, 9]. For VV and VEE the line equation is defined as in [9]. For $E4$ ESL's we adopt the method previously proposed by Teller [22]. An alternative would be to use the bisection approach described in [9].

The choice of which generators are considered as *native* depends on the order in which we evaluate them: We demonstrate below that

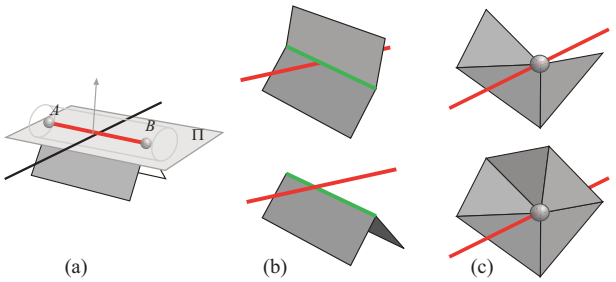

Figure 6: (a) Determining whether an edge is a blocker for an edge-stab (b) Top, the edge is blocked, bottom the edge is a generator (c) Top the vertex is a generator, bottom, the vertex is a blocker.

a consistent result is obtained with our method independently of the order of evaluation.

The algorithm proceeds with an *ESL casting* process. First, generic ESL's are formed by enumeration of native generators. For each such generic line, we propose a *candidate ESL* and cast it through the scene. The ESL-casting process has a dual function: we find and attach all additional generators to form the appropriate set attached to a degenerate ESL, and we test visibility of the stabbing line to decide whether the candidate ESL is valid. Finally, redundant ESL's are also eliminated.

4.2.1 ESL Casting

A *candidate* ESL is defined by a line and a set of native generators (VV, VEE, $E4$). The ESL casting process is very similar to a traditional ray-casting algorithm, but is based on "fat rays" as described previously. The casting process identifies the elements, either face, edge or vertex, which interact with the ESL, as described in the previous section. The fat ray emanates from the emitter, and we visit elements in order of increasing distance from the origin of the fat ray. This is achieved cheaply by the traversal routine of the acceleration structure described below; a sort is performed on elements in the interior of a cell of the acceleration structure.

For each such element, we test if it is a blocker, using the ε stabbing process. If it is, we flag the element as the down blocker of the extremal stabbing line, and the casting process is stopped. If not, we add the element to the set S of *stabbed elements* associated with the ESL, which is initially empty. Note that the native generators are *not* contained in S at the outset of the casting process.

Once the ray is stopped, we compare the set of stabbed elements S, and the set of native generators N. If $N \subset S$, then all the native generators are touched by the candidate ESL, which is then validated. The set of stabbed elements also constitutes the set of generators associated with the ESL. This process is illustrated in Fig. 7. To accelerate swath validation, references to the ESL's are stored on the scene edges and vertices.

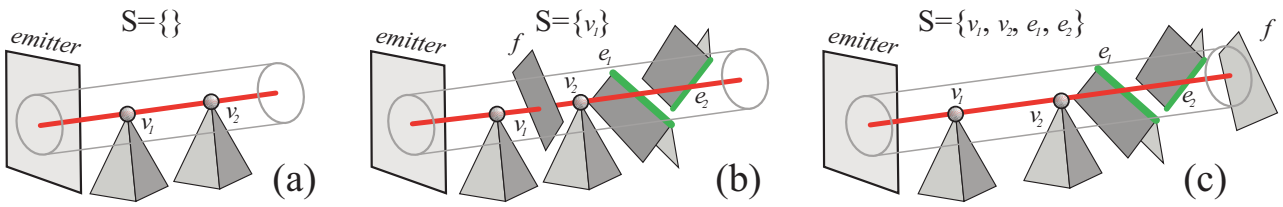

Figure 7: The ESL casting process. (a) Before casting, the line is formed by the native generators v_1 and v_2, S is empty and the line originates at the emitter. (b) The ESL encounters the generator v_1, and in this configuration, the ESL is invalidated by the face f. (c) Here, the ESL encounters generators v_1, v_2, e_1, e_2. This is a valid ESL, with down blocker f, since S contains the native generators.

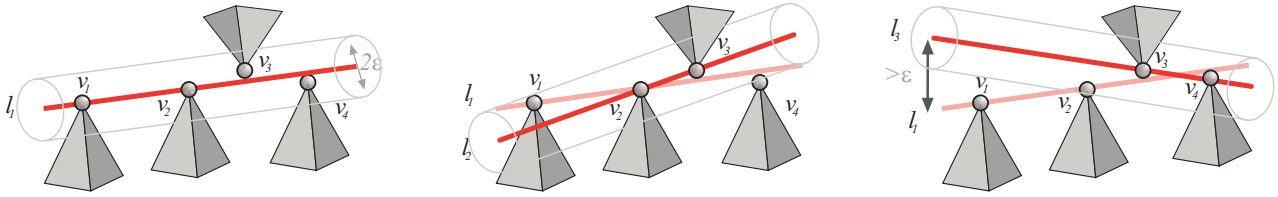

Figure 8: Elimination of redundant ESL's. (a) The native generators of l_1 are v_1 and v_2, and we add v_3 and v_4. (b) Native generators of l_2 are v_2 and v_3, and we add v_1. l_1 and l_2 satisfy the combinatorial criterion and the topological criterion, and thus l_2 is eliminated. (c) Native generators of l_3 are v_3 and v_4, and we add v_2. Even though the combinatorial criterion is satisfied with respect to l_1, the topological criterion is not, and thus l_3 is maintained as a separate ESL.

We use standard acceleration techniques for fat-ray casting. Faces are simply added to the acceleration structure cells if they are either inside the cell or at a distance less than ε. We use a recursive grid for faces and subdivide based on a $\sqrt[3]{n}$ criterion, where n is the number of polygons at each level. Most other acceleration techniques for ray tracing could also be used for our ESL casting.

4.2.2 Elimination of Redundant ESL's

As mentioned above, we consider the order in which we traverse generators to be irrelevant. For this to give a consistent result, we need a method which will *eliminate* redundant ESL's when they are "sufficiently close" to an existing ESL, with respect to the ε chosen. This elimination step must be done carefully, to avoid propagation of ε across generators, which would result in the "merge" of lines which in reality are geometrically far apart.

We thus define a *consistent* elimination method for ESL's, based on two criteria: a combinatorial and a topological criterion.

An ESL l_c satisfies the combinatorial criterion with respect to another ESL l_e, if and only if the set of its native generators is *completely* contained in the set of generators of the other ESL l_e.

An ESL l_c satisfies the topological criterion with respect to another ESL l_e, if and only if the distance between the two supporting lines is less than ε.

The distance between two lines used for this test is defined with respect to the emitter-receiver shaft. In particular it is defined as the maximum distance (over the shaft), between any point of one line with respect to the other line. If an ESL satisfies both criteria with respect to another, existing ESL, then it is redundant and is eliminated. The entire process is illustrated in Fig. 8.

Note that the test can be performed before the actual ESL casting. We thus compare the *candidate* line to existing ESL's preventing unnecessary computation implied by the actual casting algorithm. We only search a subset of existing ESL's, since we first need to satisfy the first criterion.

4.3 Enumeration and Validation of Line Swaths

Once we have generated all the ESL's, we create the swaths which join them, if required by the application. Since we have foregone the catalog, we use a procedural construction. Swath creation involves two steps, enumeration, to determine whether a swath can exist between two ESL's, and validation, which verifies visibility with respect to the rest of the scene.

Swaths are found by looping over all edges and vertices in the scene. We first loop over all pairs of edges EE to treat the special case of coplanar edges. We also collect any other edges which are coplanar to this pair; We discuss details of this case later. At the same time we find any potential EEE swaths. We then loop over all V and E pairs.

For each potential swath, EEE or EV, we examine the extremal stabbing lines shared by each generator, E or V. If there is a shared pair of ESL's, we create a candidate swath, which is then tested for visibility.

We need to validate the candidate swath, that is treat visibility with respect to the other elements of the scene. In the catalog approach [9], this step is not necessary, since it is included in the local connectivity information. An example of a swath which must be discarded is shown in Fig. 9.

Since all ESL's have been computed, there cannot be a visibility discontinuity along the swath, otherwise it would be represented by an ESL. Thus, to validate the swath, it suffices to sample visibility by a line in the middle of the line set, that is the line through the point in the middle of the edge portion concerned by the swath, which we call the *midline*.

The midline is then cast in the same manner as an ESL into the scene, using the acceleration structures. The swath is valid if the midline is not blocked before reaching the receiver.

A special case occurs for coplanar edges, as was pointed out in [13]. This special case corresponds to a line set of dimension 2, as opposed to other swaths (EEE and EV) which are of dimension 1, since they only have one degree of freedom (along the E in an EV for example). We first collect all the ESL's contained in the plane, and then find the *extremal* lines, in the sense of the emitter-receiver pair, shown in red in Fig. 10. We then sort the ESL's across the generating emitter edge, and perform the visibility test for each

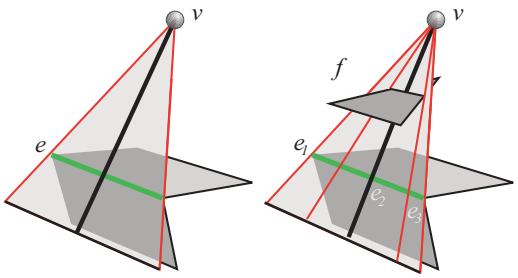

Figure 9: Swath validation. (a) The swath ev is valid since the midline does not touch any other object. (b) The swath e_2v is invalid, since the midline is blocked by the face f, and the receiver has not been encountered.

individual swath separately. For the shadow application, we create a line segment on the receiver with the boundary ESL's, we then calculate the intersections of all other ESL's, and insert the shadow mesh segments sorted from the start to the end boundary ESL's.

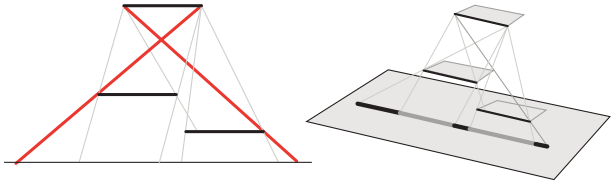

Figure 10: Coplanar edge treatment and shadow mesh segment insertion.

4.3.1 Swath Connectivity Consistency

To demonstrate consistency, we adopt the approach of the visibility skeleton [9], in which ESL's are the nodes of a graph, connected by the swaths which are arcs.

We want to demonstrate that if we merge a certain number of nodes together, due to our ε approach, the graph remains consistent. By consistent, we mean that all arcs which could potentially exist in the new configuration are preserved.

Consider the example shown in Fig. 11 (left). If we increase ε sufficiently, l_1 and l_2 will merge, Fig. 11 (right).

Once the merge has taken place, the graph becomes as shown in Fig. 11, upper right. In practice, all generators of l_2 are attached to l_1, assuming that l_1 was created first. We refer to the new "merged" node as l_m. We need to show that all arcs which were connected to all the nodes before merging are taken into account. Arcs which connected the two merged nodes no longer exist. We need to show that arcs originating at another node and arriving at one of the nodes merged into l_m, will now be connected to l_m.

This is simple to show, since the generator set of the merged node l_m is the union of the generator sets of all constituent nodes (l_1 and l_2). Consider any given external node l_e (e.g., l_3) which was linked to one of the constituent nodes before the merge. The merged node l_m thus contains the necessary generators and we are able to create an arc between l_e and l_m, when looping over generators for swath enumeration.

When creating swaths between merged nodes, an edge is necessarily shared between them. This edge is used for the midline computation, thus guaranteeing a consistent result, since it will reflect the original structure of the shadow boundary.

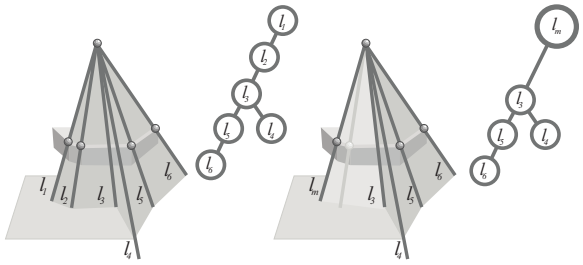

Figure 11: (Left) Original configuration with a small ε. The corresponding graph is shown in the upper left. (Right) The ε threshold is higher, and thus l_1 and l_2 will merge. A merged node l_m now takes the place of l_1 and l_2 in the graph.

5 Robust Blocker Predicates for ε Visibility

The use of fat lines, or an ε approach, implies that all geometric computation is performed up to a certain precision. Once the threshold is chosen some elements may be of size smaller than the given ε. For instance, consider the example shown in Figure 12(a) where line l interacts with many small faces. Since the faces are smaller than the ε chosen, they should not be considered individually.

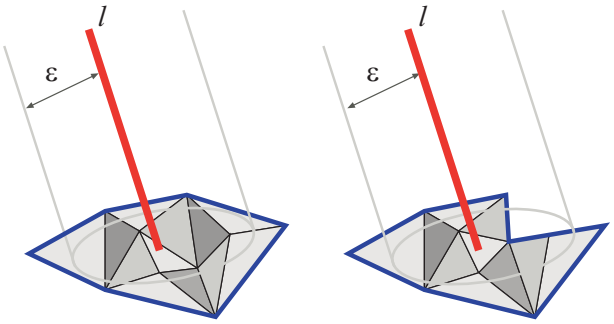

Figure 12: (a) A fat line covers several faces, and the fat line is blocked since it is contained inside the multi-face contour, in black. (b) The multi-face does not block the fat line, since it is not contained in the multi-face contour.

In particular, we need a robust way of determining whether the line is blocked by multiple faces.

For this, we develop two solutions: the *multi-face* structure, which assumes that the scenes contain faces with connectivity information; and the *blocker-fan*, which treats multiple interactions without connectivity information, and handles surface contact.

If the original model does not contain connectivity information we reconstruct it as much as possible, by finding vertices at the same position and determining faces sharing an edge. The blocker-fan is used for all remaining cases, due either to numerical imprecision or touching faces.

5.1 The Multi-face Structure

If an ESL has multiple interactions with a face (e.g., Fig. 5(e) or Fig. 12(a)), we apply the multi-face approach. The multi-face is a data structure containing the faces interacting with the fat line, and the contour of these faces used for the blocker test. The idea of the multi-face is related to the application of face-clustering to

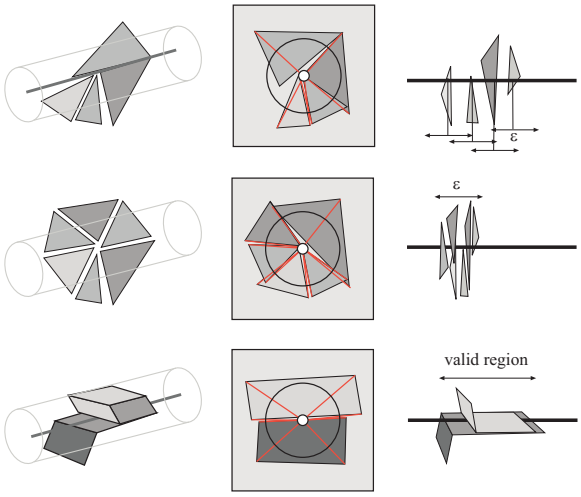

Figure 13: Blocker-fan: (Left) Perspective view of the unconnected faces encountered by the line. (Middle) The slices, on a plane perpendicular to the line. The red lines show the angular slices. (Right) View of the slice along the line direction to verify depth overlap. Upper row: The line is not blocked, both due to depth and to slice coverage. Middle row: The line is blocked, since it satisfies both criteria. Lower row: Special case of surface contact. The valid region is defined by the surfaces in contact.

visibility presented by Leblanc and Poulin [15], but is developed and used in a very different context.

For each surrounding face which is touched by the line, in the ε sense (i.e., distance less than ε), we add the face to the multi-face. All connected faces touched by the line are added if the edge is not a silhouette with respect to the line. Once the set is complete, we identify the boundary edges of this set.

To determine if the multi-face blocks a given fat line, we check if the projection of the boundary edges onto a plane orthogonal to the line is a complete loop around the line. If it is, then the multi-face is a blocker; Otherwise, it is not. In addition, the line must be entirely contained in the loop in the ε sense (see Fig. 12). This test is performed by traversing the neighboring faces to the line, adjacent to the face(s) interacting with the line.

5.2 The Blocker-fan structure

For unconnected polygons we use a different approach. We collect a "fan" of independent polygons encountered along the line of sight of the fat ray, as it is propagated through the scene. The elements encountered are either edges or vertices, and they form slices with respect to the shaft, as shown in red in Figure (13, middle column). We also consider an ε depth for each slice, in the direction of the line. The depth is taken around the point of intersection of the edge or the vertex with the line.

As with the multi-face, we project the faces onto a perpendicular plane. For each face, we take the edges outside the fat line, and create angular slices (see Fig. 13). If the slices collected cover a whole section of the shaft, and they overlap in depth, then we have a blocker. Note that, as with fat-ray casting (see Sec. 4.2.1), we visit elements in order of distance on the line.

The blocker fan also handles surface contact. In particular for touching objects and a line passing "between" them, we perform the same slice operation (see Fig. 13, lower row), and verify that the slices overlap in depth. Note that the blocker fan works together with fat ray casting, so that the routines must be called by the casting process.

6 Implementation and Results

We have implemented a system based on the algorithms presented above. In addition to the methods described in the previous sections, we compute intersections between polygons. Instead of refacetizing the input polygons along these discontinuities, we consider them as special generators and blockers, rather than true polygonal elements. They are thus attached to the original input geometry.

We have also implemented an octree acceleration structure for the hourglass formed by two edges when looping over generators as described in [9].

All the results and tests we present have been run on scenes containing unmodified objects we have either found on the internet or which are used in real applications. All timings are on a Pentium III 1 Ghz PC under Linux, with a GNU C++ implementation.

Figure 14: "Big" scene, overview and details of shadows on flower, leaves and ground.

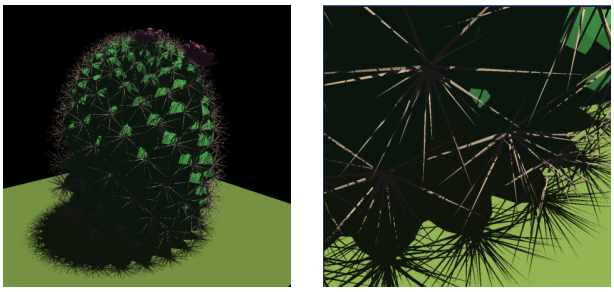

Figure 15: Two views of the cactus model.

The example scenes we use are a Dandelion (Dand) (Fig. 1), a Cactus (Fig. 15), "Vizzy the skeleton" (Fig. 16), the Agave plant (Fig. 19 right), the "Big" scene (Fig. 14), which is the largest scene tested, and a very degenerate aligned cube scene (Fig. 17). For soft shadows we have also used the Tree of "Big", containing 33,000 input polygons (Fig. 19, middle).

6.1 Shadow Computation

To compute shadow boundaries for directional light sources we only need to enumerate ESL's in the direction of the light source, and we only have planar swaths. We use all algorithms developed; that is the general definition of ESL, ε methods, multiface, blocker fan and intersecting polygons. Results for directional light sources are shown in Fig. 14 – 16.

We have also computed a solution for an extremely degenerate model, which is a cube of cubes, in which everything is aligned, shown in Figure 17. This computation took 20 minutes.

To compute soft shadows, we compute the ESL's and the swaths for each such emitter-receiver pair, as described previously. Note

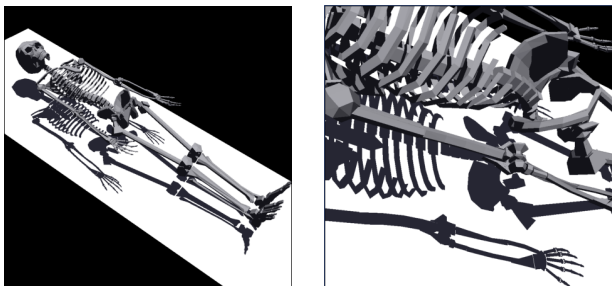

Figure 16: An overview and a zoom of "Vizzy the skeleton" with a directional light.

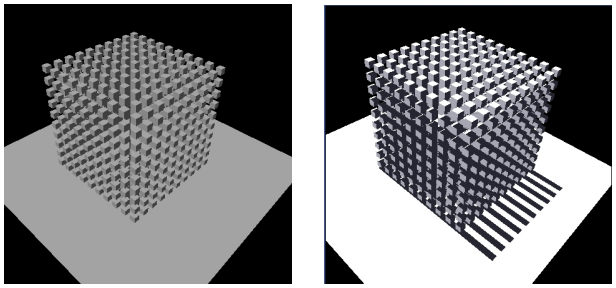

Figure 17: The cube of cubes degenerate model and resulting shadows (directional light source parallel to the cubes).

that we only compute visibility events with one generator on the source; in the full discontinuity mesh [6, 21] other events also exist, but we choose to ignore them. We made this choice for computational efficiency and because the impact of these events on shadows is usually minimal. The algorithm could easily be adapted to compute them all, at the cost of an expensive additional search for ESL's within the shaft.

To create images with shadows, we insert all discontinuity lines into a Constrained Delaunay Triangulation (CDT), and then sample the light source. Nonetheless, standard constrained Delaunay triangulation packages are numerically unrobust. For soft shadows, the CDT packages we tried failed for very complex models. We show however the discontinuity edges for an area light source computed by our algorithm, without actually computing a CDT and the resulting shadows. We computed the discontinuity edges on all objects for the Vizzy and Tree models. Due to memory limitations of our current implementation, we restricted the computation to the floor only for Agave. These results are shown in Figure 19.

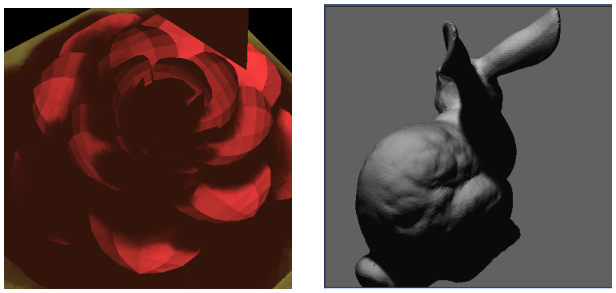

Figure 18: Soft shadow on the rose and on a 69,000 polygon bunny.

For the rose (1,700 polygons) and the bunny (69,000 polygons) models, the CDT was able to produce a soft shadow mesh, Fig. 18.

Scene/ε	Vizzy/$1e^{-4}$	Cactus/$1e^{-4}$	Dand/$1e^{-4}$	Agave/$1e^{-4}$	Big1e^{-4}
polys	11,000	34,000	61,000	114,000	121,000
time	0m35s	1h10m	2h40m	0h04m	0h10m
smesh	31,600	428,000	287,000	153,000	329,500
ESL	4,300	45,000	75,000	20,000	19,000
dESL	535	19,600	7,700	5,000	4,300
mface	254	17,000	1,200	7,700	3,100

Table 1: Statistics for test scenes with directional shadows, *polys*: number of input polygons, *time*: time to compute shadow mesh, *smesh*: size of shadow mesh, *ESL*: number of generic ESL's, *dESL*: number of degenerate ESL's, *mface*: number of ESL's implying multi-faces.

6.2 Statistics

As we can see from Table 1, we can treat models of significant complexity (from 11,000 to 121,000 input polygons), and the resulting shadow meshes multiply the polygon count by 2-5 times for a directional source; this a reasonable overhead for high-quality shadows. The Cactus is an exception where we have a factor of 13 increase. For soft shadows, the shadow mesh for the rose is 49,500 polygons and required 2h, while for the bunny it is about 95,000 polygons and required 3h; the respective times for computing the discontinuity edges only were 8m and 1h46m. Our soft shadow computation could be significantly optimized by using a hierarchical data structure on the scene graph and using more appropriate acceleration structures. For the discontinuity edges only, the times are 1h43m for Vizzy, 17h30m for Tree and 2h40m for Agave (floor only). Other than the unoptimized implementation, the reason for the high computational cost for Tree is also the large number of E4 events (91,000) compared for example to Vizzy (446). It is also interesting to note the number of swaths, 464,000 for Tree compared to 34,000 for Vizzy.

ε	Agave $3e^{-3}$	$8e^{-4}$	Vizzy e^{-2}	e^{-3}	Dand $2e^{-3}$
polys	114,000	114,000	11,000	11,000	61,000
time	0h15m	0h06m	1m45s	0m39s	1h48m
smesh	139,000	146,000	22,600	29,600	221,000
ESL	8700	15,300	1,250	3,500	55,000
dESL	7400	6,000	1,800	969	19,400
mface	7600	5,000	2,179	645	7,700

Table 2: Additional statistics for test scenes for varying ε.

It is interesting to note the much higher number of ESL's for the Cactus which has multiple interactions between spikes, compared to the Vizzy model which is quite sparse and does not contain multiple interactions. The percentage of degenerate ESL's varies, but is always more than 10%, and in some cases the number of degenerate ESL's is at the same level as the non-degenerate lines, even for small ε (see Table 2) which underlines the importance of our approach. Similarly, the number of multi-face operations on the various models shows that the systematic treatment we propose is indispensable. The running times do not always decrease with the increase in ε, due to the potential additional complexity of certain steps of our algorithm such as the blocker-fan computation.

7 Discussion and Conclusion

We have presented a novel, systematic approach to robust shadow boundary computation on real-world models. Our method first redefines visibility events in a generalized framework, thus taking into account typical degenerate configurations, notably collinear and coplanar scene features. We then develop an algorithm to robustly compute generalized extremal stabbing lines using ε methods, and a procedural approach to connect shadow boundaries. We

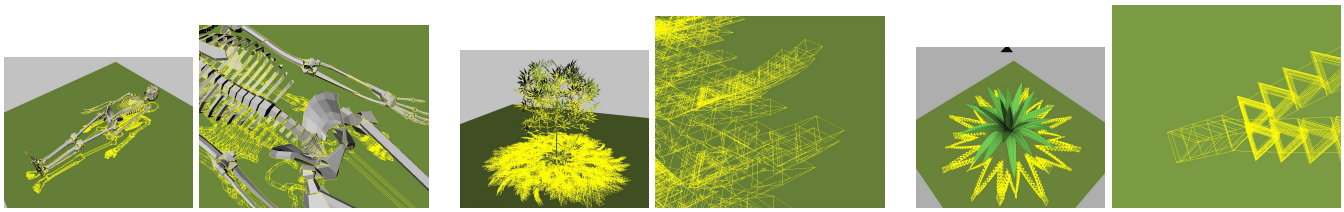

Figure 19: Discontinuity edges for an area light source for three complex models (Vizzy, Tree and Agave – floor only).

demonstrated that our approach is consistent both in terms of the usage of ε and in preservation of shadow boundary connectivity. Finally, we present robust blocker predicates for ε computations, both for connected models, and models which do not have connectivity information. We also handle intersecting polygons. Our results show that we can robustly compute both directional and soft shadow boundaries on complex models.

Our method greatly improves the usability of analytic shadow algorithms for real scenes. We consider the idea of consistent feature merging to be of particular interest, since it may lead to an interesting development of hierarchical visibility methods. Currently, the multi-face approach requires connectivity information. If connectivity information is missing, interesting problems arise, since, for large ε values we may wish to merge small features which are relatively far apart, such as the bones of the Vizzy model Fig. 16. We find this line of research interesting, and inevitably dependent on application requirements in terms of output.

Evidently, our approach is limited by the inherent computational complexity of visibility events. It is known that the worst-case computational complexity of the full discontinuity mesh is $O(n^4)$ for the E4 ESL's, with n the number of objects/edges, multiplied by mesh construction cost which can be $O(n^2)$ on each object. In cases where the number of E4 events is large, such as the Tree example, this effect becomes significant, although the worst case bounds are still very pessimistic. In our approach ESL enumeration and elimination are brute-force, but cheap, while the more expensive ESL validation step is output dependent. Developing optimal algorithms is an interesting research direction, and should follow the spirit of sweep approaches (e.g., [21]).

As with any robust algorithm, it is interesting to discuss possible cases of failure. The use of ε predicates means that we will never block an ESL which should not be blocked. For the same reason however, we may enumerate and validate an ESL which *should* have been blocked. This is not problematic since no swath will be connected to this ESL, and therefore the output will remain consistent. In its current form, we do not guarantee topological consistency of shadows for very large ε values; shadows simply disappear after a certain point. More sophisticated application-dependent approaches could be developed to guide the way simplified swaths are created.

Our approach could be directly applied to the global illumination method of [10], since all operations are performed in that method on a link basis, which is exactly equivalent to the emitter-receiver pair we use here.

Acknowledgments

Thanks to P. Poulin for reading an early version and M. Stamminger for his many comments and for making the video. This work was partially supported by the INRIA Action de Recherche Coopérative (ARC Vis3D) on 3D Visibility. The initial ideas leading to this work originated in F. Durand's Ph.D. thesis. We would like to thank him for the many fruitful discussions and exchanges.

References

[1] M. Agrawala, R. Ramamoorthi, A. Heirich, and L. Moll. Efficient image-based methods for rendering soft shadows. In *ACM SIGGRAPH 2000*, Annual Conference Series, pages 375–384, July 2000.

[2] K. Bala, J. Dorsey, and S. Teller. Radiance interpolants for accelerated bounded-error ray tracing. *ACM Transactions on Graphics*, 18(3):213–256, July 1999.

[3] A. T. Campbell, III and D. S. Fussell. Adaptive mesh generation for global diffuse illumination. *Computer Graphics (Proc. SIGGRAPH '90)*, 24:155–164, August 1990.

[4] N. Chin and S. Feiner. Fast object-precision shadow generation for areal light sources using BSP trees. In *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, pages 21–30, March 1992.

[5] F. C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (Proc. SIGGRAPH 77)*, 11(2):242–248, July 1977.

[6] G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using back projection. In *ACM SIGGRAPH 94*, Annual Conference Series, pages 223–230, July 1994.

[7] T. Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *Computer Graphics (Proc. SIGGRAPH'92)*, 26(2):131–138, July 1992.

[8] F. Durand. *3D Visibility: analytical study and applications*. PhD thesis, Université Joseph Fourier, Grenoble I, July 1999. http://www-imagis.imag.fr.

[9] F. Durand, G. Drettakis, and C. Puech. The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool. In *ACM SIGGRAPH 97 (Los Angeles, CA)*, Annual Conference Series, August 1997.

[10] F. Durand, G. Drettakis, and C. Puech. Fast and accurate hierarchical radiosity using global visibility. *ACM Trans. on Graphics*, 18(2):128–170, Apr 1999.

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

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

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

[14] J. J. Koenderink and A. J. van Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32(4):211–216, 1979.

[15] L. Leblanc and P. Poulin. Guaranteed occlusion and visibility in cluster hierarchical radiosity. In *Rendering Techniques 2000, (Proc. 11th Eurographics Workshop on Rendering 2000)*, pages 89–100, June 2000.

[16] D. Lischinski, F. Tampieri, and D. P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE CGA*, 12(6):25–39, November 1992.

[17] nvidia. webpage. `http://developer.nvidia.com/view.asp?IO=cedec_stencil`.

[18] D. Salesin, L. Guibas, and J. Stolfi. Epsilon geometry: Building robust algorithms from imprecise computations. In *Annual Symposium on Computational Geometry*, 1989. Saarbrucken, West Germany.

[19] J. M. Snyder. Interval analysis for computer graphics. *Computer Graphics (Proc. SIGGRAPH'92)*, 26(2):121–130, July 1992.

[20] C. Soler and F. X. Sillion. Fast calculation of soft shadow textures using convolution. In *ACM SIGGRAPH'98*, Annual Conference Series, pages 321–332, Jul 1998.

[21] A. J. Stewart and S. Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In *ACM SIGGRAPH 94*, Annual Conference Series, pages 231–238, July 1994.

[22] S. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, UC Berkeley, 1992.

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

[24] K. Weiler and K. Atherton. Hidden surface removal using polygon area sorting. *Computer Graphics (Proc. SIGGRAPH 77)*, 11(2):214–222, July 1977.

[25] L. Williams. Casting curved shadows on curved surfaces. *Computer Graphics (Proc. SIGGRAPH 78)*, 12(3):270–274, August 1978.