Fast Computation of Shadow Boundaries
Using Spatial Coherence and Backprojections

A. James Stewart and Sherif Ghali

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S 1A4
{jstewart,ghali}@dgp.toronto.edu

Abstract

This paper describes a fast, practical algorithm to compute the shadow boundaries in a polyhedral scene illuminated by a polygonal light source. The shadow boundaries divide the faces of the scene into regions such that the structure or "aspect" of the visible area of the light source is constant within each region. The paper also describes a fast, practical algorithm to compute the structure of the visible light source in each region. Both algorithms exploit spatial coherence and are the most efficient yet developed.

Given the structure of the visible light source in a region, queries of the form "What specific areas of the light source are visible?" can be answered almost instantly from any point in the region. This speeds up by several orders of magnitude the accurate computation of first level diffuse reflections due to an area light source. Furthermore, the shadow boundaries form a good initial decomposition of the scene for global illumination computations.

CR category: I.3.7 [Computer Graphics] Three-Dimensional Graphics and Realism

Key words: shadows, backprojections, discontinuity mesh, aspect graphs, radiosity, visual events, efficient surface enumeration, spatial coherence

1 Introduction

In a polyhedral scene illuminated by an area light source, discontinuities in illumination must be considered if the scene is to be accurately and quickly rendered. Taking these discontinuities into account goes a long way to producing realistic images.

Discontinuities in illumination occur where the appearance of the visible light source changes abruptly. For example, when a viewpoint crosses from a light area into a shadow, part of the light source disappears and the rate of change of the illumination at the viewpoint diminishes sharply.

Illumination discontinuities form planar or quadric surfaces in space, across which the appearance of the visible light source changes abruptly. When intersected with the faces of the scene, they form linear or quadratic discontinuity curves. The discontinuity curves divide the faces of the scene into cells in which the structure of the visible light source doesn't change. Since the structure of the visible source is constant in each cell, a symbolic description of the structure (called the "backprojection") can be stored in each cell; this allows almost instant reconstruction of the visible areas of the light source at any point in the cell. These concepts will be discussed in more detail later.

In some of the earliest work, Crow computed the shadow volumes made by objects and a point light source [7]. Nishita and Nakamae [18] used shadow volumes from an area light source and classified luminance on each face of the scene as being in umbra, in penumbra, or in light. Radiosity–style algorithms [6; 5] motivated much effort in computing shadows boundaries for use as an initial mesh in radiosity computations. Chin and Feiner [4] and Campbell and Fussell [1] used BSP trees to compute the shadow boundaries in a scene illuminated by a point light source.

Attention has recently turned to computing shadow boundaries for area light sources. Lischinski, Tampieri, and Greenberg [15] classified the types of shadow boundaries that can arise according to the discontinuity in the derivatives of the illumination function on a receiver straddling the boundary. They and Heckbert [14] implemented algorithms that compute the shadow boundaries in a scene illuminated by a polygonal light source. Both algorithms restrict their attention to planar discontinuity surfaces. Algorithms that compute both planar and quadric discontinuity surfaces were given by the authors of the present paper [22] and by Drettakis [8].

Of importance to discontinuity surfaces, Teller recently described how to determine the common stabber to a set of four lines in space using Plücker coordinates [24; 25]. Plücker space provides an elegant parameterization of lines in space that finds the relative orientation of two directed lines using a dot product [21; 23; 3]. We expect Plücker coordinates to become a popular tool in computer graphics.

The discontinuity mesh is related to the aspect graph which is studied in computer vision. Gigus and Malik [12] described an algorithm to compute the aspect graph for planar and quadric discontinuity surfaces. The best algorithm to date for the computation of the aspect graph is due to Gigus, Canny, and Seidel [11].

The results from the present paper can be used in several applications. The discontinuity mesh can be used as the initial mesh for a radiosity–style algorithm. The mesh algorithm can be called before each step of a progressive refinement radiosity using the emitting cell as the light source [15]. Multiple views of the same scene can be computed quickly given the discontinuity mesh, which is independent of the eyepoint and only needs to be computed once.

2 Backprojections

This section describes an algorithm that computes the backprojection in every cell of a discontinuity mesh. Construction of the mesh is described in later sections. The algorithm maintains a winged-edge structure to store adjacency information about faces of the scene. and takes advantage of spatial coherence (i.e. the adjacency of faces) to avoid unnecessary computation. The only other algorithm that performs similar processing of the mesh [8] considers each face of the scene separately, but could easily be modified to incorporate our approach. The original work in this area [12] computed the aspect graph (not the discontinuity mesh) and did not need to take advantage of spatial coherence.

2.1 Definition of the Backprojection

Recall that the visible source has the same structure or "aspect" from every viewpoint within a cell of the discontinuity mesh, although different areas of the source are visible from different viewpoints

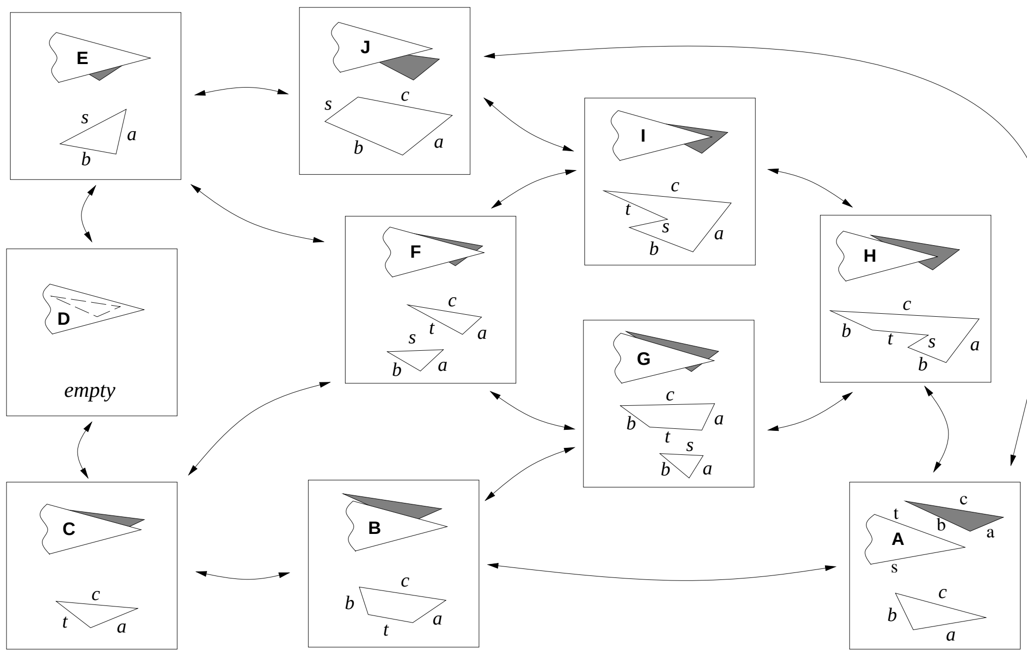

Figure 1: Visible Source and Backprojection in the Mesh Cells

within the cell. Only the position, not the structure, of the area of the visible source varies from viewpoint to viewpoint within a cell.

Figure 2 shows part of a discontinuity mesh formed by a triangular source and a triangular blocker. Each cell of the mesh is labeled with a capital letter. The source edges are a, b, and c; the scene edges of interest are s and t. In this example the only discontinuity surfaces that arise are defined by an edge and a vertex and are called EV surfaces. In general, a discontinuity surface is defined by three edges (this is discussed in Section 3).

One such EV surface is formed by vertex bc and edge t and is denoted $\langle b, c, t \rangle$. The discontinuity curve that is the intersection of $\langle b, c, t \rangle$ with the rectangular receiver separates cells I and H, F and G, and C and B in Figure 2.

Figure 1 shows the visible source for each cell of Figure 2. Each box in Figure 1 corresponds to one cell in Figure 2 and contains (i) the visible source in its upper part as it appears from a viewpoint in the cell, looking upward and (ii) the abstract structure of the visible source in its lower part. The arrows in Figure 1 join adjacent cells. Cell A shows the unoccluded source with labeled edges.

Define a point u to be *visible* from a point v if the line segment uv does not intersect the interior of any polyhedron in the scene. Define the *visible source* at a point v to be the set of points on the source that are visible from v. The visible source is a set of polygons and has edges defined by source edges or scene edges. This abstract structure is called the "backprojection" and was described by the authors in [22].

At a viewpoint v, define the *backprojection* to be a set of polygons whose edges (i) correspond one-to-one to edges of the visible source at v and (ii) are labeled with the corresponding edges of the scene that bound the visible source. In Figure 1 the top part of each box is the visible source and the bottom part is the backprojection.

Given the backprojection at a particular point, the polygonal boundary of the visible source can be computed as follows: for each backprojection edge, project the corresponding scene edge through the viewpoint onto the plane of the source. Then the exact illumination at the point can be calculated by integrating around the boundary of the visible source.

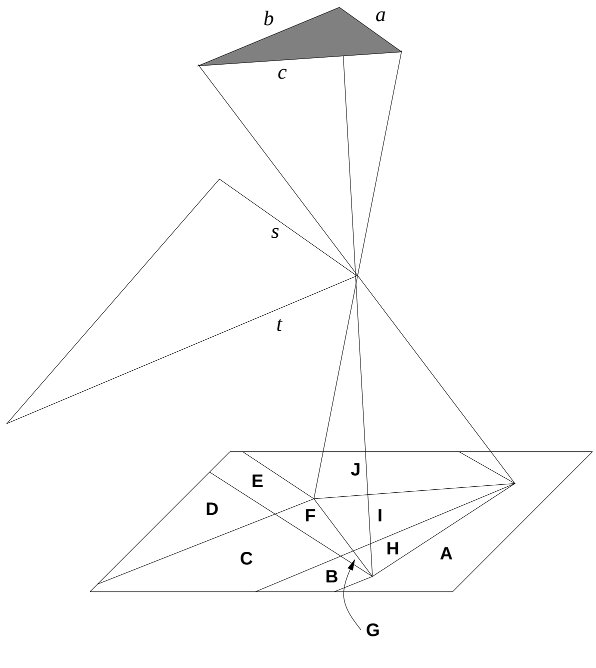

Figure 2: Discontinuity Mesh

2.2 Computing the Backprojections

This section describes how to efficiently compute the backprojection for every cell of a mesh. The key insight is the following: *Given the backprojection in one cell, the backprojection in an adjacent cell can be computed by adding or removing a few edges.* For example, in going from cell E to cell J of Figure 1 an edge c is added to the backprojection.

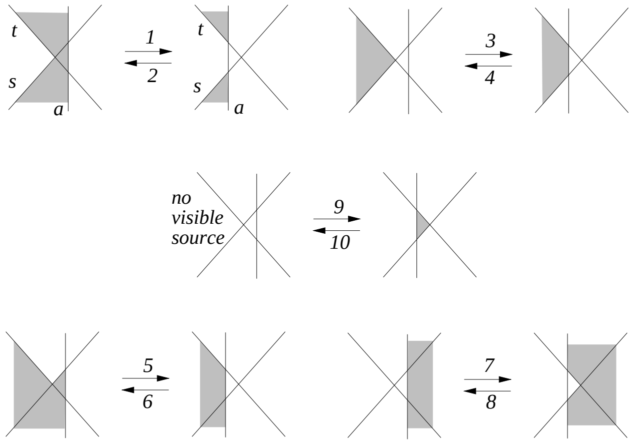

Figure 3: Transitions Across a Discontinuity Surface

To compute the backprojection in every cell of a mesh, the backprojection is computed in an initial cell, then propagated to all adjacent cells, then propagated to all cells adjacent to those cells, and so on. This is a breadth-first traversal of the discontinuity mesh.

To compute the backprojection in the initial cell, a visible surface algorithm (for example, that of McKenna [17]) must be used to determine which parts of the source are visible. However, our algorithm only performs this step at a very few vertices of each polyhedron in the scene.

2.3 Updating the Backprojection

This section describes how to update the backprojection from one cell to another. A *visual event* [12] occurs when the viewpoint crosses a discontinuity surface. Since the discontinuity surface intersects the source, some of the three edges that define the surface bound parts of the visible source, as seen from the viewpoint.

For example, consider the transition from cell I to cell F in Figure 2. As the viewpoint crosses the discontinuity surface $\langle a, s, t\rangle$, the vertex st (as seen from the viewpoint) appears to pierce the edge a, dividing the visible source into two parts (see Figure 1). The upper left diagram in Figure 3 shows this as an inversion of three lines, following the arrow labeled "1" (the edges s and t are extended past the st vertex and the visible source is shaded).

All transitions can be thought of in a unified manner as an inversion of three edges. This unified approach results in a simple implementation. There are only five symmetric pairs of transitions, as shown in Figure 3. All possible combinations of visible source were considered and only these five pairs were physically possible. Each configuration in Figure 3 consists of the three lines that embed the three edges defining the discontinuity surface. Given the three edges of a discontinuity surface and a backprojection on one side of that surface, the transition across the surface can be determined symbolically (with no numeric computation), making the determination robust in the presence of numerical inaccuracy.

Table 1 describes the conditions and actions associated with each transition. The transition is determined by the following two criteria which are listed in the table:

#E (number of edges) The three scene edges that define the discontinuity surface will appear in the backprojection some number of times (0,1,2,3, or 4).

#V (number of vertices) The scene edges that appear in the backprojection will intersect at vertices some number of times (0,1,2, or 3).

Table 1: Transition Conditions and Actions

#E	#V	type	action
0		9	None of a, b, or c are in the backprojection Create a cycle of a, b, and c. Order is determined numerically.
1		7	a is in the backprojection. b and c are not. Break a into $a_1 \rightarrow a_2$. Insert b and c between a_1 and a_2. Determine b and c order numerically.
2		3 or 6	$a \rightarrow b$. c is not in the backprojection. Insert c between a and b.
3	1	1	$a \rightarrow b$. c is separate. Split c into $c_1 \rightarrow c_2$. Join $a \rightarrow c_2$ and $c_1 \rightarrow b$.
3	2	4 or 5	$a \rightarrow b \rightarrow c$. Delete b.
3	3	10	$a \rightarrow b \rightarrow c$ in a cycle. Delete a, b, and c.
4	2	2	$a \rightarrow c_1$. $c_2 \rightarrow b$. Join $a \rightarrow b$. Merge c_1 and c_2 into c.
4	3	8	$a_1 \rightarrow b \rightarrow c \rightarrow a_2$. Delete b and c. Merge a_1 and a_2 into a.

For example, if four edges appear in the backprojection (one appears twice) and they intersect at two vertices then a Type 2 transition applies (shown in Table 1).

Since a scene edge can appear in the backprojection many times, we must be careful to count only those instances that participate in the backprojection update (i.e. those edges shown in Figure 3). From inspection of Figure 3, a backprojection edge e is only counted if either (i) it is attached to another backprojection edge that is labelled with one of the three scene edges, or (ii) it is not attached to any such edge and the two scene edges that do *not* label e meet at a vertex in the scene. Each edge in Figure 3 satisfies condition (i) except for the vertical edges in Type 1 and 7 transitions which satisfy condition (ii).

Table 1 also describes the actions taken for each type of transition. Each entry consists of a description of the configuration of backprojection edges before the transition and an action to perform. The edges that define the discontinuity surface are labeled a, b, and c. Where an edge appears more than once in the backprojection it is labeled with subscripts, like a_1 and a_2. In Table 1 the notation $x \rightarrow y$ means that edge x precedes edge y in the backprojection. Edges in the backprojection are directed such that the visible part of the source appears to the left of the edge.

For example, consider a Type 2 transition in Table 1. Before the transition the backprojection has four edges, two of which are generated by the same edge on the discontinuity surface (see Figure 3). Keeping in mind that the visible source is to the left of each edge, the edges can be labeled a, b, c_1, and c_2 in exactly one way such that $a \rightarrow c_1$ and $c_2 \rightarrow b$. Upon the transition, a and b are joined and c_1 and c_2 are merged into a single edge.

2.4 Computing the Backprojection Across an Edge

The algorithm of the previous section can propagate a backprojection between cells within a single face of the scene. However, when propagating between cells in *two adjacent faces* the backprojection must cross the edge that is shared by the two faces. Since an edge might be embedded in several discontinuity surfaces, the backprojection might have to traverse several surfaces.

For example, Figure 4 shows a source with edges a, b, and c and a polyhedron with two faces labeled F and G sharing an edge e. The edge e is embedded in two discontinuity surfaces:

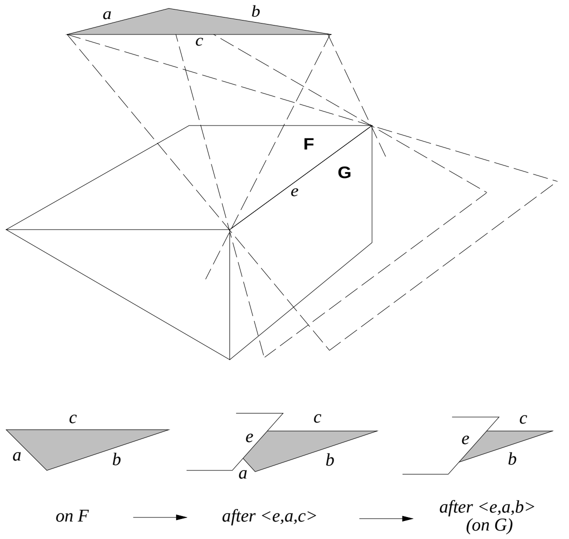

Figure 4: Backprojection Traversing an Edge

$\langle e, a, c\rangle$ and $\langle e, a, b\rangle$ (note that there is no surface $\langle e, b, c\rangle$ since it would intersect the interior of the polyhedron). In face F the backprojection consists of the whole source. In moving to the adjacent face G the backprojection first traverses the surface $\langle e, a, c\rangle$ below e (hiding the vertex ac), then traverses the surface $\langle e, a, b\rangle$ below e (hiding the vertex ab). The two transitions are shown at the bottom of Figure 4.

The backprojection must cross each discontinuity surface in radial order around the edge. In Figure 4 it would not have made sense to traverse the surface $\langle e, a, b\rangle$ before the surface $\langle e, a, c\rangle$.

In summary, a backprojection can be propagated from one cell of a discontinuity mesh to all connected cells for very little cost, even if cells are on separate faces of the same polyhedron. Drettakis [8] takes a different approach and computes a new backprojection on each polyhedron face.

3 Casting Discontinuity Surfaces

This section describes how to *cast* a discontinuity surface through the scene. The casting operation computes the discontinuity curves that are formed by the intersection of a discontinuity surface with the faces of the scene. In this section it is assumed that the three edges defining the surface have already been discovered. Section 4 will describe how to efficiently discover the triples of edges that define discontinuity surfaces.

A discontinuity surface is defined by a triple of edges. In some cases, two of the three edges might meet at a vertex. A surface defined by three edges is denoted EEE; a surface defined by an edge and a vertex is denoted EV. Figure 5 shows two such shaded surfaces.

A *discontinuity surface* is a set of points in space. For any point w on the surface there is a ray that intersects three edges e_1, e_2 and e_3 and the light source at some point s without intersecting the interior of any polyhedron in the scene. If the three edges are skew and distinct, the surface generated is a subset of a quadric [19; 12; 24]. If two of the three edges intersect in a point, the surface is a subset of a plane.

A *discontinuity curve* is the intersection of a discontinuity surface with a face in the scene. If the surface is a quadric, the resulting curve is a conic. If the surface is a plane, the curve is a line segment. For a viewpoint on a face, the appearance of the visible source changes as the viewpoint crosses the discontinuity curve.

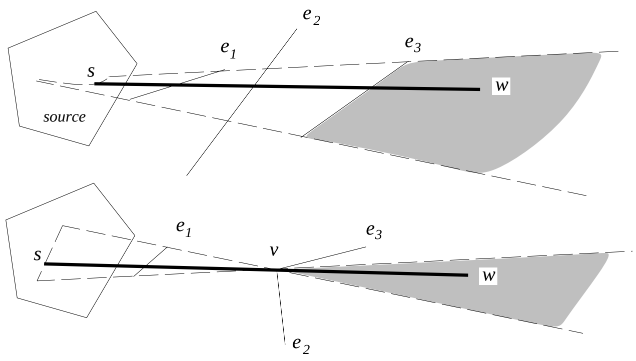

Figure 5: EEE and EV Discontinuity Surfaces

A *discontinuity mesh* is the arrangement of discontinuity curves on all faces of the scene.

Heckbert [14] considers only EV surfaces. For every potential EV surface in a scene he computes the intersection of the surface with every face of the scene, resulting in a set of discontinuity segments on various faces. Those segments which are not visible from the light source are removed with a two dimensional visibility algorithm. The remaining segments form the discontinuity mesh. The following sections generalize Heckbert's approach to handle EEE as well as EV surfaces.

3.1 Quadrics Defined by Three Edges

Given a point and two lines in space, there exists a single line that touches both lines and pass through the point. The ruled quadric, or *regulus*, formed by the set of lines simultaneously touching three lines in space can be easily synthesized if one thinks of a point moving on one of the three lines. There exists a single line (called the *moveable line* [19]) that passes by each point position and touches the two other lines. The collection of all such lines forms a regulus. This is the basis of a simple parameterization described below. Note that, in general, any line in space pierces such a regulus in *two* points.

The three lines that define the regulus are called *generators* and any line simultaneously touching all three lines is called a *directrix*. The resulting regulus is doubly ruled and if the set of directrices is called the regulus, the set of generators is called the *opposite regulus* [20].

The intersection of a quadric with a plane is, in general, a conic. In affine space, this conic can be an ellipse, a parabola, or a hyperbola depending on whether the line at infinity intersects the conic in zero, one, or two real points in projective space [26]. The section of a conic that arises in our problem is a hyperbola, though it can degenerate into a line segment. We perform all computations using homogeneous coordinates and avoid the need to classify the type of conic in each case.

3.2 Computing Surface/Face Intersections

A key operation is to compute the intersection of a discontinuity surface and the scene. First, the equation of the quadric surface is computed. Then this quadric is intersected with each face of the scene to form a conic in each face. The segments of the conic that lie in the face are determined by intersecting the quadric with each edge of the boundary of the face. For each edge of the boundary, this is equivalent to finding the two directrices that simultaneously intersect the three generators of the surface and the line that carries the edge. By substituting the parametric equation of each polygon edge into the equation of a quadric ([19; 12]), we get an equation of the second degree. The real solutions to this equation give the intersection points and the directrices at those points. Each intersection point can be specified by two parameters: the index of the directrix that intersects the point (i.e. the position of the directrix on one of the generators) and the distance along the directrix from the point to the source.

3.3 Computing the Visible Conic Segments

Before adding a discontinuity curve to a face, it must be determined that the curve is visible from the light source. Otherwise, the curve is not added to the face. The distances and the indices computed for the endpoints of discontinuity segments are used as input to a sweep line visibility algorithm. This is a simplified version of the hidden surface removal problem (see [14; 8] for details). This approach works whether the quadric is a paraboloid of one sheet or a hyperbolic paraboloid and works whether the conic is a hyperbola or a (subset of a) line.

3.4 Computing the Mesh in a Face

After all surfaces have been cast, each face has a list of discontinuity curves associated with it. To this list are added the edges of the face. The arrangement of curves (and the cells defined by the curves) can be computed using a plane sweep algorithm [10]. This is similar to the arrangement Heckbert considers [14]. In our case, we need to make sure, however, that all curves are monotonic in the direction of the sweep. This is clearly true for line segments. For conic segments, check to see if the segment is monotonic and, if it is not, break it into two segments at the point with a derivative parallel to the sweep line.

4 Efficient Surface Enumeration

A naive algorithm to build the discontinuity mesh would test *every* triple of edges as a potential discontinuity surface. This section presents a much better algorithm that efficiently determines which triples of edges form discontinuity surfaces. That is, it efficiently enumerates the surfaces.

For an environment of n edges the naive approach considers $\binom{n}{3}$, or about $1/6\ n^3$, triples of edges. For a relatively small environment of 10,000 edges the naive algorithm must consider about 167,000,000,000 such triples. Even if each takes only 0.01 milliseconds to test for validity (about 40 floating point operations at 4 megaflops), it would take 19 days just to enumerate the valid surfaces.

Drettakis [8] uses voxel grid to reduce the number of triples considered, as follows: for each edge e, draw the smallest cone that encloses the edge and the source. Only consider those other edges that lie in the voxels that intersect the cone. For each *pair* of such edges, see if it forms a discontinuity surface with the original edge e. Drettakis reports that nine tenths of edges are eliminated this way.

Since the number of surfaces is usually much less than $\binom{n}{3}$ an efficient algorithm should not consider a triple of edges unless it actually defines a discontinuity surface. The algorithm presented below uses the backprojection and spatial coherence to accomplish this.

4.1 Backprojections and Discontinuity Surfaces

Each discontinuity surface is defined by a triple of edges, two of which might happen to intersect at a vertex. If one of these edges is the most distant from the source it is called the *generating edge*. If two edges intersect at a vertex that is most distant from the source it is called the *generating vertex*. Given three edges, the most distant can be determined by drawing a line that intersects the three edges and the source and ordering the intersection points by increasing distance from the source.

In the following discussion, the light source is assumed to be higher than every polyhedron in the scene. This is not a restriction since nothing above the source (i.e. behind it) is directly illuminated

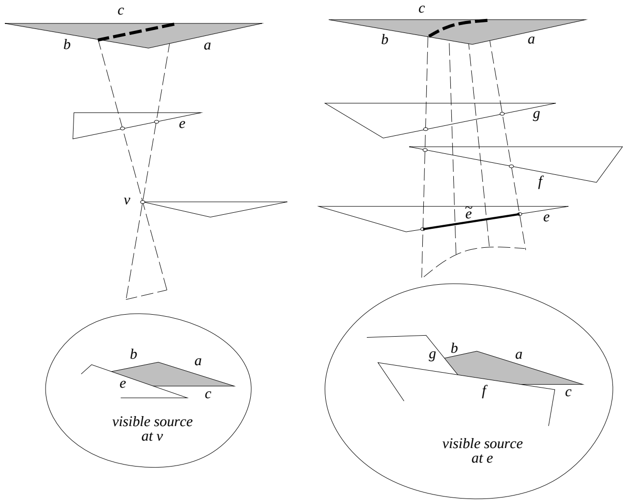

Figure 6: Generating Vertex v and Generating Edge $\tilde{e}$

and can be ignored. *The following observations about generating vertices and generating edges form the core of our approach and allow us to exploit spatial coherence.*

4.1.1 Generating Vertices

If v is the generating vertex of some EV surface $\langle e, v\rangle$ then from the definition of a discontinuity surface there must be a line ℓ that simultaneously intersects v, e, and the source without intersecting the interior of any polyhedron between v and the source. Furthermore, this line must extend below v into empty space; otherwise it would extend into the interior of a polyhedron and the surface would stop at v.

For example, consider the left side of Figure 6 in which a vertex v, an edge e, and a source abc are identified. The backprojection at v is shown to contain four edges: e, c, a and b. There is a line through v and e that intersects the source without intersecting the interior of any polyhedron, so v is the generating vertex of the discontinuity surface $\langle e, v\rangle$. Similarly, v generates the surfaces $\langle a, v\rangle$, $\langle b, v\rangle$, and $\langle c, v\rangle$.

As shown from the example (and proven in our previous work [22]) the vertex v generates a discontinuity surface with exactly those edges that appear in the backprojection at v.

Observation 1 *A vertex v generates a surface $\langle e, v\rangle$ if and only if e is an edge in the backprojection at v and some line intersecting v, e, and the source extends below v into empty space.*

4.1.2 Generating Edges

On an edge e, consider an open subsegment $\tilde{e}$ that does not cross any discontinuity surface (which means that the backprojection is the same at every point on $\tilde{e}$). If $\tilde{e}$ generates a discontinuity surface $\langle \tilde{e}, f, g\rangle$ (where f and g are two edges) then by the definition of a discontinuity surface there must be a line ℓ that simultaneously intersects $\tilde{e}$, f, g, and the source without intersecting the interior of any polyhedron between $\tilde{e}$ and the source. This line must also extend below $\tilde{e}$ into empty space.

For example, consider the right side of Figure 6 in which three edges e, f, and g, and a source abc are shown. The backprojection on $\tilde{e}$ is a polygon with edges a, b, g, f, and c. Since there is a line simultaneously touching $\tilde{e}$, f, g, and the source, $\langle \tilde{e}, f, g\rangle$ is a discontinuity surface.

By a similar argument as for generating vertices, whenever a pair of edges x and y meet in the backprojection on $\tilde{e}$ a line can be drawn that simultaneously touches $\tilde{e}$, x, y, and the source, since the edges of the backprojection define the visible source. Thus $\langle \tilde{e}, g, b\rangle$, $\langle \tilde{e}, b, a\rangle$, $\langle \tilde{e}, a, c\rangle$, and $\langle e, c, f\rangle$ are also discontinuity surfaces generated by $\tilde{e}$ in Figure 6.

Observation 2 *A subsegment $\tilde{e}$ of an edge e generates a surface $\langle \tilde{e}, x, y\rangle$ if and only if edges x and y meet at a vertex in the backprojection on $\tilde{e}$ and some line intersecting $\tilde{e}$, x, y, and the source extends below $\tilde{e}$ into empty space. Although x and y meet in the backprojection, the corresponding scene edges do not necessarily meet at a vertex.*

In summary, if the backprojection is known for each vertex and along each edge of the scene, the discontinuity surfaces can be efficiently enumerated simply by looking at the backprojections: At a vertex, all edges in the backprojection are tested; on a subsegment of an edge, all vertices in the backprojection are tested. Since the size of the backprojection is usually very small this testing will be fast. Thus, all discontinuity surfaces can be enumerated efficiently in time approximately proportional to the number of surfaces.

4.2 Propagating Backprojections Locally

The algorithm presented below computes the backprojections at a few selected vertices and then propagates them along the edges of the scene to every other edge and vertex. Thus backprojections will be known everywhere and all discontinuity surfaces can be enumerated efficiently. The following sections describe how to propagate the backprojection through the scene.

4.2.1 Propagating Along an Edge

An edge e can be subdivided into maximal subsegments in which the backprojection is constant. By definition, these subsegments will be separated by discontinuity surfaces which intersect e but which don't embed e. The points of intersection can be ordered from one end of e to the other. Let $s_1, s_2, \ldots s_k$ be the maximal subsegments on e, in order.

Given the backprojection at one end of the edge (in s_1) it is propagated to the other end (in s_k) by crossing the discontinuity surfaces in order. The actions of Table 1 are applied to update the backprojection with each crossing.

To enumerate the discontinuity surfaces generated by e, Observation 2 is applied. Let B_i be the backprojection in s_i. Every vertex in B_1 (the backprojection of the first subsegment) is combined with e according to Observation 2 and the resulting surfaces are cast. Upon updating the backprojection from B_i to B_{i+1} only the *new vertices* in B_{i+1} are combined with e and the resulting surfaces are cast (the old vertices were considered at a previous stage).

Thus the backprojection can be propagated down an edge and all surfaces generated by that edge can be efficiently and easily enumerated.

4.2.2 Computing the Backprojection at a Vertex

An edge (u, v) adjacent to a vertex v is said to be *above* the vertex if the other endpoint u is higher than v. It will be assumed that u and v are never at the same level (this can be accomplished by imposing a total order on the vertices).

The backprojection at the vertex is the union of the backprojections on the edges immediately above the vertex: If some part of the source (which is above the vertex) is visible at the vertex then it is visible on one of the edges immediately above the vertex. Conversely, if some part of the source is visible on an edge immediately above the vertex then it is visible at the vertex.

To enumerate the surfaces generated by v it is sufficient from Observation 1 to determine which edges are in the backprojection at v (the backprojection does not actually need to be constructed). These are exactly the edges in the union of the backprojections above v, minus the set of edges adjacent to v. This set can be easily computed and the corresponding surfaces cast.

Thus all surfaces generated by a vertex can be efficiently enumerated, given the backprojections on the edges above the vertex.

4.2.3 Propagating Past a Vertex

At a vertex the backprojection is propagated from edge to adjacent edge in *radial order* around the vertex. In this manner the backprojection is propagated from edges above the vertex to edges below.

Let e_i and e_{i+1} be two adjacent edges at a vertex v. Let f_i be the face between them and let f_{i-1} be the face on the other side of e_i. The propagation from e_i to e_{i+1} consists of three steps: from e_i to f_i, across f_i, and from f_i to e_{i+1}. See Figure 7, which shows a cutaway view of the faces.

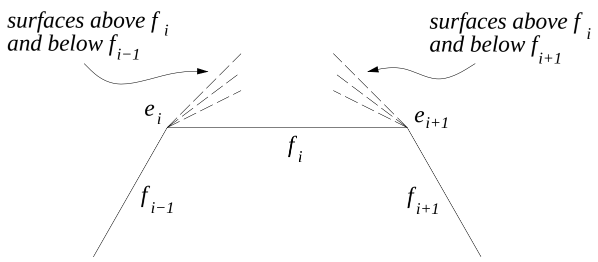

Figure 7: Propagating from Edge to Edge

The first step is similar to the propagation across an edge described in Section 2.4: the backprojection on e_i must be incrementally updated across the surfaces that embed e_i until it reaches f_i (see also Figure 4). To determine these surfaces, consider every vertex in the backprojection at v (which is the union of the backprojections on edges above v): Draw a ray from v through the two scene edges that define the backprojection vertex. Order these rays radially around e_i and traverse the corresponding surfaces in radial order.

However, in propagating the backprojection from e_i to f_i, it is incorrect to traverse all of the surfaces embedding e_i. Only those surfaces that fall below f_{i-1} and above f_i should be traversed (i.e. those surfaces for which the corresponding ray leaving v has a positive dot product with the outward pointing normal of f_{i-1} and a negative dot product with the normal of f_i). These are the surfaces encountered as the backprojection moves from e_i to f_i. The other surfaces which do not satisfy this condition are those encountered moving from e_i to f_{i-1}.

For example, in Figure 4 the surfaces $\langle e, a, b\rangle$ and $\langle e, a, c\rangle$ satisfy this condition (where $F = f_{i-1}$ and $G = f_i$). In propagating the backprojection from e to G both surfaces would be traversed. But in propagating from e to F neither would be crossed. One can think of the propagation from face F to face G in two stages: from F to e and from e to G.

The second step involves propagating the backprojection across f_i, since the backprojection on f_i adjacent to e_i could be different from that on f_i adjacent to e_{i+1}. This only occurs if there are discontinuity curves in f_i which have one endpoint on v (recall that e_i and e_{i+1} meet at v). These curves must be traversed in radial order around v, from e_i to e_{i+1}. Note that these curves will be known when this step takes place.

The third step involves propagating the backprojection from f_i to e_{i+1}. This is the same as the first step, but only the surfaces above f_i and below f_{i+1} are traversed.

Thus the backprojection can be propagated from any edge above a vertex to every edge below the vertex. The propagation is fairly easy to implement.

4.3 The Global Propagation Algorithm

Define a *peak vertex* to be a vertex with no adjacent edges above it. Since every vertex and edge is reachable from some peak vertex, it is sufficient to compute backprojections for all peak vertices and to propagate them as described above.

However, when propagating along an edge e the discontinuity surfaces that intersect e must be known. The generating edge of any such discontinuity surface must lie above e, since it lies between e and the source, which lies above everything. If the edges are processed from top to bottom then the generating edges will be encountered first, ensuring that all surfaces that intersect e are known before e is reached.

The global propagation algorithm operates as follows: Initally, all scene vertices are placed in a priority queue ordered by decreasing height. While the queue is not empty the highest vertex is removed and one of the following operations is applied:

- At a peak vertex, the backprojection is computed with a visible surface algorithm. The backprojection is also computed for one of the edges below the vertex and is propagated to the other edges as described in Section 4.2.3. The surfaces generated by the vertex and its adjacent edge are cast through the scene. Whenever a surface intersects an edge of the scene, the surface/edge intersection point is added to the priority queue.
- At a vertex that is not a peak, the backprojection is propagated from the edges above the vertex to the edges below as described in Section 4.2.3. The surfaces generated by the vertex and the edges below the vertex are cast through the scene, as described in Section 3. Whenever a surface intersects an edge of the scene, the surface/edge intersection point is added to the priority queue.
- At a surface/edge intersection point, the backprojection on the edge is propagated across the discontinuity surface that defines the point. Surfaces are enumerated as described in Section 4.2.1 and are cast through the scene as usual.

When the priority queue becomes empty, all surfaces have been cast and each face of the scene stores an unordered set of discontinuity curves that lie in that face. On each face the mesh is computed from these curves (Section 3.4) and then populated with backprojections using the incremental algorithm of Section 2.3. Note that the backprojection is *already known* for many cells in each face: It was computed when the backprojection was propagated radially around each vertex of the face (Section 4.2.3) and should be stored at that time.

The algorithm is efficient because it does not have to consider all triples of edges as potential discontinuity surfaces, which can be extremely expensive. It enumerates discontinuity surfaces in time approximately proportional to the number of surfaces.

In summary, the algorithm uses spatial coherence to reduce the number of triples of edges considered. Observations 1 and 2 show how to compute the discontinuity surfaces generated by an edge or a vertex, given the backprojection on the edge or vertex. Propagation operations describe how to propagate the backprojection downward through the scene. The algorithm processes points in order of decreasing height so that generating edges of discontinuity surfaces are encountered before the edges intersected by such surfaces.

5 Implementation Results

The complete algorithm has been implemented and handles all types of discontinuity surfaces. We believe that any competent programmer can implement the algorithm without difficulty. In fact, the most complicated data structure needed is a winged-edge structure to store the polyhedra, the backprojections, and the mesh.

The algorithm was tested on various scenes, two of which are similar to those of Drettakis and Fiume in these proceedings. One of these two scenes consisted of a rectangular prism suspended between a light source and a receiving surface. The other contained a table, a light source, and a floor.

A third scene, shown in Figure 8, is meant to be a spiral staircase. Notice that discontinuity mesh falls upon the steps as well as the floor. Figure 9 shows a close view of some curved EEE discontinuities from the table scene (which is not shown).

The results are summarized in the following table which lists, for each scene, the number of scene edges, the number of surfaces cast, the time to enumerate the surfaces, the time to cast the surfaces, the time to build the mesh, and the time to populate the mesh with backprojections. These times are from runs on a Silicon Graphics Indigo^2 with benchmarks of 59 for SPEC 92 INT and 61 for SPEC 92 FP.

Table 2: Timings (in seconds)

scene	# edge	# surf	enum time	cast time	mesh time	BP time	total time
prism	19	48	0.00	0.10	0.02	0.01	0.13
table	67	121	0.02	1.04	0.14	0.02	1.22
spiral	151	433	0.20	9.48	0.69	0.20	10.57
4 tables	247	633	1.50	23.05	0.79	0.20	25.54

From Table 2 it is clear that the casting of surfaces takes the most time, and that it is essential to cast the absolute minimum number of surfaces. Even if the surface casting is accelerated with a voxel subdivision as is done by Drettakis and Fiume, it will still be the dominant cost of the algorithm. The primary advantage of our algorithm is that it *does* enumerate the minimum number of surfaces and doesn't take much time doing so.

6 Discussion

A fast and practical algorithm has been presented that computes the discontinuity mesh for a polyhedral scene and polygonal light source. Another algorithm has been presented that computes the backprojection in every cell of the discontinuity mesh. These algorithms fully exploit spatial coherence to obtain fast running times.

The implementation of this algorithm has been separate from that of the algorithm by Drettakis and Fiume described in these proceedings. Ideas from both approaches can be combined to yield an algorithm which is more efficient than either. In particular, in our algorithm, the casting of a discontinuity surface can be sped up by using a voxel or BSP approach as is done by Drettakis and Fiume. Their algorithm would be faster if it updated the backprojection from face to face as we describe in Section 2.4.

The most important comparison is in the number of surfaces cast by each algorithm. Surface casting is the most expensive operation, whether or not it is accelerated by a voxel grid. Compared to surface casting, the operations of enumerating the surfaces, building the mesh, and populating it with backprojections take almost no time at all (about one tenth the time for casting).

The heuristic approach of Drettakis and Fiume avoids considering all surfaces with a voxel grid heuristic. Our approach provably considers the minimum number of surfaces [22] by propagating the backprojection through the scene and applying Observations 1 and 2. It is not yet clear which algorithm will perform better in practice.

Acknowledgements

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

[1] A. T. Campbell III and Donald Fussell. Adaptive mesh generation for global diffuse illumination. *Computer Graphics (SIGGRAPH '90 Proceedings)*, 24(4):155–164, August 1990.

[2] 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.

[3] Bernard Chazelle, Herbert Edelsbrunner, Leonidas Guibas, Micha Sharir, and Jorge Stolfi. Lines in space: Combinatorics and algorithms. New York University, Courant Inst. of Math. Sc. Technical Report No. 491, (also in STOC 1989, pp. 382-393), February 1990.

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

[5] Michael Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):75–84, August 1988.

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

[7] Franklin C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (SIGGRAPH '77 Proceedings)*, 11(2):242–248, July 1977.

[8] George Drettakis. *Structured Sampling and Reconstruction of Illumination for Image Synthesis*. PhD thesis, University of Toronto, January 1994.

[9] George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojections. *COMPUTER GRAPHICS Proceedings, Annual Conference Series 1994*, August 1994.

[10] Herbert Edelsbrunner. *Algorithms in Computational Geometry*. Springer-Verlag, 1987.

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

[12] Ziv Gigus and Jitendra Malik. Computing the aspect graphs for line drawings of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):113–122, February 1990.

[13] Pat Hanrahan, David Salzman, and Larry Auperle. A rapid hierarchical radiosity algorithm. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):197–206, July 1991.

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

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

[16] Dani Lischinski, Filippo Tampieri, and Donald Greenberg. Combining hierarchical radiosity and discontinuity meshing. *COMPUTER GRAPHICS Proceedings, Annual Conference Series 1993*, pages 199–208, August 1993.

[17] M. McKenna. Worst-case optimal hidden-surface removal. *ACM Trans. Graph.*, 6:19–28, 1987.

[18] Tomoyuki Nishita and Eihachiro Nakamae. Half-tone representation of 3-d objects illuminated by area sources or polyhedron sources. *COMPSAC'83, Proc. IEEE 7th Intl. Conf. Soft. and Appl. Conf.*, pages 237–242, November 1983.

[19] George Salmon. *A treatise on the Analytical Geometry of Three Dimensions*. Longmans, Green and Co., 1912.

[20] Arthur Scherk. personal communications.

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

[22] A. James Stewart and Sherif Ghali. An output sensitive algorithm for the computation of shadow boundaries. In *Canadian Conference on Computational Geometry*, pages 291–296, August 1993.

[23] Jorge Stolfi. *Oriented Projective Geometry*. PhD thesis, Stanford University, 1988.

[24] Seth Teller. Computing the antipenumbra of polyhedral holes. *Computer Graphics (SIGGRAPH Procedings)*, August 1992.

[25] Seth Jared Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, University of California at Berkeley, 1993.

[26] Oswald Veblen and Wesley Young. *Projective Geometry*. Blaisdell Publishing Co., 1938.

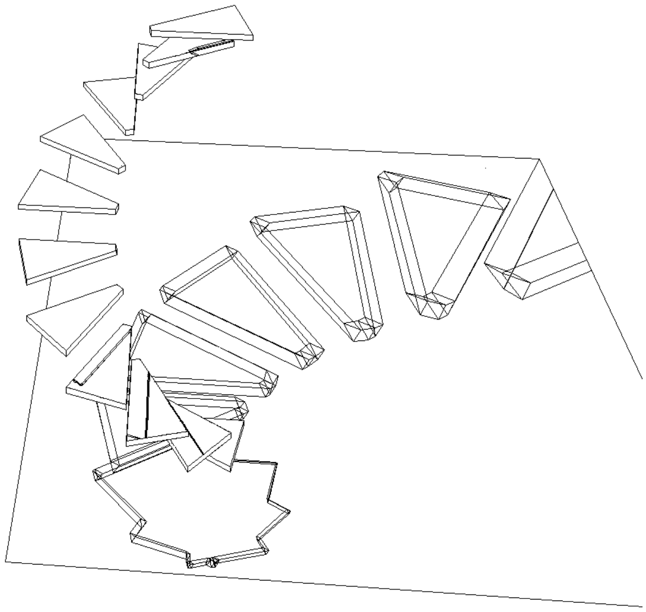

Figure 8: Spiral Staircase and its Discontinuity Mesh

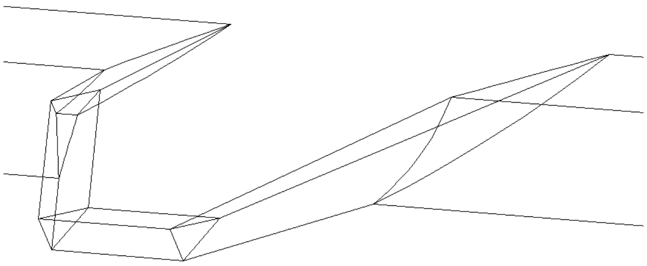

Figure 9: Curved EEE Discontinuities from Table Scene