A Fast Shadow Algorithm for Area Light Sources Using Backprojection

George Drettakis [1]
Eugene Fiume

Department of Computer Science
University of Toronto,
Toronto, Ontario CANADA M5S 1A4

Abstract

The fast identification of shadow regions due to area light sources is necessary for realistic rendering and for discontinuity meshing for global illumination. A new shadow-determination algorithm is presented that uses a data structure, called a *backprojection*, to represent the visible portion of a light source from any point in the scene. A complete discontinuity meshing algorithm is described for polyhedral scenes and area light sources, which includes an important class of light/geometry interactions that have not been implemented before. A fast incremental algorithm for computing backprojections is also described. The use of spatial subdivision, and heuristics based on computed statistics of typical scenes, results in efficient mesh and backprojection computation. Results of the implementation show that the use of the backprojection and discontinuity meshing permits accelerated high-quality rendering of shadows using both ray-casting and polygon-rendering with interpolants.

CR Categories and Subject Descriptors: I.3.7-[Computer Graphics] Three-Dimensional Graphics and Realism.

Additional Key Words: Shadows, umbra, penumbra, discontinuity meshing, global illumination, backprojection.

1. Introduction

The accurate depiction of shadows has long been a concern to computer graphics (e.g., [BoKe70][Appe68]). Shadow algorithms are sensitive to underlying geometric models and to light sources. Because a point-light source is either occluded by an object or it is not, transitions from light to umbral shadow are abrupt. An area-light source can be partially occluded, which results in graduated *penumbral* shadow transitions as well. The identification of shadow boundaries is central to computing better-quality discontinuity meshes and to accelerating global-illumination computations.

Shadow algorithms incorporating penumbra for linear and area light sources have been presented in several ray-tracing contexts [Aman84][PoAm90][TaTo91]. Just as visibility algorithms exploit coherence, so too shadows from area sources have coherent structure that can be used by rendering algorithms. Of special interest is the relationship between scene-light geometry and the radiance within shadow regions. Campbell and Fussell [CaFu91] noted that multiple extrema in radiance can arise in a penumbral region. In [LiTG92][Heck92a], it is suggested that reconstruction quality will improve when discontinuity lines are identified.

We shall consider environments of non-interpenetrating, diffusely reflecting polyhedra illuminated by diffusely emitting area light sources. We present an algorithm that partitions the scene into a mesh of faces, so that in each face the view of the source is topologically equivalent. This view is represented by a data structure called the *backprojection*. We develop a set of heuristics based on properties of typical interior scenes that allow this mesh, called the *complete discontinuity mesh*, to be computed efficiently. Once the mesh has been computed and the backprojection calculated in each face of the mesh, scenes involving area light sources can be quickly rendered. An *incremental* backprojection calculation algorithm is used to greatly accelerate illumination computations. In other approaches this expense is at least that of computing the mesh.

The use of the backprojection and the generation of a complete mesh are important contributions for several reasons. Backprojection can be used to compute images with exact radiance values in the penumbra cheaply and view-independently. These images are useful as a reference to evaluate the quality of approximations such as those in [LiTG92][Dret94a]. Incremental backprojection is generally so fast that high-quality rendering is achieved even when using interpolation, at speeds competitive with previous lower-quality interpolatory approaches. A complete discontinuity mesh is essential to computing backprojections, and it provides precise information regarding variation in radiance that is unavailable in other approaches that compute incomplete meshes. This information has been used for the study of radiance properties [Dret94a] in penumbral regions, which was previously prohibitively expensive.

In Section 2, we review discontinuity meshing and its relationship to previous work. In Sections 3-6 the new algorithm to compute the mesh and an efficient incremental backprojection calculation algorithm are presented. We conclude in Section 7. We present statistics from our implementation throughout, substantiating the intuitions used to develop the algorithm.

2. The Discontinuity Mesh and Backprojection

Changes in visibility, or *visual events*, are related to the interaction of edges and vertices in the scene [GiMa90][GiCS91]. The visual events of interest are: *EV events*, caused by the interaction of a vertex and an edge; and *EEE events*, caused by the interaction of three edges in environment. An EV event is shown in Figure 1 and a EEE event is shown in Figure 2. In both, the visual event occurs when crossing from the point P_1 (red cross) to the point P_2 (green cross). For the EV event, visibility changes only when crossing the plane formed by an edge and a vertex, shown as a white triangle. Specifically, vertex v_1 is visible at P_2, but is not at P_1. For EEE events, three edges form a ruled quadric surface shown in white in Figure 2. At P_2, the edge e_1 of the source is visible, while at P_1 it is not. The plane of Figure 1 and the quadric surface of Figure 2 are *discontinuity surfaces*.

The visible regions of a source from a point are polygons whose vertices are either formed by the projection of scene edges onto the source, or vertices of the original light source. A *backprojection instance* at a point *P*, with respect to a source, is the set of polygons forming the visible parts of the source at that point. In Figures 1 and 2, the polygons with color vertices on the source are the backprojection instances at point P_2. The *backprojection* in a region is a data-structure containing the set of ordered lists of emitter vertices

1. First author's current address: iMAGIS/IMAG, BP 53, F-38041 Grenoble Cedex 9, FRANCE. E-mail: George.Drettakis@imag.fr.

and edge pairs such that at every point P in that region, the projection through P of these elements onto the plane of the source form the backprojection instance at P.

Given a polygonal light source σ and polygonal scene, the partition of the scene into regions having the same backprojection is the *complete discontinuity mesh* of σ. To generate such a mesh, all EV and EEE surfaces that interact with the emitter must be computed. A region of the complete mesh with the *same* backprojection is a *face* of the mesh. Previous algorithms ([LiTG92][Heck92a]) would have missed the EEE curves due to source edges and the corner formed by the two objects in Figure 2 (in red).

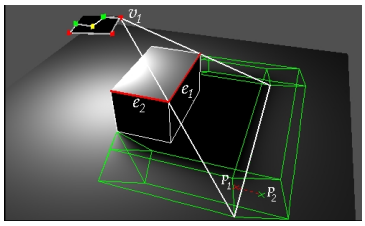

Figure 1. EV Discontinuity Surface

Backprojection Elements. Backprojection lists are composed of three types of elements, called *bp-elements*, and are illustrated in Figures 1 and 2 with different colors. The three types are:

1. Emitter bp-elements, which are vertices of the emitter (in red).
2. Emitter-edge/scene-edge bp-elements. The projection of a non-emitter scene edge (e.g., edge e_1 of the box in Figure 1) through a point P in a face intersects an emitter edge, giving a vertex in the backprojection instance (green vertices).
3. Scene edge/scene edge bp-elements. The projection through a point P of the intersection of two non-emitter edges (e.g., edges e_1 and e_2 of the box) onto the plane of the emitter is a backprojection vertex that lies within the emitter (in yellow).

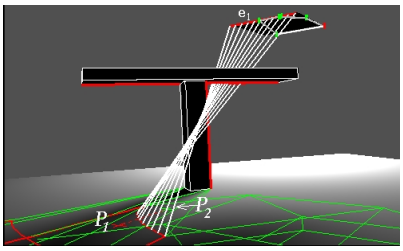

Figure 2. A EEE Discontinuity Surface

Computing the backprojection instance. To compute the instance of a backprojection at a point P, each edge of a Type 2 or 3 bp-element is projected through the point P to a point P' on the emitter plane. To perform this projection for Type 2 bp-elements, the plane formed by the scene edge and point P is intersected with the emitter edge. P' is the resulting point of intersection. Similarly, to find the vertex for a Type 3 bp-element the two planes defined by the point P and each of the two scene edges are formed. The intersection of these two planes forms a ray, whose intersection with the emitter plane is P'. Thus the list of invariant vertices that form the visible portions of the light source at a given point P can be computed at low cost.

The diffuse illumination at P from each resulting polygonal subsource in the backprojection is

$$L(x,y) = \sum_{i=1}^{n} \gamma_i \cos \delta_i, \qquad (1)$$

where γ_i is the angle formed by the point $P(x,y)$ and the vertices of the source v_i, v_{i+1}, and δ_i is the angle between the normal to the receiver, and the plane formed by v_i , v_{i+1}, and P (e.g., [BaRW89]).

2.1. Related Work

The literature contains several partial discontinuity-meshing algorithms. The *extremal penumbral boundary* is the boundary between completely unoccluded regions and penumbral regions. The *extremal umbral boundary* is the boundary between penumbra and umbra. Nishita and Nakamae [NiNa83] directly computed these boundaries for simple environments. Campbell and Fussell [CaFu90] approximated area sources by collections of point light sources, and extended the algorithm in [CaFu91] to area light sources. The extremal penumbral boundary is formed by BSP union operations, but umbral boundaries were not always correctly found, because of the need to compute quadric EEE surfaces. The resulting mesh was represented as a 2-D BSP tree on each receiving surface. Similar mesh computations underlie [ChFe92][ChFe90].

Non-extremal EV surfaces were first calculated in [LiTG92], using BSP trees. Only EV surfaces containing a source edge or vertex were considered, and thus mesh elements often contained subregions with different backprojections. Heckbert considered 2-D discontinuity meshing in [Heck92b], which was extended to 3-D in [Heck92a], in which every EV surface is traced, ignoring EEE surfaces. Teller proposed a similar computation [Tell92], which is equivalent to computing the extremal umbral boundary for such configurations. This is the first treatment of EEE surfaces.

An algorithm to compute the full mesh is proposed by Stewart and Ghali [StGa93][StGa94]. In their algorithm a plane parallel to the source is swept through the scene, and the mesh and the backprojection are built incrementally on this plane. This structure changes during the sweep, requiring more intricate 3-D updates and geometric computations, but guarantees better worst-case behavior than our algorithm. In our case, updates are local and are performed in 2-D, requiring simpler data structures. We also exploit heuristics that, as evidenced below, appear to give excellent results for realistic geometry.

3. Efficiently Computing a Complete Mesh

We require some definitions prior to describing our meshing algorithm. A *feature* is either an edge or a vertex. Any EV surface forms a (planar) *wedge*. A *shadow edge* of a polyhedron with respect to a point is an edge that is contained in the silhouette of that polyhedron when viewed from that point. A *shadow vertex* is a vertex that is attached to at least one shadow edge.

Before computing discontinuity surfaces, we compute the lines at which objects touch. These are D^0 *events* in [Heck92a]. Each object in the environment is visited, and its neighbors are found using a spatial subdivision structure (see below). Lines are inserted on the faces of objects that touch, denoting radiance discontinuities.

The discontinuity surfaces needed for the complete discontinuity mesh for one emitter are: (a) *emitter-EV surfaces*, of which one feature is on the emitter; (b) *non-emitter EV surfaces*, which do not include an emitter feature, but whose plane intersects the emitter; (c) *emitter EEE surfaces*, that contain an emitter edge, which we call E_eEE surfaces; and (d) *non-emitter EEE surfaces*, that do not contain an emitter edge, but cut the emitter polygon. We have designed efficient algorithms for each type, and we have made an engineering decision to employ algorithms with feasible data structures and solution methods. In doing so we sacrifice worst-case asymptotic complexity for practical performance. As we present our algorithms, heuristic assumptions about scene behavior will be made, and the effect on the resulting complexity presented. Statistics run on typical scenes will be presented to support the assumptions. We now discuss our mesh data structure, and algorithms to

handle each type of discontinuity surface.

3.1. An Extended Topological Data Structure

We employ a topological, winged-edge data structure to store the mesh (cf. [Glass91]). The structure stores vertices that are connected by edges. These edges enclose faces that can contain edge cycles. This face-edge-vertex structure maintains consistent adjacency information. Each edge has a left and right face pointer, and a twin edge running in the opposite direction.

During the computation of the penumbral boundaries, edges can be added so that the faces of the mesh are not closed. We have augmented the standard data structure to handle such intermediate configurations. Each face of the mesh is bounded by an edge cycle, called a *face boundary*. A *chain* of edges is a sequence of edges that do not necessarily form a closed loop. The following special edges are identified: *lonely edges*, which are not connected to a face boundary, and *dangly edges*, which are connected to a face boundary but are not in any simple cycle. The left and right faces of a lonely or dangly edge are the same, which maintains consistency.

Computing the arrangement of edges on a receiver. In Heckbert's approach [Heck92a], the edges resulting from the intersection of EV-surfaces with receiver objects are associated in an unconnected fashion with the receiver surfaces. A line-sweep step is required to connect these segments. As the sweep-line passes the points, intersections and face structures are built. In our algorithm, the mesh is instead built incrementally as discontinuity surfaces propagate.

EV surfaces are formed by the edges of a blocker polygon and a vertex either of the emitter or other polygon. These edges are traversed in order of the blocker polygon boundary. After each edge insertion, the receiver surface stores the edge as the "last edge inserted". When the next insertion due to the same blocker occurs, it will be connected to the previous vertex (an endpoint of the previous edge), and thus no search time is required to locate the face in which the new edge will be inserted.

When a new blocker is processed, the line connecting the previous insertion point to the current insertion point is followed. In the early stages, no lines will be crossed, and thus the face of insertion will be found without a search. In later stages, the mesh structure may be searched. During this incremental construction, mesh configurations containing lonely and dangly edges often arise. A chain is "closed" to form a closed face, and all dangly and lonely edges are associated with the correct face. The connectivity structure of the faces is thus built incrementally, avoiding some of the numerical robustness problems of the line sweep approach.

In practice, the number of faces traversed to locate the face of insertion is small, as is the number of faces traversed for intersections during edge insertions. Statistics gathered of the number of faces intersected when inserting an edge support this claim. For scenes where s (the total number of faces) was 360, 1256 and 4829 faces, the average number of faces crossed while inserting an edge was 1.19, 1.15 and 1.27 respectively (Table 1).

Augmenting the mesh to include curved edges. Since EEE surfaces are being treated, it is necessary to store curved boundaries of faces. These resulting quadratic curves (see below) are stored symbolically and any operation to determine point-in-face inclusion or line/edge intersections, operates using curved boundaries.

3.2. Computing the EV Surfaces

Our algorithm to compute all EV surfaces extends Heckbert's [Heck92a]. Apart from the augmented data structure described above, we also use a voxel-based spatial subdivision structure to greatly decrease the number of intersections of EV surfaces with objects in the environment. The algorithm is completed with the extension to non-emitter EV surfaces. Spatial subdivision is again used to accelerate the computation of these events.

3.2.1. Casting Emitter-EV Surfaces Through the Environment

To generate the edges of the discontinuity mesh due to emitter EV surfaces, the general structure of Heckbert's algorithm is followed. For each vertex of the emitter and every other shadow edge in the environment (e.g. v and e in Figure 3(a)), and for each edge of the emitter and every shadow vertex in the environment a wedge is formed. Each wedge is potentially intersected with every polygon in the environment. For each such wedge, the line segments corresponding to these intersections are then transformed onto the wedge plane, and inserted into a sorted list (Figure 3(b)). A visibility step is performed that results in the correct segments being inserted into the meshes of the surfaces intersected. These are represented as thick lines in Figure 3(c). The visibility algorithm is a modified 2-D Atherton-Weiler algorithm [AWG78].

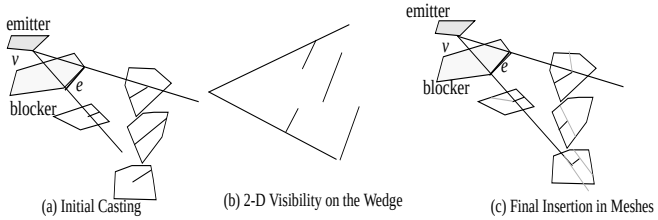

Figure 3. EV surface casting.

Maximal edges are used to determine the boundary between light and penumbra, as described in [CaFu91]. For example, for a specific blocker edge and a source with p vertices, p wedges are formed. At least one of these wedge planes will have all the other wedges in its negative half-space, and is thus maximal. After computing the mesh, the faces are searched to determine those faces whose edges are all due to maximal surfaces. Thus, once the mesh has been computed, the faces that have a completely unoccluded view of the source can be immediately identified.

Acceleration of emitter-EV computations. Since each EV surface needs to be intersected with every object in the environment, the cost of casting one wedge is $O(n)$, where n is the number of objects. Since m, the number of resulting wedge/object intersections, is much smaller than n (see Table 1), the cost of 2-D visibility on the wedge plane is unimportant. The total cost of the emitter-EV surface processing is thus heuristically close to $O(n^2)$.

To reduce the number of wedge intersections, we pre-classify objects on a regular grid [AmW87]. The environment is preprocessed once, at which point the objects are inserted into the voxels that they intersect. We can thus restrict object traversal to those in the affected voxels. To intersect an EV wedge with the objects in the scene, we extend and clip it to the bounding box of the environment. The resulting polygon is scan converted on the voxel grid, and a set of candidate objects is created. As seen in Table 1, the casting time for all emitter-EV wedges may approach $O(n)$ in practice, instead of $O(n^2)$, for the class of scenes studied. The performance of voxel spatial subdivision suffers for densely populated scenes requiring high subdivision [PoAm90]. However, since the wedges are distributed in all directions, poor performance for one wedge is likely to be outweighed by economies on many others.

3.2.2. Non-emitter EV Surfaces

Of all possible EV-surfaces, only those that intersect the emitter polygon are relevant. To efficiently identify these surfaces, at each vertex v in the scene, a pyramid is formed with v as its apex and the emitter polygon as its base (see Figure 4(a)). The pyramid is also extended to the other side of the vertex. Objects that intersect the pyramid are added to a list. The list is traversed, and every edge is tested to see if the EV wedge formed with v cuts the source. These

additional EV surfaces are collected into a list of candidate non-emitter EV wedges. After the candidate list is formed, the non-emitter EV surfaces in the list are cast, using a modified version of the algorithm used to cast the emitter EV surfaces.

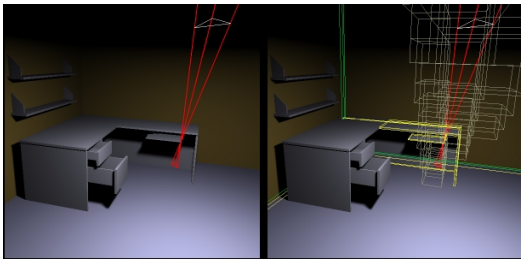

Figure 4. (a) the EV pyramid and (b) the voxels visited.

Acceleration using the voxel structure. The creation of the candidate lists of non-emitter EV events for all vertices, if performed naively, would be $O(n^2)$ time, where n is the number of edges in the environment. Using the spatial subdivision structure described previously, the voxels that are either internal to the EV-pyramid, or are cut by the bounding planes are found (Figure 4(b)). Subsequently, only the wedges formed by v and the edges of these objects are tested to determine if they cut the source. For each vertex, the number k of edges examined is on average expected to be much smaller than n. This is supported by statistics shown in Table 2, run on typical scenes. For a scene of $n = 372$ edges, k was 22.5, while in another scene with $n = 1152$, k was 31.7. In Figure 4(b), only the edges of the highlighted objects are actually tested.

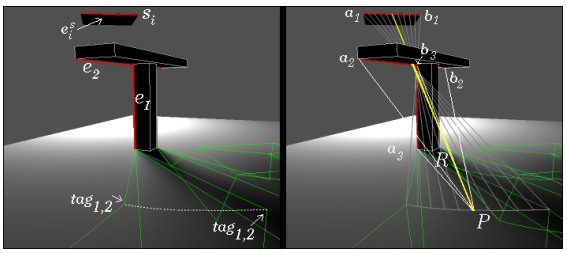

Figure 5. (a) E_eEE tags; (b) Calculation of t

3.3. EEE Surface Computation

We now consider separately (a) E_eEE surfaces that include an emitter edge, and (b) non-emitter EEE surfaces, which do not include an emitter edge, but cut the emitter polygon.

3.3.1. E_eEE Surfaces

The most common EEE curves are caused by discontinuity surfaces that include an emitter edge. Whenever there are more than two skew edges in a scene, these events occur. It is important to identify them efficiently. During the processing of EV surfaces, each mesh vertex receives an integer tag, indicating its generator edges (Figure 5(a)). When two non-adjacent, skew edges meet at a vertex in the mesh, the resulting vertex v_i is inserted in a list. Each such v_i was generated by an emitter vertex s_i; thus the EEE conic defined by the triple $\{e_i^s, e_1, e_2\}$ is drawn onto the surface (Figure 5(a)) and similarly for the triple $\{e_{i+1}^s, e_1, e_2\}$. These quadratic curves either join vertices with the same tag, or are appropriately clipped. If a curve is clipped to an occluding edge of a polygon or a mesh edge, the more general solution for non-emitter EEE surfaces is applied.

3.3.2. Non-Emitter EEE Surfaces that Cut the Emitter Polygon

To compute a complete mesh, it is necessary to identify and process EEE surfaces formed by three non-emitter edges that cut the emitter polygon. For scenes of moderate geometric complexity in which the emitter polygon is small, such events are extremely rare. This is because not only does the ruled quadric surface need to cut the emitter polygon, but the portion that cuts the emitter must be composed of lines that touch the interior of all three defining edges.

Quadric surfaces arising from EEE events. The equation of a general quadric surface is

$$Ax^2 + By^2 + Cz^2 + Dyz + Exz + Fxy + Gx + Hy + Iz + J = 0. \quad (2)$$

The coefficients of the ruled quadric formed by three edges can be derived from [GiCS91, Sa1874]. For every point P on the ruled surface defined by edges e_1, e_2, e_3, there is a unique line on the surface that passes through e_1. If a_1, b_1 are the endpoints of e_1, then the parametric form for any point P_t on e_1 is

$$P_t = a_1 + t(b_1 - a_1), \quad t \in [0,1]. \quad (3)$$

The value of t is used to characterize the point P on the surface given by the other two edges. We call t the *first-edge parameter*. To compute t for a point P_t on the surface, we form the two planes containing P and e_2, and P and e_3. (If P lies on e_2 or e_3, then one plane is formed with the other edge, which is then intersected with e_1 to find t.) Next, we intersect the ray defined by the intersection R of these two planes (yellow line in Figure 5(b)) with the line equation of e_1. Then we solve Eq.(3), setting R=P_t, for $t \in [0,1]$.

The valid region of the quadric is that for which t varies over $[0,1]$. This region is further limited so that the corresponding ruled lines touch the interior of e_2 and e_3. To determine the valid region, the first-edge parameter is found for the endpoints a_2, b_2 of edge e_2 and for the endpoints a_3, b_3 of edge e_3. The intersection of these intervals is the valid region.

Processing the quadratic curves on receiver polygons. The intersection of a plane with the quadric surface is a quadratic curve. To compute this curve, the three generating edges are first transformed into the coordinate system such that the polygon is embedded in the plane z=0. The quadratic curve is now immediately derived from Eq. (2) by keeping only the terms not containing z. The quadratic is converted into standard form such that a monotonic parameterization exists [RoAd90]. The quadratic curve is intersected with the edges of the receiver polygon in the $z = 0$ plane, resulting in a collection of segments, possibly clipped to the edges of the receiver polygon. At the endpoint of each segment, two parameter values are computed: the value of the parameter of the curve s_i and the value of the first-edge parameter t_i. To determine the valid portions of the curve segments, they are sorted by the curve parameter s. The first edge parameters of these segments are computed and only the portions corresponding to valid regions of the quadric are kept.

Identifying the non-emitter EEE surfaces. For each edge e, a volume is formed between the edge and the convex hull of the emitter polygon, in a manner similar to the EV-pyramid shown in Figure 4(a). This polyhedral volume is used to determine which objects are between the edge and the source, and thus potentially can participate in non-emitter EEE surfaces.

All shadow edges that are not outside the volume, and not parallel to e are inserted into a list. This list contains k edges. For the quadric formed by each non-parallel pair of edges, a trivial culling is performed: if all the vertices of the emitter polygon have the same sign when their coordinates are substituted into Eq. (2), the EEE surface is rejected, since it cannot cut the emitter polygon. A large number of surfaces are culled in this fashion. If some vertices have opposite signs, the curve given by the intersection of the quadric with the emitter plane is formed. If the region of the quadric cutting the emitter polygon is valid (as above), the triple of edges is inserted into the candidate non-emitter EEE surface list.

Casting EEE surfaces, visibility processing, and EEE curve insertion. Each EEE surface in the candidate list is intersected with the

scene's polygons. The valid segments are stored in a structure including their endpoints, the object they are associated with, and the parameter values s and t for each endpoint. For endpoint p_i, the distance d_i along the ruled line from the first-edge is calculated. The coordinate system given by (t_i, d_i) is used to perform visibility processing. Each curve segment now corresponds to a line in (t, d) space (see Figure 6(a,b), where the arrows denote increasing d). The lines preserve exact distance only at the endpoints of the curve segments. Relative ordering is maintained because the environment does not include interpenetrating polyhedra.

Once the transformation has been performed, a line sweep visibility processing algorithm is applied. The segments are sorted by increasing first-edge parameter t. Each segment has a structure containing "new segments" as they are inserted during the line sweep. A line parallel to the d axis is swept across the plane (Figure 6(b)), and the intersections of the line with the endpoints of the segments are computed. Visibility is calculated and the parameter t recomputed at the resulting new vertices (see Figure 6(c)). Details of the event processing of the sweep can be found in [Dret94a].

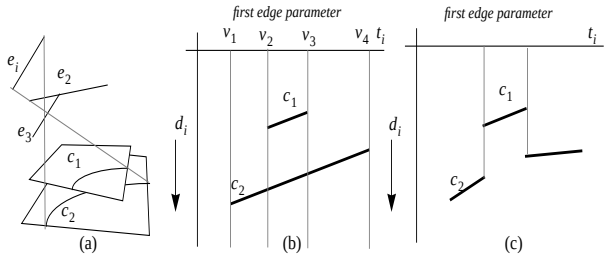

Figure 6. Processing the EEE Surfaces

After the visibility step, the curve segments associated with object that are past the third edge in the sense of the distance d are inserted into the corresponding meshes.

Accelerating EEE surface identification and casting. For the EEE surfaces, a volume is formed between the edge e and the emitter polygon. Grid traversal is extended to trace volumes bounded by planes, so that only the objects that are in the voxels that contain the planes, or voxels cut by the planes are processed. The objects in these voxels then comprise a candidate list significantly smaller than the number of objects. As an example (shown in Table 2), for scenes with 372 and 1152 edges, the average number of candidate edges is 46.74 and 67.37 respectively. After rejecting edges outside the volume, and those parallel to edge e, the average number of edges k actually examined is usually small. From the statistics, the number k is 6.36 for an environment of 372 edges, and 6.18 for an environment of 1152 edges. The number of EEE surfaces that need to be examined is k^2.

When casting the EEE surface, the intersection of the valid region of the quadric surface with the boundaries of the grid is found. The resulting quadratic curves are used to form bounding boxes on the maximal faces of the grid that are then used to form a bounding box of the surface. Within this box, only the objects contained in the voxels cut by the quadric surface are examined.

4. Incremental Backprojection Calculation

Since each penumbral face has a unique backprojection, it suffices to create one structure per mesh face, which contains the appropriate lists of emitter vertices and edge pairs. The backprojection instance can then be evaluated at a small cost at any point. In contrast, explicit computation (such as that in [NiNa83]) of the backprojection instance for each point in the face is expensive. The explicit backprojection instance computation at a point P, is performed by forming and casting all the EV-wedges defined by P and the edges e lying between P and the emitter. Mesh edges are only inserted on the plane of the emitter. This creates the correct list of polygons of the backprojection instance on the emitter plane.

With each edge in the mesh, we store the features that induce the corresponding discontinuity surface. The incremental algorithm traverses each edge in the mesh, and uses this information to incrementally update the backprojection from one face to the next. The expensive explicit backprojection calculations in each face are thus avoided.

4.1. An Incremental Backprojection Calculation Algorithm

The idea of an incremental algorithm is inspired by [GiCS91]. Our incremental algorithm performs a small number of updates to the backprojection when crossing an edge. The incremental algorithm performs a modified depth-first search of the discontinuity mesh on every receiver surface. Starting from a face in light, for which the "backprojection" is the source itself, every face in each penumbral group is visited. When crossing an edge from one face to another, the appropriate incremental actions are taken. The actions required can be distinguished according to the type of edge being crossed. The discontinuity mesh curves are of two types: (a) *EV induced lines*, caused the intersection of an EV wedge and the scene polygon, (b) *EEE induced curves* that are caused by an EEE surface intersecting the scene polygon. We briefly outline the changes to the backprojections when crossing each edge, depending on its type.

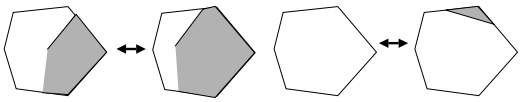

Figure 7. Backprojection Updates for EV edges (emitter vertex).

EV induced edges. An EV edge corresponds to the appearance or the disappearance of a vertex behind an edge. For emitter EV-events involving an emitter vertex and scene edge, the updates to the backprojection are shown in Figure 7. For emitter EV-events involving an emitter edge and a scene vertex, the updates required are shown in Figure 8. For non-emitter EV edges, the updates are shown in Figure 9(a).

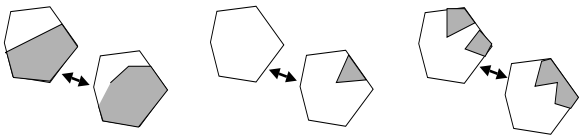

Figure 8. Backprojection Updates for EV edges (emitter edge).

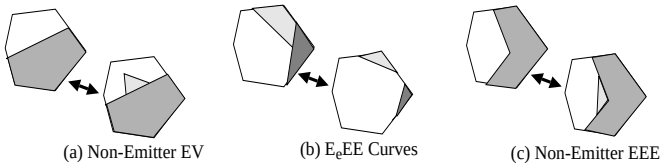

Figure 9. Updates for non-emitter EV, E_eEE and EEE edges.

EEE-induced curves. Traversing a EEE curve requires the addition or deletion of up to two scene edge/scene edge elements in the backprojection. For E_eEE edges the update is shown in Figure 9(b), while for non-emitter EEE edges the updates are shown in Figure 9(c). Details of the data structures and their manipulation are presented more fully in [Dret94a].

5. Rendering

Once the complete mesh has been computed and the backprojections in each face calculated, an accurate image of direct illumination can be rendered. Rendering is performed either by computing exact radiance values at each pixel, or by interpolation.

5.1. Computation of Images with Analytic Radiance Values

To generate an image containing an area light source, previous methods ([CaFu91][LiTG92]) explicitly compute the visible portion of the source at the vertices of the mesh and compute radiance from this. The radiance at other desired points is estimated by interpolation. This is both approximate and expensive. Indeed, the reported cost of computing the illumination at the vertices is at least as large as that of mesh computation.

Given the backprojection in every penumbral face, exact radiance values can be inexpensively computed in penumbral regions. The radiance at any pixel can be computed exactly by finding the instance of the backprojection at the corresponding visible point in the scene, and by applying Eq. (1) for each component of the backprojection, without querying the environment to determine the visible portion of the source. This allows multiple views of scenes with area sources to be computed exactly and cheaply. Examples of such images are shown in Figures 11-14.

5.2. Approximate Rendering Using Polynomial Interpolants

Polynomial approximation allows faster rendering and is essential in the context of light transport calculations such as radiosity-based methods. To achieve such an approximation, the penumbral regions of the scene are collected into *groups* of connected penumbral faces. Bounding boxes of these groups are then calculated (see Figure 10(a)). The remaining regular regions of unoccluded illumination are assigned tensor product interpolants of appropriate degree, using the method described in [DrFi93]. The remaining penumbral faces and the light regions not in the tensor product domains, are assigned triangular quadratic interpolants (see Figure 10(b)). Rendering is performed by either querying the interpolants using ray-casting, or by subdividing the polygons of the interpolant domains into smaller polygons, sampling the interpolant at the vertices and rendering in graphics hardware using Gouraud-shaded polygons. The polygons sent to the pipeline and the resulting image are shown in Figures 10(c) and (d).

The hardware rendering allows interactive walkthroughs to be performed with high quality shadows. In addition, by using the incremental algorithm almost-interactive update rates are achieved with a moving source for simple scenes. When moving the source in the table scene of Figure 10 an effective update rate of 5 seconds per frame is achieved using our current implementation. This includes the complete mesh computation, incremental backprojection calculation and the construction, polygonalization and rendering (in hardware) of the interpolants.

6. Results and Statistics

All of the above algorithms have been implemented, with the following exceptions: previous-edge caching has been partially implemented, so that explicit searches are sometimes performed to locate the face to insert a new edge; the non-emitter EEE surface processing. The results to follow are strong positive indications that for an interesting classes of scenes drawn from office interiors, the performance of our meshing algorithm is good. All of our computations were performed on an SGI Indigo 2 (R4000).

6.1. Performance

In [Dret94a], a simple worst-case complexity analysis of our algorithm shows that it has $O(n^{18})$ time complexity, where n is the number of edges! This is an unrealistic statistic. It is very difficult to construct even an artificial scene for which all worst cases would simultaneously occur. The gap between worst-case analysis and practical performance is worthy of study, but our results suggest that such an analysis may have little to say about real scenes.

In realistic scenes, the use of the voxel grid restricts the number of objects l examined for each surface cast to be much smaller than n. The number m of segments for visibility processing is also small, since most often objects are distributed around the scene. The average number p of faces visited when inserting an edge is small due to the mesh connectivity information and because mesh faces are arranged in groups formed as shadows of objects. Thus the surface casting and mesh-edge insertion algorithms perform efficiently. In addition, the number of non-emitter EV and EEE surfaces appears to grow linearly with n, and by using the voxel structure they can be identified in $O(n)$ execution time (Table 2).

Our algorithm thus runs in close to $O(n)$ time for an interesting class of interior scenes. It achieves such running times: (a) by taking advantage of expected geometric structure, since the number of non-emitter discontinuity surfaces, the average number of objects tested for intersection by any discontinuity surface, and the actual number of intersections are expected to be small and (b) because it is output sensitive, since the cost of inserting an edge into the mesh depends on the size of the resulting mesh.

6.2. Statistics for Emitter EV and E_eEE Implementation

Results are presented in Table 1 for scenes consisting of a table, one complicated desk, the desk and a chair, two desks, and two desks and two chairs respectively. See Figures 10-14 for the rendered scenes. The number of input polygons is n, s is the size of the mesh (number of faces), m is the average number of intersections processed in the 2-D visibility step, and $\max m$ is the maximum. The average number of objects actually tested for intersection for each wedge is l, and p is the average number of faces crossed when inserting an edge. Timings for the use of grid subdivision are given by Grid and NoGrid. The timings include the casting of emitter EV surfaces, the identification and treatment of non-emitter EV surfaces, as well as the treatment of the E_eEE curves.

Scene	n	s	Time (s) (Grid)	Time (s) (NoGrid)	m	max m	l	p
Table	55	360	6.34	9.29	1.60	5	17.89	1.19
1 Desk	187	1256	44.42	139.07	2.09	16	13.48	1.15
1 Desk 1 Chair	331	2437	98.48	527.30	1.91	16	16.20	1.14
2 Desks	349	2488	89.25	626.43	2.13	29	14.07	1.15
2 Desks 2 Chairs	601	4829	181.01	2302.27	1.98	29	17.80	1.27

Table 1. Statistics from the current implementation the meshing algorithm.

In Table 1, it can be seen that the 2-D visibility step is unimportant, since the average number of segments on the wedge plane is small compared to the scene (m vs. n). In addition, the number of faces (p) intersected on average during the insertion of each edge into the mesh is small and nearly constant. This suggests that the incremental mesh-building approach will perform well. Also, note that the size of the mesh s grows linearly with n.

Observe that the average number of objects in the candidate queue for each wedge cast, l, is small compared to n, and indeed may not always increase with n. If l does not grow with n, then the cost of casting a discontinuity surface does not grow either. As more objects were added to the office scenes tested (chairs, shelves, etc.), l did not grow much or even shrunk with n, since these objects do not interfere with each other with respect to the light source. This is often the case for scenes of interiors.

6.3. Statistics for Non-Emitter Events

The identification steps for non-emitter EEE surfaces were also implemented. In this section we present statistics for the EEE surfaces for the scenes computed above, together with the statistics for the identification of non-emitter EV surfaces. In Table 2, e is the number of edges in the scene. The average number of edges in the EV-pyramid is "EV k", and the number of objects visited in the

EEE-volume, and edges found, are "Queue EEE" and "EEE *k*". These quantities have been discussed in Sections 3.2.2 and 3.3.2.

Scene	*e*	EV *k*	EV Surf.	EV Time	Queue EEE	EEE *k*	EEE Surf.	EEE Time
1 Desk	372	22.45	73	18.58	46.74	6.36	0	25.09
1 Desk/1 Chair	636	27.33	148	36.86	57.83	5.90	0	52.10
2 Desks	696	25.56	168	38.81	54.27	6.88	0	47.70
2 Desks/2 Chairs	1152	31.75	394	83.93	67.37	6.18	3	107.23
1 Desk (Src 2)	372	18.85	457	18.95	47.94	6.22	135	28.90
1 Desk/1 Chair (Src 2)	636	28.41	2002	47.16	62.33	7.78	720	61.99
2 Desks (Src 2)	696	26.10	1024	45.83	67.69	7.32	324	62.52
2 Desks/2 Chairs (Src 2)	1152	33.24	2770	91.45	73.16	6.94	1103	110.96

Table 2. Statistics for non-emitter discontinuity surfaces.

The total number of non-emitter EV-surfaces (EV Surf.) was between 73 and 394 for the scenes presented above. The same scenes were run with a bigger source (marked "Src 2" in Table 2), purposely placed so that more of these surfaces cut the emitter. For these new scenes the number of non-emitter EV surfaces was significantly larger (up to 2770), but the average number of edges in the volume was still low. For the first three scenes there were no EEE surfaces cutting the source, while for the scene with two chairs and two desks there were only 3. For "Src 2", the value of *k* was still small, but the number of potential EEE surfaces was larger.

The cost of identifying all the non-emitter surfaces (EEE time and EV Time) was substantial, but not dominant. The computation time for this step grows slowly in *n*; we do not expect this step to become overwhelming with increased scene complexity, assuming the average number of interactions along each discontinuity surface does not grow significantly.

6.4. Results for the Incremental Backprojection Algorithm

The incremental backprojection calculation algorithm has been implemented for relatively simple scenes, in which only EV and E_eEE surface exist in the discontinuity mesh. In Table 3, the time to explicitly compute the backprojections in every face (Time(s) Explicit) is compared to the time taken to compute the backprojections using the incremental algorithm (Time(s) Incremental). The simple scene consists of a parallelepiped floating over a floor and a triangular light source. The table scene is shown in Figure 10, in which the interaction of the table-top and the leg edges create E_eEE surfaces. For both cases the speed-up is immense. For the simple scene the cost of backprojection calculation drops from 2.06 sec. to 0.16 sec., while for the Table scene the cost of computation goes from 9.19 sec. to 0.57 sec., when the incremental algorithm is used. Even better results are expected when the scene is more complex and the explicit computations are thus more expensive.

Scene	n	s	Mesh Time	Time(s) Explicit	Time(s) Incremental
Simple Scene (no EeEE)	19	154	0.77	2.06	0.16
Table Scene (w/ EeEE)	35	363	2.61	9.19	0.57

Table 3. Statistics for the incremental backprojection algorithm.

7. Summary and Conclusions

In this paper a complete and efficient discontinuity meshing algorithm based on the fundamental notion of backprojection has been presented. An incremental backprojection calculation algorithm has also been introduced. Our implementation indicates that our algorithm computes the complete discontinuity mesh in time that grows linearly with the number of objects for typical interior office scenes, and that the use of incremental backprojection results in substantial savings in the computation of radiance in the penumbra. Spatial subdivision is used to reduce the number of intersections between objects and discontinuity surfaces. Identification time for non-emitter surfaces is similarly reduced. Our implementation is the first to compute the complete discontinuity mesh for nontrivial scenes involving all classes of EV and EEE events. The only other algorithm that can compute the complete mesh is described in [StGa94]; we shall compare results when that implementation is complete.

The incremental backprojection algorithm allows the backprojection to be computed in an output sensitive manner. This greatly reduces the expense of computing illumination, once the mesh is computed, compared to previous methods where the illumination computation time can surpass that of computing the mesh. Computed radiance is exact at every pixel, not approximate.

The algorithm has been used to study the behavior of radiance in penumbral regions, and to develop efficient representation of illumination for scenes with shadows [Dret94a]. The algorithm has also been used to develop a mesh of varying quality for scenes with multiple light sources [Dret94b]. For the regions in which one source "washes out" the details of the penumbra, only extremal boundaries are computed, while for the regions where the penumbral detail is required, the complete mesh is computed.

Acknowledgements

The authors wish to acknowledge the financial support of NSERC, ITRC and the University of Toronto. The first author wishes to thank James Stewart for the first implementation of the extended face-edge-vertex structure as well as his helpful insights and suggestions.

References

[Aman84] Amanatides, John, ‘‘Ray Tracing with Cones,’’ *ACM Computer Graphics (Proc. SIGGRAPH ’84)*, vol. 18, no. 3, July 1984.

[AmW87] Amanatides, John and Andrew Woo, ‘‘A Fast Voxel Traversal Algorithm for Ray Tracing,’’ *Proc. of Eurographics ’87*, 1987.

[Appe68] Appel, A., ‘‘Some Techniques for Shading Machine Renderings of Solids,’’ *Proc. of AFIPS JSCC*, vol. 32, pp. 37-45, 1968.

[AWG78] Atherton, P., K. Weiler, and Donald P. Greenberg, ‘‘Polygon Shadow Generation,’’ *ACM Computer Graphics (Proc. SIGGRAPH ’78)*, vol. 12, no. 3, July 1978.

[BaRW89] Baum, Daniel R., Holly E. Rushmeier, and James M. Winget, ‘‘Improving Radiosity Solutions Through the Use of Anaytically Determined Form-Factors,’’ *ACM Computer Graphics (Proc. SIGGRAPH ’89)*, vol. 23, no. 3, July 1989.

[BoKe70] Bouknight, W. J. and K. Kelley, ‘‘An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources,’’ *SJCC, AFIPS*, vol. 36, 1970.

[CaFu90] Campbell, A. T., III and Donald S. Fussell, ‘‘Adaptive Mesh Generation for Global Diffuse Illumination,’’ *ACM Computer Graphics (Proc. SIGGRAPH ’90)*, vol. 24, no. 4, August 1990.

[CaFu91] Campbell, A. T. III and Donald S. Fussell, ‘‘An Analytic Approach to Illumination with Area Light Sources,’’ *Tech. Report TR-91-25*, Comp. Sci. Dept, Univ. of Texas Austin, August 1991.

[ChFe90] Chin, Norman and Steven Feiner, ‘‘Near Real-Time Shadow Generation for Global Diffuse Illumination,’’ *ACM Computer Graphics (Proc. SIGGRAPH ’90)*, vol. 24, no. 4, August 1990.

[ChFe92] Chin, Norman and Steven Feiner, ‘‘Fact Object Precision Shadow Generation for Area Light Source using BSP Trees,’’ *ACM Computer Graphics (SIGGRAPH Symp. on Inter. 3D Graphics)*, 1992.

[Dret94a] Drettakis, George, ‘‘Structured Sampling and Reconstruction of Illumination for Image Synthesis,’’ Ph.D. Thesis, Dept. of Computer Sci., University of Toronto, (CSRI T.R. 293 ftp:ftp.csri.toronto.edu:csri-technical-reports/293), January 1994.

[Dret94b] Drettakis, George, ‘‘Simplifying the Representation of Radiance from Multiple Emitters,’’ *Submitted for publication*, April 1994.

[DrFi93] Drettakis, George and Eugene Fiume, ‘‘Accurate and Consistent Reconstruction of Illumination Functions Using Structured Sampling,’’ *Computer Graphics Forum (Eurographics ’93 Conf. Issue)*, vol. 12, no. 3, Barcelona Spain.

[GiCS91] Gigus, Ziv, John Canny, and Raimund Seidel, ‘‘Efficiently Computing and Representing Aspect Graphs of Polyhedral Objects,’’ *IEEE Trans. on Pat. Matching & Mach. Intelligence*, vol. 13, no. 6, June 1991.

[GiMa90] Gigus, Ziv and Jitendra Malik, ‘‘Computing the Aspect Graph for the Line Drawings of Polyhedral Objects,’’ *IEEE Trans. on Pat. Matching & Mach. Intelligence*, vol. 12, no. 2, February 1990.

[Glass91] Glassner, Andrew S., ‘‘Maintaining Winged-Edge Models,’’ *In Graphics Gems II, edit. by Jim Arvo, Academic Press*, 1991.

[Heck92a]Heckbert, Paul, ‘‘Discontinuity Meshing for Radiosity,’’ *3rd Eurographics Workshop on Rendering*, Bristol, UK May 1992.

[Heck92b]Heckbert, Paul, ‘‘Radiosity in Flatland,’’ *Proc. of Eurographics ’92, Cambridge*, Elsevier, September 1992.

[LiTG92] Lischinski, Dani, Fillipo Tampieri, and Donald P. Greenberg, ‘‘Discontinuity Meshing for Accurate Radiosity,’’ *IEEE C.G. & Appl.*, vol. 12, no. 6, pp. 25-39, November 1992.

[NiNa83] Nishita, Tomoyuki and Eihchiro Nakamae, ‘‘Half Tone Representation of 3-D Objects Illumination By Area Source or Polyhedron Sources,’’ *COMPSAC’83, Proc IEEE 7th Intl. Comp. Soft. and Applications Conf.*, pp. 237-242, November 1983.

[PoAm90]Poulin, Pierre and John Amanatides, ‘‘Shading and Shadowing with Linear Light Sources,’’ *Proc. of Eurographics ’90*, 1990.

[RoAd90]Rogers, David F. and J. Alan Adams, ‘‘Mathematical Elements for Computer Graphics,’’ *(2nd Edition) McGraw-Hill*, 1990.

[Sa1874] Salmon, G., ‘‘Analytic Geometry of Three Dimensions,’’ *Metcalfe*, Cambridge, England 1874.

[StGa93] Stewart, A. James and Sherif Ghali, ‘‘An Output Sensitive Algorithm for the Computation of Shadow Boundaries,’’ *Fifth Canadian Conference on Computational Geometry*, August 1993.

[StGa94] Stewart, A. James and Sherif Ghali, ‘‘Fast Computation of Shadow Boundaries Using Spatial Coherence and Backprojections,’’ *ACM SIGGRAPH Annual Conference Series*, July 1994.

[TaTo91] Tanaka, Toshimitsu and Tokiichiro Takahashi, ‘‘Shading with Area Light Sources,’’ *Proc. of Eurographics ’91*, 1991.

[Tell92] Teller, Seth, ‘‘Computing the Antipenumbra of an Area Light Source,’’ *Computer Graphics (Proc. SIGGRAPH ’92)*, vol. 26, no. 2, pp. 139-148, July 1992.

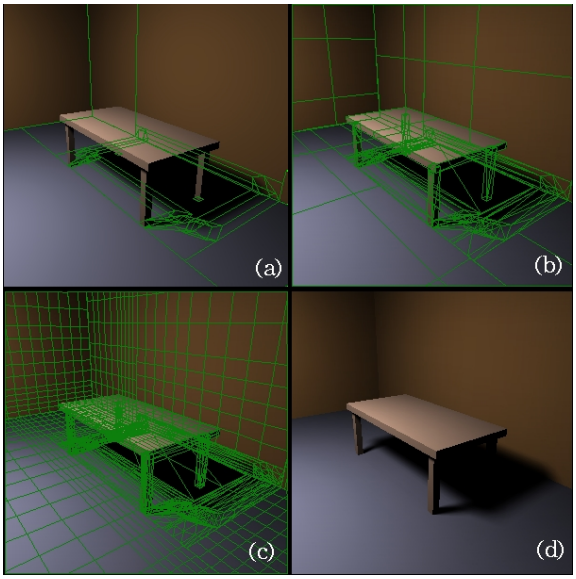

Figure 10. Table scene: (a) mesh and penumbral groups (b) interpolant domains (c) polygons sent to pipeline (d) hardware-rendered image.

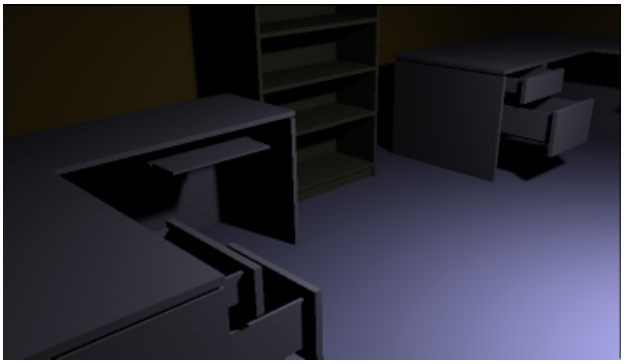

Figure 11. Two desk office scene.

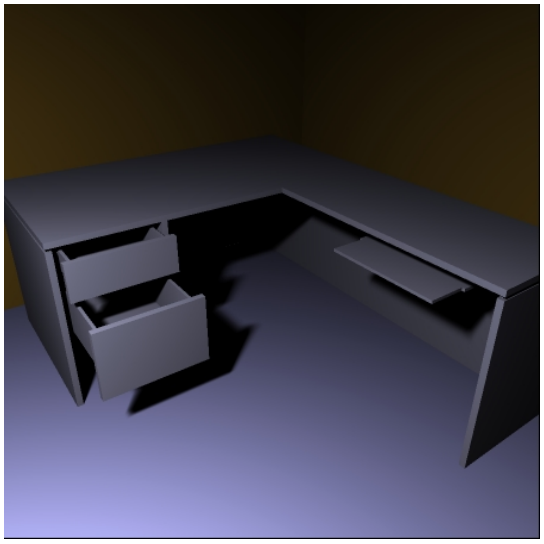

Figure 12. One desk office scene.

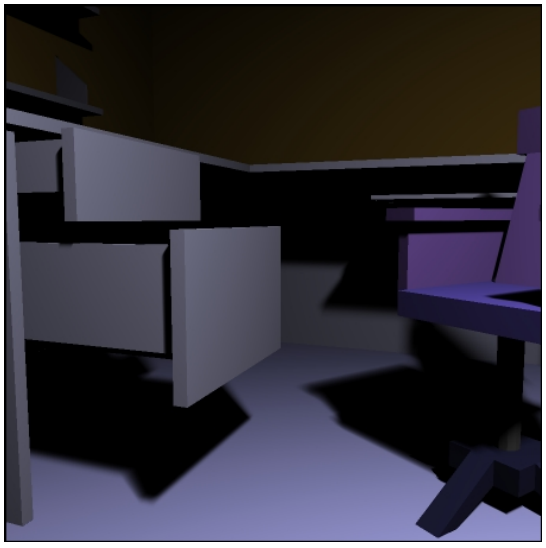

Figure 13. One desk office scene with chair.

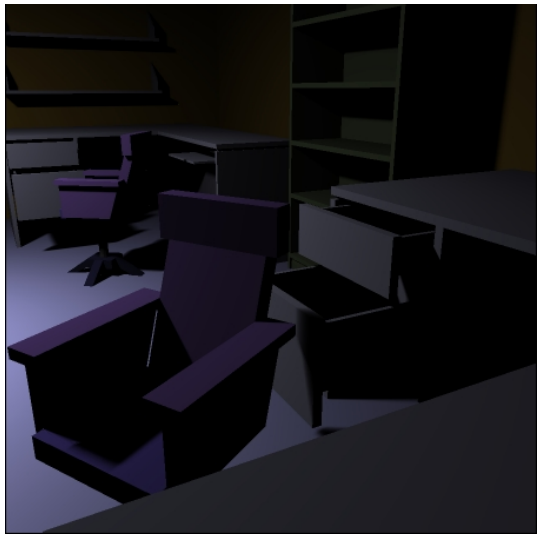

Figure 14. Two desks with two chairs.