from Proceedings of the Fifth Canadian Conference on Computational Geometry, pp 291--296

An Output Sensitive Algorithm for the Computation of Shadow Boundaries

A. James Stewart
Sherif Ghali

Department of Computer Science
University of Toronto

Abstract

Given a set of polyhedra and a polygon σ in E^3, the σ–aspect graph is a partition of the faces of the polyhedra into regions such that the area of σ that is visible from each region has the same structure or "aspect." We present an output-sensitive algorithm to compute the σ–aspect graph. In computer graphics, this problem arises in rendering images of polyhedral scenes illuminated by area light sources. This is the first analytical solution for the problem and the algorithm is simple enough to be useful in practice.

1 Introduction

A problem that arises in computer graphics is the computation of the shadow boundaries of a set of polyhedra lit by a polygonal light source. The shadow boundaries are curves on the faces of the polyhedra at which there is discontinuity in the intensity of the received light.

State-of-the-art radiosity-style rendering algorithms partition the polygons of the scene into somewhat arbitrary regions and compute light transmission between pairs of regions. Since it is assumed that the intensity of light varies continuously over each region, the images generated by these algorithms are erroneous when the edges of the regions do not embed the shadow boundaries. A standard technique to avoid this problem is to recursively subdivide regions over which there is a sharp change in intensity. However, this leads to a very large number of very small regions near shadow boundaries.

Another problem that arises in computer graphics is to determine the intensity of light received by a point on a surface in the scene. This is usually done by computing the portions of the light source that are visible from the point and evaluating a particular integral over these portions. With this approach, the vast majority of time is spent computing the structure of the visible light source, as seen from the point.

Since the σ–aspect graph partitions the faces of the polyhedra into regions such that the structure of the visible light source σ is constant within each region, it is sufficient to compute the structure once for any region of the σ–aspect graph in order to efficiently answer intensity queries for points within that region. The structure computation can be done incrementally at a cost of $\mathcal{O}(\log n)$ per region in a scene of size n, as will be described at the end of Section 5. This technique results in a considerable speed increase over existing algorithms that answer intensity queries.

The *umbra* and *penumbra* are regions in space from which the light source is completely obscured and partially obscured, respectively. Previous work in computer graphics has concentrated on computing the umbral and penumbral boundaries and incorporating these boundaries in the partition of the scene. This is not sufficient since a penumbra region will usually contain discontinuities that require further subdivision of the region. Chin and Feiner compute the umbral and penumral boundaries for point and area light sources [CF89, CF92] using binary space partitioning trees. Campbell and Fussell [CF91], Lischinski, Tampieri, and Greenberg [LTG92], and others compute a subset of the shadow boundaries which includes the boundaries of the umbra and penumbra regions. The most complete algorithm to date is that of Heckbert [Hec92], which will be described later.

The problem we address is closely related to the computation of aspect graphs, as studied in computer vision [GCS91]. The aspect graph represents all possible views of a scene by partitioning the viewpoint space into regions such that the structure of the scene is the same for each point within a given region.

2 Discontinuity Surfaces

A *scene* is a set of polyhedra $\{\Phi_1, \Phi_2, \ldots, \Phi_m\}$ in E^3 and a polygon σ of bounded size which is considered to be a light source. There are a total of n vertices among the polyhedra. Each vertex has bounded degree. Every pair of features (vertices, edges, or faces) that are not topologically adjacent are separated by a distance greater than some constant α.

In a given scene, a *discontinuity surface* $<e_a, e_b, e_c>$ is the locus of points w such that, for some point s on σ, the interior of the line segment (w, s) intersects edges e_a, e_b, and e_c and nothing else. The surface $<e_a, e_b, e_c>$ is a ruled surface [Som34]. All directed segments (w, s) that intersect e_a,

e_b, and e_c do so in the same order. (Otherwise, the intersection points of two of the edges would be coincident for some such segment, and hence for all such segments.) The edge whose intersection is most distant from σ is said to *generate* the discontinuity surface.

If two of the edges intersect at a vertex v the discontinuity surface is denoted $<e, v>$, where e is the remaining edge. All directed segments that intersect e and v do so in the same order. The one of e and v whose intersection is most distant from σ is said to generate the discontinuity surface.

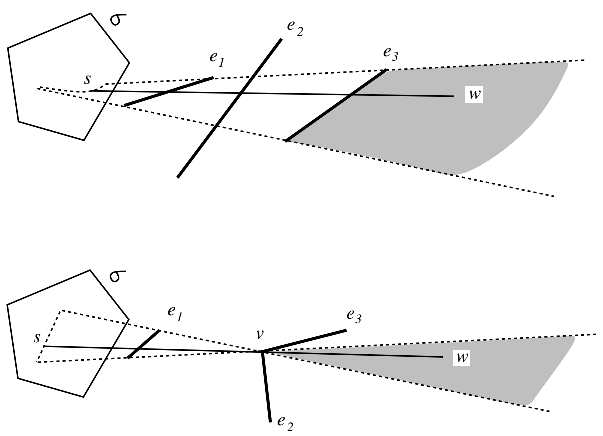

Figure 1: Surfaces $<e_1, e_2, e_3>$ and $<e_1, v>$ (shaded)

For example, see Figure 2. As a viewpoint moves from a to b on face f, it crosses the surface $<e, v>$ and a triangular region of σ becomes visible. Before crossing the surface the intensity of light received from the light source σ is zero. After crossing the surface the intensity increases approximately quadratically because the visible area of σ increases quadratically. The intensity of light received along the segment (a, b) is shown in the graph. At the intersection of (a, b) and $<e, v>$ a discontinuity in the second derivative occurs, hence $<e, v>$ is called at D^2 discontinuity surface.

If two edges in the scene are parallel and the plane defined by them intersects σ a D^1 discontinuity surface is defined (across which the illumination function has a discontinuity in its first derivative). For simplicity of exposition, we ignore this case in the following presentation; only minor modifications of the algorithm are needed to handle it.

3 The Problem

A *discontinuity curve* is the intersection of a discontinuity surface with one of the faces of the scene. For a given scene, the *σ–aspect graph* consists of, for every polyhedron face of the scene, the arrangement of all discontinuity curves on that face. The problem is to compute the σ–aspect graph for a given scene.

Heckbert [Hec92] considers a slightly different problem, that of computing the aspect graph with respect to every polygon in the scene, rather than a single polygon σ. The algorithm generates all potential *planar* discontinuity surfaces, each being defined by an edge-vertex pair in the scene. The algorithm computes the intersection of each such plane with the polygons of the scene and solves a two dimensional visibility problem to find the discontinuity curves on the plane. After all curves have been found, a plane sweep across each face suffices to compute the arrangement of the curves. The algorithm takes time $\mathcal{O}((n^3 + k) \log n)$ and does not consider the discontinuity surfaces defined by triples of edges ($k \in \mathcal{O}(n^4)$ is the complexity of the resulting aspect graph when only edge-vertex surfaces are considered).

Given a discontinuity surface $<e_a, e_b, e_c>$, the discontinuity curves that are the intersection of the surface with the faces of the scene can be found in time $\mathcal{O}(n \log n)$ as follows (this is similar to Heckbert's approach). Edges e_a, e_b, and e_c define a ruled surface which is the locus of all lines that simultaneously intersect the three edges. For each face of the scene, compute the curve which is the intersection of this surface and the face. Discard those curves which lie above the generating edge (i.e. on the same side of the generating edge as σ). This step takes time $\mathcal{O}(n \log n)$ since the intersection points are sorted along the curves. Among the curves that remain, determine which segments lie on $<e_a, e_b, e_c>$ by computing their upper envelope in time $\mathcal{O}(n \log n)$ with a sweep across the ruled surface. In a similar manner, the discontinuity curves from surfaces $<e, v>$ can also be computed in time $\mathcal{O}(n \log n)$. These procedures are said to *cast a surface*.

A naive solution to the σ–aspect graph problem is to consider every triple of edges in the scene. The surface defined by each triple is intersected with the polygons in the scene and a two dimensional visibility problem is solved on that surface. After all curves are found, each face is swept to compute the arrangement. This approach would take time $\mathcal{O}((n^4 + k) \log n)$, where $k \in \mathcal{O}(n^6)$ is the complexity of the σ–aspect graph.

In the following sections we present an algorithm whose running time is sensitive to the number of discontinuity surfaces.

4 Preliminaries

In a given scene, the *backprojection* $\mathcal{B} = (\mathcal{P}, \mathcal{L}, \mathcal{S})$ at a point v consists of a set $\mathcal{P}$ of polygons, a set $\mathcal{L}$ of ordered sets of edges, and a sphere $\mathcal{S}$ with center v and radius α. The polygons in $\mathcal{P}$ are the areas of σ visible from v, centrally projected onto $\mathcal{S}$. Each polygon is represented as a sequence of edges, and each edge stores the polyhedron edge whose projection it is. Since each polyhedron edge e_j can contribute up to n edges to $\mathcal{P}$ (on the same great circle of $\mathcal{S}$), an ordered set ℓ_j of these

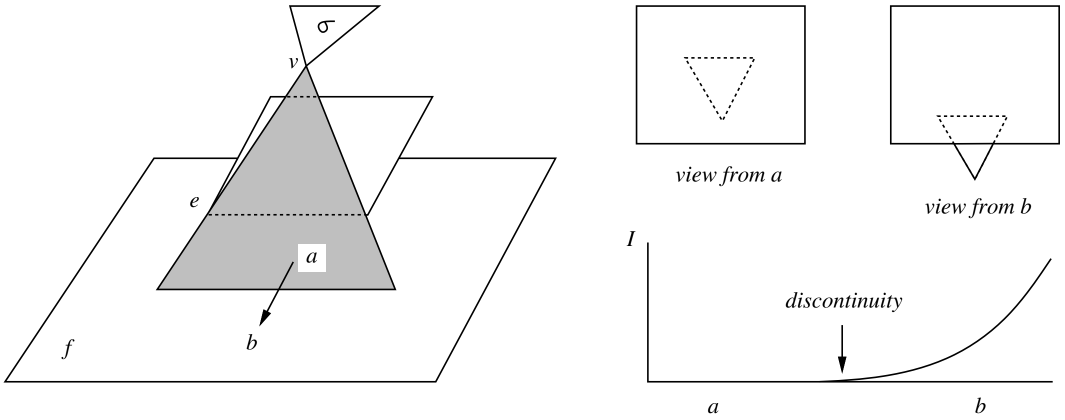

Figure 2: D^2 discontinuity curve

edges is maintained for each e_j. The set of ordered sets is $\mathcal{L}$.

For an edge e in $\mathcal{P}$, let $scene(e)$ be the segment of a scene edge whose projection is e and let $image(e)$ be the projection of e through v onto the other side of $\mathcal{S}$.

Lemma 1 *Let v be a vertex of polyhedron Φ, let $\mathcal{B} = (\mathcal{P}, \mathcal{L}, \mathcal{S})$ be the backprojection at v, and let e be an edge in $\mathcal{P}$. Then the discontinuity surface $<\mathrm{scene}(e), v>$ is nonempty if and only if some point of* image(e) *does not intersect Φ.*

Proof Any point w on the discontinuity surface $<scene(e), v>$ lies on a segment (w, s) that intersects v and *scene*(e) on its interior (where s is a point on σ). The line of (w, s) intersects $\mathcal{S}$ at some point w' of *image*(e). Since w is on the discontinuity surface, w does not intersect Φ and the ray from v through w (and w') lies on the exterior of the cone defined by the faces adjacent to v. It follows that the open segment (v, w') does not intersect any polyhedron, since the only polyhedron in the sphere $\mathcal{S}$ is Φ. Since (w', s) intersects only v and *scene*(e) on its interior, w' is a point on the discontinuity surface. Thus, if the discontinuity surface is nonempty there is some $w' \in$ *image*(e) that does not intersect Φ. It is clear that if there is some $w' \in$ *image*(e) that does not intersect Φ then the surface is nonempty. □

Given the backprojection at v, the d nonempty discontinuity surfaces generated by v can be enumerated in time $\mathcal{O}(n \log n + d)$, as follows. From Lemma 1, it is sufficient to enumerate the edges $e \in P$ for which *image*(e) is not wholly interior to Φ. Recall that $\ell_j \in \mathcal{L}$ is an ordered set of edges of $\mathcal{P}$ that lie on some great circle of $\mathcal{S}$. The great circle can be partitioned into open arcs $a_1 \ldots a_k$ whose images *image*(a_i) are alternately interior and exterior to Φ. If ℓ_j is stored as a balanced binary tree the endpoints in ℓ_j of an arc can be found in time $\mathcal{O}(\log n)$, and the edges of ℓ_j that lie on arcs whose image is exterior to Φ can be enumerated in time $\mathcal{O}(k \log n + d_j)$, where d_j is the number of such edges. Since a vertex has a bounded number of adjacent faces, k is bounded and all such edges in $\mathcal{P}$ can be enumerated in time $\mathcal{O}(n \log n + d)$.

In the following, a point is said to be *above* a face if it is in the halfspace containing the face's outward pointing normal.

Lemma 2 *Let e be a convex edge adjacent to faces f_i and f_j of a polyhedron Φ, let $\tilde{e}$ be an open segment of e that does not cross any discontinuity surface, let v be any point on $\tilde{e}$, and let $\mathcal{B} = (\mathcal{P}, \mathcal{L}, \mathcal{S})$ be the backprojection at v. Then for a vertex in $\mathcal{P}$ defined by the intersection of two edges, $e_a \cap e_b$, the discontinuity surface $<\tilde{e}$,* scene(e_a), scene(e_b)*$>$ is nonempty if and only if $e_a \cap e_b$ is above one of f_i and f_j and below the other.*

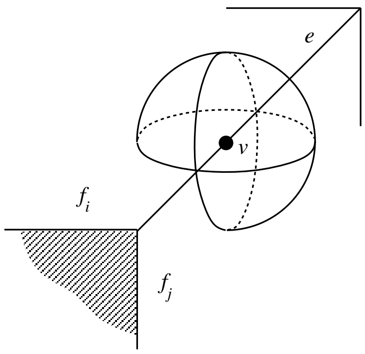

Figure 3: Quadrants defined by f_i and f_j

Proof Let (v, s) be the line segment that intersects on its interior exactly *scene*(e_a) and *scene*(e_b) for some point s on σ. The discontinuity surface is nonempty if and only if there exists some v on $\tilde{e}$ such that (v, s) can be extended to include v on its interior without intersecting any polyhedron. Since $\tilde{e}$ is an open segment that does not cross any discontinuity surface, a point defined by $e_a \cap e_b$ is present in the backprojection of every point on $\tilde{e}$ and has the same relation to f_i and f_j at every point. Thus the condition of the lemma is true of either all points on $\tilde{e}$ or no points on $\tilde{e}$. If $\tilde{e}$ is nonconvex or $e_a \cap e_b$ is above both f_i and f_j then any extension of (v, s) would intersect the interior of Φ (note that $e_a \cap e_b$ cannot be below both f_i and f_j, since this region is not visible from v). Otherwise, the quadrant opposite $e_a \cap e_b$ is empty within $\mathcal{S}$ and (v, s) can be extended. Thus there is an extension at every point along $\tilde{e}$ if and only if $e_a \cap e_b$ is above one of f_i and f_j and below the other. $\square$

Given the backprojection on $\tilde{e}$, the d nonempty discontinuity surfaces generated by $\tilde{e}$ can be enumerated in time $\mathcal{O}(n \log n + d)$ as follows. From Lemma 2 it is sufficient to enumerate those vertices in $\mathcal{P}$ that lie above one of f_i and f_j and below the other. Each $\ell_j \in \mathcal{L}$ corresponds to a great circle of $\mathcal{S}$ which can be divided into four arcs by the planes of f_i and f_j. With the same approach as was discussed after Lemma 1, the vertices of ℓ_j that lie on the two arcs above one of f_i and f_j and below the other can be enumerated in time $\mathcal{O}(n \log n + d)$ (duplicates can be avoided by assigning a vertex to exactly one of the two great circles on which it lies).

5 The Algorithm

The algorithm uses a volume sweep to compute the σ–aspect graph. A plane π, parallel to σ, is swept from σ in the direction of σ's normal. This direction is defined as *down*. The sweep plane stores a set of polygons which is the intersection of π with the boundaries of the polyhedra of the scene. Edges λ_i in π correspond to faces f_i in the scene and vertices μ_j in π correspond to edges e_i in the scene. For each vertex in π the backprojection is maintained. For each edge λ_i in π, the cross section of the aspect graph of f_i is maintained. As π sweeps through the scene, λ_i sweeps across f_i and constructs the aspect graph using a standard plane sweep.

The structure of π is updated as *events* occur corresponding to the intersection of π with certain points in space. Events are stored in a priority queue in order of increasing distance from the plane of σ. While events remain in the priority queue, the next event is removed from the queue and processed, possibly causing new events to be added to the queue. Initially, π lies in the plane of σ, the structure on π is empty (we will assume that no polyhedron intersects the plane of σ) , and the priority queue contains the vertices of the scene.

For some events, discontinuity surfaces will be cast which result in new discontinuity curves on the faces of the scene below π. Each such curve is stored in a list associated with the face and is processed when reached by π. If a discontinuity curve intersects the edge of a face an *edge event* corresponding to the intersection point is inserted into the priority queue.

An *edge event* occurs when a vertex μ of π (corresponding to some edge e of the scene) crosses a discontinuity surface. At this point, the backprojection $\mathcal{B} = (\mathcal{P}, \mathcal{L}, \mathcal{S})$ at μ changes with the addition or removal of up to three vertices and edges. For an example, see Figure 4, in which the visible area of σ is shaded. Given the three scene edges which define the discontinuity surface $<e_a, e_b, e_c>$, binary searches among the edges of the corresponding ℓ_a, ℓ_b, and ℓ_c in $\mathcal{L}$ determine which features of the backprojection need to be changed. Since the number of added or removed features is bounded, the update takes time $\mathcal{O}(\log n)$. Each new vertex in $\mathcal{B}$ corresponds to a pair of polyhedron edges, say e_a and e_b. Since these edges are visible from v, they define a discontinuity surface $<e, e_a, e_b>$ which is cast as discussed in Section 4.

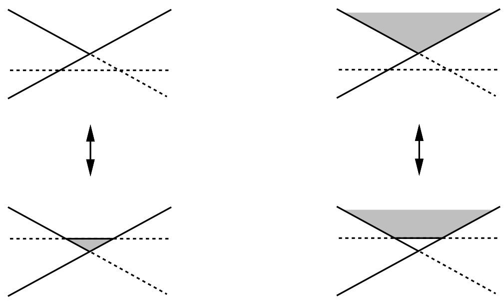

Figure 4: σ seen from either side of a surface

A *face event* occurs when π crosses a vertex v in the aspect graph of a face f_i. This corresponds to the sweep line λ_i crossing v and can be treated in the usual manner of the plane sweep in time $\mathcal{O}(\log n)$.

Define a *peak vertex* of a polyhedron as one for which all adjacent edges point down when directed outward from the vertex. A *peak vertex event* occurs when π crosses a peak vertex v. After π passes v, at least one polygon must be added to the structure of π, corresponding to the intersection of π with the faces adjacent to v. Since the polygons of π may be nested (although they do not intersect), the new polygons can be located in time $\mathcal{O}(n)$ by finding the innermost polygon containing v. At v and at each vertex μ_i of the new polygons in π, the backprojection is computed in time $\mathcal{O}(n^2 \log n)$ with a naive hidden surface removal algorithm that uses a plane sweep. (More sophisticated algorithms could achieve a lower expected cost. For example, see Mulmuley [Mul89].) For each backprojection, discontinuity surfaces are cast as described in Section 4. On each edge λ_j of the new polygons in π (corresponding to face f_j adjacent to v) the cross section of f_j's aspect graph is initialized.

A *normal vertex event* occurs when π crosses a nonpeak vertex v on a polyhedron Φ. Let $f_1 \ldots f_k$ be the faces adjacent to v, ordered counterclockwise around v as seen from the outside of Φ. Let $e_1 \ldots e_k$ be the edges adjacent to v: $e_i = f_i \cap f_{i+1}$. For each edge e_i above v, the backprojection $\mathcal{B}_i$

is known (at a point $\mu_i = e_i \cap \pi$ arbitrarily close to v). The following steps are performed in order.

1. For each edge e_i above v, consider e_i's backprojection $\mathcal{B}_i$ to be the backprojection at v and enumerate the discontinuity surfaces generated by v as discussed in Section 4. Eliminate duplicates by storing the surface names $<e, v>$ in a dictionary. The surfaces thus enumerated are exactly the surfaces generated by v, since the part of the source visible from v is the union of the parts visible from the points μ_i. Cast the surfaces as discussed in Section 4.

2. For each edge e_i below v do the following. For each backprojection $\mathcal{B}_j$ above v, determine the vertices of $\mathcal{B}_j$ that are above one of f_i and f_{i+1} and below the other (as discussed in Section 4). These vertices $e_a \cap e_b$ are exactly those with which e_i generates a discontinuity surface $<e_i, e_a, e_b>$. Let Υ_i be the union of these vertices. For each vertex $e_a \cap e_b$ in Υ_i, cast the the surface $<e_i, e_a, e_b>$. See Figure 5, in which $\Upsilon_i = \{u_3 \dots u_7\}$ and $\Upsilon_{i-1} = \{u_1 \dots u_4\}$.

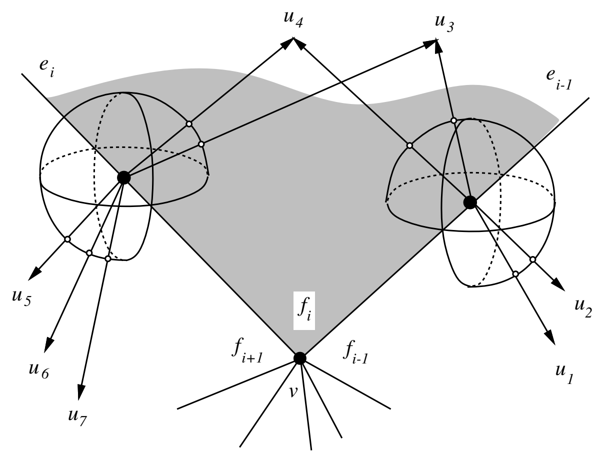

Figure 5: Backprojection on adjacent edges of a face

3. The backprojection on each edge e_i below v is computed from the backprojection on e_{i-1} as follows. For simplicity, assume that no discontinuity curve of f_i intersects v. Let Σ_{i-1} be the vertices of Υ_{i-1} that are below f_i, sorted radially around e_{i-1} in counterclockwise order as seen from v. Let Σ_i be the vertices of Υ_i that are below f_i, sorted radially around e_i in counterclockwise order. In Figure 5, $\Sigma_i = <u_5, u_6, u_7>$ and $\Sigma_{i-1} = <u_1, u_2>$. Let $\mathcal{B} = \mathcal{B}_{i-1}$. For each vertex $e_a \cap e_b \in \Sigma_{i-1}$ in order, update $\mathcal{B}$ as though the viewpoint of $\mathcal{B}$ crossed the discontinuity surface $<e_{i-1}, e_a, e_b>$. Then, for each vertex $e_a \cap e_b \in \Sigma_i$ in order, update $\mathcal{B}$ as though the viewpoint of $\mathcal{B}$ crossed the discontinuity surface $<e_i, e_a, e_b>$. After this procedure, $\mathcal{B} = \mathcal{B}_i$. The various backprojections $\mathcal{B}_i$ below v are stored in a persistent structure that costs log time and space per update (somewhat similar to the persistent balanced tree of Sarnak and Tarjan [ST86]).

4. After π passes v, the polygons of π are updated by the addition and removal of a bounded number of vertices and edges. For each face f_i strictly below v, λ_i is initialized with the cross section of f_i's aspect graph.

Recall that the number of edges and faces adjacent to v is bounded. Step 1 takes time $\mathcal{O}((n + d_v) \log n)$ to enumerate the d_v surfaces generated by v. For each edge e_i, Step 2 takes time $\mathcal{O}((n + d_i) \log n)$ to enumerate the d_i surfaces generated by e_i. The total cost for Step 3 is $\log n$ times the number of discontinuity surfaces generated by the edges at v since each much be crossed in computing the backprojections $\mathcal{B}_i$ below v and each must be sorted. Thus the cost of a nonpeak vertex event is $\mathcal{O}(n \log n)$ plus $\mathcal{O}(n \log n)$ for each discontinuity surface that is cast. In practice, the complexity of the backprojection is likely to be very small and the fixed cost per vertex should be less that $\mathcal{O}(n \log n)$.

Exclusive of the cost for casting surfaces, each peak vertex event takes time $\mathcal{O}(n^2 \log n)$ and each face and edge event takes time $\mathcal{O}(\log n)$.

Theorem 1 *The σ–aspect graph of a scene of complexity $\mathcal{O}(n)$ can be computed in time $\mathcal{O}((n^2 + pn^2 + dn + k) \log n)$, where p is the number of peak vertices, d is the number of discontinuity surfaces in the scene, and k is the complexity of the resulting σ–aspect graph.*

We expect p to be small in practice, so the running time should be dominated by the $dn \log n$ and $k \log n$ terms.

Section 1 described the problem of computing the intensity of light received by a query point on a surface of the scene. This problem can be efficiently solved if the structure of the visible light source (i.e. the backprojection) is known for the region of the σ–aspect graph that contains the query point. The backprojections for all regions can be efficiently computed with an algorithm of Gigus, Canny, and Seidel [GCS91] which takes advantage of the fact the adjacent regions in the aspect graph have backprojections that differ in a bounded number of features.

Their algorithm traverses the aspect graph and builds an interval tree which stores the "life span" of each backprojection feature during the traversal (i.e. the points during the traversal that the feature is created and destroyed). This tree uses space $\mathcal{O}(r)$ and takes time $\mathcal{O}(r \log r)$ to construct, where r is the size of one backprojection plus a constant times the number of regions. The tree can produce backprojection of a particular region in time linear in the size of the backprojection and logarithmic in the size of the tree.

6 Conclusion

We have introduced the problem of computing the σ–aspect graph for a set of polyhedra, which is of particular importance in accurately rendering polyhedral scenes. We have presented an output sensitive algorithm that solves the problem by sweeping a plane through the scene. Two key ideas used by the algorithm are that the backprojection at various points in the scene can be used to efficiently enumerate the discontinuity surfaces, and that the backprojections can be "carried along" with the sweep plane for very low cost. Furthermore, the use of backprojections avoids the enumeration and casting of all $\mathcal{O}(n^3)$ potential discontinuity surfaces, which would cost $\mathcal{O}(n^4 \log n)$ in the best case.

Acknowledgements

We would like to thank George Drettakis for his helpful insights and discussions. This research is supported by the Information Technology Research Centre of Ontario, the Natural Sciences and Engineering Research Council of Canada, and the University of Toronto.

References

[CF89] Norman Chin and Steven Feiner. Near real-time shadow generation using bsp trees. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):99–106, July 1989.

[CF91] A. T. Campbell III and Donald Fussell. An analytic approach to illumination with area light sources. Department of Computer Sciences, University of Texas at Austin, technical report TR-91-25, August 1991.

[CF92] Norman Chin and Steven Feiner. Fast object-precision shadow generation for area light sources using bsp trees. *1992 Symposium on Interactive 3D Graphics*, pages 21–30, March 1992.

[GCS91] Ziv Gigus, John Canny, and Raimund Seidel. Efficiently computing and representing aspect graphs fo polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):542–551, June 1991.

[Hec92] Paul Heckbert. Discontinuity meshing for radiosity. *Third Eurographics Workshop on Rendering*, pages 203–215, May 1992.

[LTG92] Dani Lischinski, Filippo Tampieri, and Donald Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics Graphics & Applications*, pages 25–39, November 1992.

[Mul89] Ketan Mulmuley. An efficient algorithm for hidden surface removal. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):379–388, July 1989.

[Som34] Duncan M. Y. Sommerville. *Analytical Geometry in three dimensions*. Cambridge University Press, 1934.

[ST86] Neil Sarnak and Robert Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, July 1986.