Incremental Update of the Visibility Map as Seen by a Moving Viewpoint in Two Dimensions

Sherif Ghali   A. James Stewart

Department of Computer Science
University of Toronto

Abstract. Consider the following problem: A viewpoint moves amongst a set of line segments in the plane and it is desired to maintain the sequence of lines visible from the viewpoint at every increment in its position. The sequence of visible lines is identical for most increments in the position of the viewpoint. It is different only when the viewpoint crosses a visual discontinuity line. Our objective is to be able to quickly report whether the sequence of visible lines needs to be updated and perform the update quickly in that case. We propose an algorithm that satisfies both criteria while using space linear in the number of visual discontinuity lines. This last condition is important because constructing the arrangement of these lines would take space quadratic in their number.
Keywords: animation, visibility maps, on–line algorithms, incremental update, discontinuity lines.

1 Introduction

As a viewpoint moves along a trajectory in a static scene represented by a set of line segments, we would like to compute the view at successive points along that trajectory. This problem can be solved in two settings: off–line and on–line. In the off–line version, we are given all of the trajectory as part of the input whereas, in the on–line version, the position updates are only known as they occur. Clearly, the on–line problem is the more interesting for interactive systems (for example, those that control the viewpoint with a mouse or an infrared tracking system).

The term *frame–to–frame coherence* refers to the similarity between consecutive frames in an animation. This similarity makes it wasteful to render each frame from scratch, given that most of the information has already been computed for the preceding frame. The attempts to take advantage of frame–to–frame coherence fall into several camps:

The traditional solution to this problem is to preprocess the scene such that the depth order can be computed for an arbitrary viewpoint. Binary Space Partitioning, or BSP, is a popular technique in this category [6, 18]. The BSP algorithm is run once for a static scene and produces a tree of size in $\Omega(n)$ and $\mathcal{O}(n^2)$, where n is the size of the scene. For each rendered image, a painter type algorithm traverses the tree. In a real–time system, the cost of this traversal is prohibitive even if the size of the tree is linear in the size of the input.

Define a "plane of support" to be a plane that is tangent to two polyhedra in a scene. By generating these planes and testing whether any one of them is intersected by the viewpoint trajectory, it is possible to update the visibility map as seen by the viewpoint [12]. However, intersecting the viewpoint trajectory at each increment with all planes of support is too expensive.

If the flight trajectory is restricted to be a line segment and is given as part of the input (off–line version), it is possible to compute the locations on the line segment where the visibility map changes [2, 15]. This takes time between quadratic and cubic in the size of the input.

The BSP and the 5D ray classification [1] algorithms have been modified to take advantage of frame–to–frame coherence [25, 10, 26]. The painting step in a BSP–based algorithm is too time–expensive, though, and the tree stored by space partitioning algorithms for ray tracing needs large storage to be effective, which may not be practical for a large input scene.

In [20], an algorithm is presented that preprocesses a set of static objects of size n in $\mathcal{O}(n^2)$ storage and close to $\mathcal{O}(n^2)$ time to maintain the view from a given trajectory in $\mathcal{O}(\log n)$ per update (off–line version).

The algorithm presented in [4] avoids storing the complete set of visual discontinuities by storing only those that are in close proximity to the viewpoint. A hierarchical voxel subdivision of the scene is performed and the discontinuity surfaces intersecting the increasingly larger voxels surrounding the viewpoint are stored. When the viewpoint moves, the discontinuities intersecting the voxels from the new viewpoint position to the nearest ancestor between the old and the new positions need to be computed. A comparative discussion between the algorithms of Pocchiola [20], Coorg and Teller [4], and the algorithm described in this paper is provided in Section 5.

2 Outline of the approach

Consider the following problem. Preprocess a set of segments in the plane such that it it possible to efficiently determine which segments are visible from a moving viewpoint after computing the solution from an initial viewpoint. The output will be a *visibility cycle* describing the segments visible in radial order around the viewpoint. A special tag for a segment (or a line) at infinity will be used when no segment is visible in a certain interval [20].

In Fig. 1, the scene consists of three line segments. The dotted lines define the sites of visual discontinuities across which a viewpoint sees a change in the topological structure of the visibility cycle. For the trajectory shown, the viewpoint can only see line segment b until line l is crossed, at which point segment a becomes visible.

The three dimensional analog of these lines has been used in the computation of aspect graphs [19, 9, 8] and in the computation of discontinuity meshes for scenes illuminated by area light sources [11, 13, 5, 24].

To solve the problem, we take the following steps.

1. Build the visibility graph for the given n line segments. The nodes in

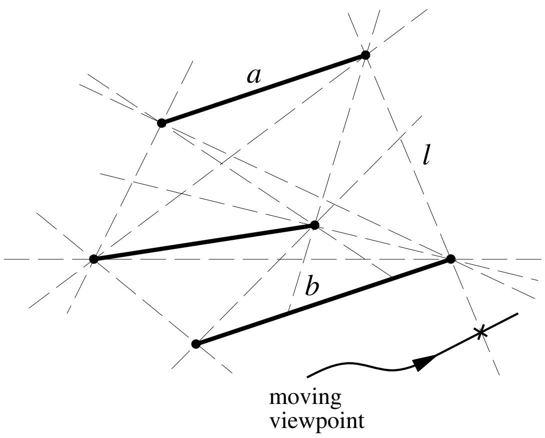

Fig. 1. The dotted lines connect each pair of mutually–visible endpoints. The scene visible from the moving viewpoint remains the same until the viewpoint crosses the line l.

the visibility graph are the endpoints of the line segments. Two nodes are joined by an edge if a line segment connecting the corresponding endpoints does not intersect any other line segment. This graph can be constructed in $\mathcal{O}(m \log n)$ time [17] for a graph with m edges.

2. Given a set of these discontinuities, preprocess it into a data structure that a viewpoint traversing the scene can query to determine whether any discontinuity has been crossed. This will be discussed in Section 3.
3. As long as no discontinuity is crossed, the viewpoint sees the same visibility cycle. If one is crossed, however, then an update needs to be performed on the cycle to reflect the features newly visible. This will be discussed in Section 4.

3 Efficiently Detecting the Intersection of the Flight Path with a Discontinuity Line

Imagine constructing the arrangement [22] of the m discontinuity lines which embed the pairs of mutually–visible endpoints in the visibility graph. As the viewpoint moves but remains inside one cell in the arrangement, the set of visible segments and their order around the viewpoint remains unchanged. We would therefore like to compute the description of the boundary of the current cell and intersect it with the path with each position increment. The obvious approach to do so is to compute the complete arrangement. The description of each cell will consist of the set of lines bounding it. Unfortunately, the size of the resulting arrangement will be in $\Theta(m^2)$. Given that the number of discontinuity lines is already between linear and quadratic in the number of input segments, we cannot afford to spend that much space for the arrangement. We attempt instead to form an implicit description of the cell the viewpoint is currently in. To do so, we review a tool that we shall use: geometric duality.

3.1 Geometric Duality

Geometric Duality is an old tool in classical Projective Geometry. In recent years, it has proved useful in solving a number of geometric problems (see, for example, [3]). Duality allows us to map a set of objects with which we have little intuition (e.g. lines, or, in general, hyperplanes) to another set about which we can think more easily (e.g. points). Furthermore, this mapping does not cost us anything: we simply reinterpret the data we have.

One such duality function D is to map a point (a, b) onto the line $ax + by = 1$ and vice–versa. Each of the line and the point is called the *dual* of the other. This mapping is a bijection. The plane where the original data sit is called the *primal plane* and the one to which they map is called the *dual plane*. Duality is *incidence preserving*: If a point P is incident to a line L in the primal plane then the dual of L, the point $L' = D(L)$ in the dual plane, is also incident to the dual of P, the line $P' = D(P)$.

Consider that a line L defines two open halfspaces: one that contains the origin (called L_1) and one that does not (called L_2). Duality is *sidedness preserving:* Point P lies in halfspace L_1 of a line L if and only if the dual point L' lies in halfspace P'_1 of the dual line P'. In the next section we exploit this property.

3.2 Computation of the description of the current cell

We need to store the description of the set of lines bounding the cell in which the viewpoint lies in the arrangement of lines in the primal plane. Since computing the arrangement is too expensive, we take the following approach. Dualize (i.e. compute the dual) of the set of discontinuity lines S. Dualize the viewpoint V. Let the duals be $S' = D(S)$ and $V' = D(V)$. V' divides the dual plane and the set of points S' into two sets S'_1 and S'_2, the first lying on the same side of V' as the origin and the second on the opposite side. Construct C_1, the convex hull of S'_1, and C_2, the convex hull of S'_2. See Fig. 2. It follows from the construction and from duality being sidedness preserving that the set of lines bounding the cell in which the viewpoint lies in the primal plane is a subset of the set of points on the two convex hulls C_1 and C_2.

As long as the viewpoint moves inside the current cell in the primal plane, the set of visible segments remains unchanged. As the viewpoint V gets closer to a line L on the boundary of the cell and becomes incident on it, the dual $V' = D(V)$ becomes incident on the dual of that line, $L' = D(L)$. When V crosses L in the primal plane, it enters another cell. This is reflected in the dual plane by the point L' crossing line V'. See Fig. 3. L' now has to be deleted from one convex hull and inserted into the other. In what follows, we discuss how to perform the insertion and deletion of points.

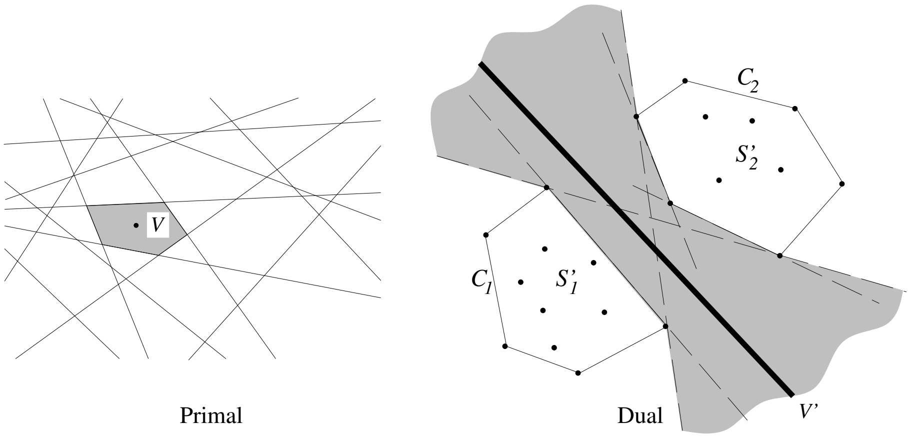

Fig. 2. The figure on the left shows the arrangement of the discontinuity lines. The set of points shown on the right is the set of dual points of these lines. All viewpoints in any one cell in the arrangement on the left have the same visibility cycle. When the viewpoint V moves in the shaded cell in the primal plane, its dual line V' also moves in the shaded region shown in the dual. The exit of V from the shaded cell in the primal is signaled in the dual by V' intersecting one of the two convex hulls C_1 or C_2.

3.3 Dynamic Maintenance of Convex Hulls

We need to maintain the convex hulls of two sets of points in the dual plane as points are deleted from one and inserted into the other. A lower bound of $\Omega(\log n)$ per insertion/deletion operation follows from the lower bound of the convex hull problem [22]. Currently, the best known algorithm achieves $\mathcal{O}(\log^2 n)$ time per operation [16]. It is an open problem whether an algorithm exists that can perform the updates in logarithmic time.

When inserting a point into a convex hull in the dual plane, the point is connected to the two common tangents between it and the hull. The points that disappear from the hull boundary correspond to the discontinuities in the primal plane that no longer bound the current cell. When deleting a point from a hull in the dual, an arbitrary number of points can appear on the hull boundary. These points correspond to the lines in the primal plane that bound the new cell the viewpoint enters.

The two convex hulls that we maintain in the dual plane give us a concise description of the discontinuities in the primal plane which, when crossed, may require an update of the visibility cycle. The hulls can be stored in space linear in the number of discontinuities. For every new position of the viewpoint V, the dual V' is computed and a test performed to see if any point on either of the two convex hulls in the dual plane has switched sides. If more than one point has switched sides, the distances separating the corresponding discontinuities from the old viewpoint is compared in the primal plane to determine which one is the first to be crossed. Comparing each new viewpoint position with all points on the two hulls can potentially be expensive but this is sufficient because few points lie on the hulls in practice.

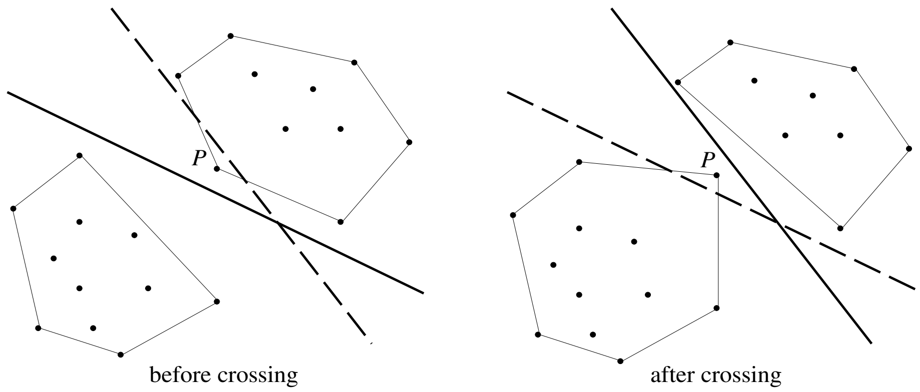

Fig. 3. The figure on the left (respectively, right) shows the configuration in the dual plane before (respectively, after) the discontinuity line is crossed. In both figures, the solid line is the dual of the current position of the viewpoint with the dashed line showing the position of the dual after (respectively, before) crossing. As the discontinuity point P is crossed, P is deleted from one set of points and inserted into the other, while maintaining the convex hulls of both sets of points.

Occasionally the viewpoint V will cross a discontinuity that is defined by two *hidden* endpoints (recall that the discontinuity lines embed two segment endpoints and extend to infinity). An endpoint is hidden if the line segment joining it to the viewpoint intersects a segment of the scene. A portion of a discontinuity is called a *true* discontinuity if the visibility cycle changes when the viewpoint crosses the discontinuity in that portion. It is called a *false* discontinuity if no change in the visibility cycle occurs. To determine whether a discontinuity crossing is a true or a false one, we store with each discontinuity the coordinates of the two endpoints that gave rise to it. Also, a small modification of the algorithm of Overmars and Welzl [17] allows us to identify the two edges (if any) that intersect each discontinuity line nearest to each of the two endpoints that gave rise to it. Even though a false discontinuity does not require a change to the visibility cycle, the convex hulls in the dual plane still need to be updated to maintain a current description of the cell the viewpoint is in.

We take advantage of the fact that the set of points to be maintained is static by the following simplification of the algorithm of Overmars and van Leeuwen [16] which lends itself to an easy implementation. The points are sorted in the x-direction and are stored at the leaves of a binary tree. See Fig. 4. Every node in the tree stores two convex hulls; the one above the dual V' of the viewpoint V and the one below it. Initially, each leaf node stores a variable to identify whether it is above or below V'. The two convex hulls at a node are constructed from its children's hulls by finding their upper and lower common tangents. When a point switches sides with respect to V', only those nodes that lie on the path from the corresponding leaf node to the root need be updated. If the number of points is n, there are $\mathcal{O}(\log n)$ nodes on that path and $\mathcal{O}(\log n)$ time needs to be spent at each node to compute the upper and lower common tangents for a total update time of $\mathcal{O}(\log^2 n)$. In practice, since the tree itself is static (but not the hulls stored at its nodes) a heap implementation suffices. To guarantee the bounds above for time but in particular for space, Overmars and van Leeuwen avoid duplicating the points on the convex hull at each level in the tree by storing only one copy of the convex hulls at the root node in concatenable queues. An update operation (either insert or delete) is performed by first descending down the tree on the path from the root to the leaf that needs to be updated and the concatenable queues are broken at each level into two halves. This is followed by a percolating step to reconstruct the hulls at each level from the leaf to the root. Given that the number of points on the convex hull is small in practice, our implementation stores the complete convex hulls at each level.

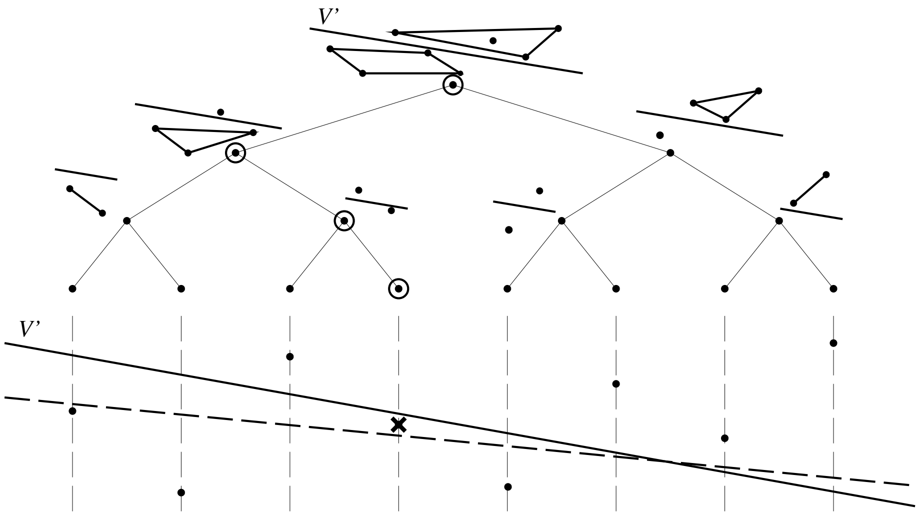

Fig. 4. The bottom half of the figure shows the points in the dual plane. These points are sorted by x-coordinate and stored at the leaves of the binary tree shown above. The dotted line is the dual of the previous position of the viewpoint. At every node in the tree, the convex hulls of the points that lie above and below the line are stored. After the point (shown in 'x') crosses the line, only the convex hulls at the nodes of the tree that lie on the path to the root need to be updated (nodes are circled in the figure).

4 Updating the Visibility Cycle

An initial view can easily be computed in $\mathcal{O}(n \log n)$ time by sweeping radially around the starting location of the viewpoint. The output will be a circular list of the visible segments. This list may need to be updated when the viewpoint crosses a discontinuity line. The problem of updates is well–studied in the computation of aspect graphs [19, 9] and in the computation of the visible portions of area light sources [24, 5]. The following presentation is the two–dimensional version of the problem and is similar to ideas presented in [27, 20].

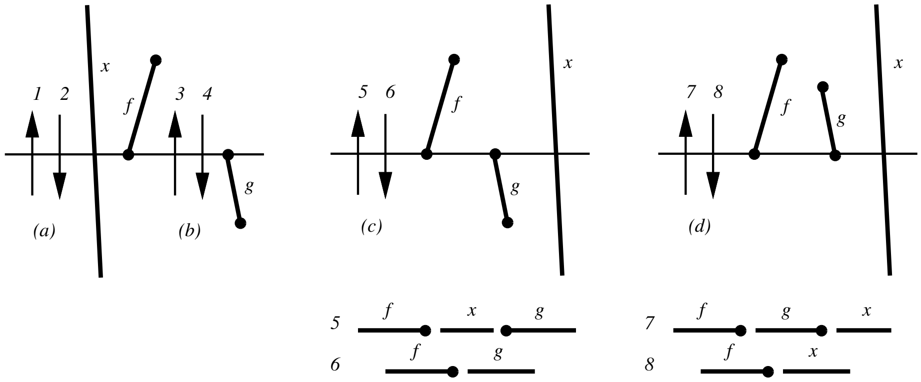

Fig. 5. Modes of crossing a discontinuity line: (a) Edges contributing to the discontinuity are not visible; no action needs to be taken. (b) Viewpoint is crossing the interior of the edge defining the visibility graph; no action needs to be taken. (c) Both edges are visible; a third edge is either added or deleted. (d) One of the two edges is visible from one side of the discontinuity line and both are visible from the other side; either insert or delete edge g from the visibility cycle.

Fig. 5 shows the four modes in which a discontinuity line can be crossed. The two edges contributing to the discontinuity are labeled f and g, and an arbitrary other edge is labeled x. In (a) the endpoints of f and g are obscured by x. In (b) the viewpoint crosses the interior of the line segment connecting the endpoints of f and g. No update is needed in either of these two cases. In cases (c) and (d) the visibility cycle needs to be modified. Underneath each of these cases a section of the cycle is shown. An endpoint is highlighted with a solid point if it is the actual endpoint of the segment. Non–highlighted endpoints refer to edges that are partially obscured by another edge. The section of the cycle where the two edges f and g occur is compared to each of the four transitions $5 \ldots 8$ and a simple update is performed on the cycle in each case.

5 Conclusion

An on–line algorithm to maintain the current cell in an arrangement of discontinuity lines has been implemented in C++. The implementation has been simplified by building it on the LEDA library [14]. Real–time tracking of the position of the viewpoint is performed as the mouse moves and the set of visible line segments is identified for each increment. An animation of this algorithm is available on video [7].

Note that the number of discontinuities is in $\Omega(n)$ and $\mathcal{O}(n^2)$ in a scene of n segments. Experimental results with random sets of line segments indicate that the number of discontinuities grows linearly with the number of segments. Intuitively, in a dense scene, each segment endpoint will tend to see a constant number of other segment endpoints.

Comparisons

In what follows, we compare the algorithm presented here with the one by Pocchiola [20]. There are two main advantages to our algorithm. The first is the ability to process queries on–line, which is important for interactive applications. The second is that we only spend time proportional to the number of edges in the visibility graph during preprocessing [17] whereas Pocchiola's algorithm needs time in $\Omega(n^2)$ for both time and storage. (Notice, however, that the number of edges in the graph can reach $\Theta(n^2)$, e.g. for a set of segments on a circle [27].) However, we spend time $\mathcal{O}(\log^2 n)$ per update whereas he only spends $\mathcal{O}(\log n)$ and can deal with more complex convex shapes than line segments.

Also, we use the dual plane to avoid having to compute the arrangement of lines in the primal plane whereas Pocchiola computes an arrangement of curves which takes $\Theta(n^2)$ space in a scene of n objects. Finally, the algorithm presented here has been implemented using simple data structures. A drawback in our approach, on the other hand, is that we keep the portions of the discontinuity lines that are not important (because a viewpoint crossing such a portion cannot see the the two endpoints that gave rise to the discontinuity). Pocchiola is able to remove the superfluous curved segments arising in his case during the preprocessing and thus does not get false signals to update the view during the processing of the trajectory.

It is more difficult to compare the algorithm presented here and that of Coorg and Teller [4] without empirical trials. The algorithm we propose is simple and is able to update the position of the viewpoint in polylog time in the number of discontinuities. The algorithm by Coorg and Teller suffers from the traditional problem of relying on a user–given parameter to perform the subdivision of the scene into voxels. In our approach, we don't compute the arrangement of the discontinuity lines since it has quadratic cost, but we spend time in the preprocessing step to compute the (full) visibility graph. Coorg and Teller, on the other hand, avoid the computation of the complete set of discontinuities (which could use space cubic in the size of the input for the 3D case) by only considering the interaction between the objects visible and the partition of the space that lie in their shadow volume with respect to the viewpoint. Of course, the algorithm of Coorg and Teller operates in 3D whereas ours is in the plane.

Problem in 3D

The obvious question is whether it is possible to scale upwards and solve the problem in three dimensions. Several complications arise in the 3D case that make this a non–trivial task. No algorithm is known that can compute the discontinuity surfaces in a scene in time proportional to the size of the output. Also, as is well known [23, 19, 9], *three* edges can construct a quadric surface. The arrangement of the discontinuity surfaces in the primal space would no longer be a planar arrangement. Quadrics would also add a difficulty since the dual of these surfaces is no longer a point. Finally, an insertion or deletion of a point to a convex hull of a set of discontinuities in the dual space can take linear time in the size of the hull, although we would expect this to be much less in practice.

References

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

[2] M. Bern, D. Dobkin, D. Eppstein, and R. Grossman. Visibility with a moving point of view. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 107–117, 1990.

[3] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

[4] S. Coorg and S. Teller. Temporally coherent conservative visibility. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 78–87, 1996.

[5] G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using backprojection. *Computer Graphics Proceedings, Annual Conference Series 1994*, 28:223–230, August 1994.

[6] H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3):124–133, 1980.

[7] S. Ghali and A. J. Stewart. Maintenance of the set of segments visible from a moving viewpoint in two dimensions. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages V3–V4, 1996.

[8] Z. Gigus, J. Canny, and R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Trans. PAMI*, 13(6):542–551, 1991.

[9] Z. Gigus and J. 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.

[10] E. Groller and W. Purgathofer. Using temporal and spatial coherence for accelerating the calculation of animation sequences. In Werner Purgathofer, editor, *Eurographics '91*, pages 103–113. North-Holland, September 1991.

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

[12] H. Hubschman and S. W. Zucker. Frame-to-frame coherence and the hidden surface computation: Constraints for a convex world. *ACM Transactions on Graphics*, 1(2):129–162, 1982.

[13] D. Lischinski, F. Tampieri, and D. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics & Applications*, pages 25–39, November 1992.

[14] K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *CACM*, 38:96–102, 1995.

[15] K. Mulmuley. Hidden surface removal with respect to a moving point. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 512–522, 1991.

[16] M. H. Overmars and J. van Leeuwen. Dynamically maintaining configurations in the plane. In *Proc. 12th Annu. ACM Sympos. Theory Comput.*, pages 135–145, 1980.

[17] M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 164–171, 1988.

[18] M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5:485–503, 1990.

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

[20] M. Pocchiola. Graphics in Flatland revisited. In *Proc. 2nd Scand. Workshop Algorithm Theory*, volume 447 of *Lecture Notes in Computer Science*, pages 85–96. Springer-Verlag, 1990.

[21] M. Pocchiola and G. Vegter. The visibility complex. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 328–337, 1993.

[22] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

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

[24] J. Stewart and S. Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. *Computer Graphics Proceedings, Annual Conference Series 1994*, 28:231–238, August 1994.

[25] E. Torres. Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes. In C. E. Vandoni and D. A. Duce, editors, *Eurographics '90*, pages 507–518. North-Holland, September 1990.

[26] D. Tost. An algorithm of hidden surface removal based on frame-to-frame coherence. In Werner Purgathofer, editor, *Eurographics '91*, pages 261–273. North-Holland, September 1991.

[27] E. Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Inform. Process. Lett.*, 20:167–171, 1985.