COMPUTATION AND MAINTENANCE OF VISIBILITY AND SHADOWS IN THE PLANE

Sherif Ghali
Department of Computer Science
University of Toronto
Toronto, Canada
ghali@dgp.toronto.edu

ABSTRACT

We consider visibility and shadow problems in the plane under static and dynamic conditions. We show that several problems in this domain are intimately related and that a careful formulation of the requirements of each problem can lead to a great deal of reuse in the implementation. This is of interest to the practitioner who may find it difficult to implement visibility and shadow algorithms given their complexity. A solution of these problems in the plane turns out to be a necessary component of their solution in space and by separating these components, difficult issues of implementation are taken care of at the design level. We also give specific algorithmic results. We show a reduction between shadow and visibility computations under a point light source, between static and dynamic computations under a point light source, and visibility and shadow computations between a point and a linear light source.

Key Words: Computational Geometry, Visibility, Shadows, Dynamic Algorithms, Class Hierarchy

1 INTRODUCTION

Visibility problems lie at the center of computer graphics. They are interesting both because they pose fundamental questions about the geometric primitives that we use and because they lie at the heart of graphics computation. The large amount of algorithms available in this area, unfortunately, consists of separate results. The theoretician is at a loss because these individual results give little insight into the structure of these questions and the approaches needed for an implementation and the practitioner has many scattered algorithms available to solve any one of these problems. The shrewd, or industrial, practitioner is not willing, however, to implement solutions to these problems individually. The design complexity of some of these problems is high and the implementation quickly becomes a mass of unmanageable code, leading the practitioner to resolve either into taking implementation shortcuts by not implementing the most efficient techniques available, or to develop the software purely as a research tool to show that it works on a small subset of the input domain.

Even though we also give specific algorithmic results, we consider that the major contribution of this paper is to develop an abstraction model for visibility and shadow problems in the plane. We hope that the theoretician will see a clear hierarchy of the problems which may make it easier to uncover interesting open problems and that the practitioner versed in object–-oriented paradigms will find it straight–forward to implement the solutions described in this paper. In particular, it is prudent to proceed carefully and address all the issues for these problems in the plane for two reasons. First, these same questions arise in space and the planar study gives us valuable insight. Second, a solution of the problems in space turns out to consist of planar subproblems.

We first allude to a result on computing visibility from a point, then we show how the boundary points separating light from shadow areas can be generated efficiently and describe a method to partition the input scene into two lists, one giving the portions in light and the other giving the portions in shadow. These two lists are valuable since they would allow us to use hardware renderers for fast shading of objects while calculating the shadow boundaries in "object space." We then show how to maintain the data structures that we built for the case when a primitive in the input is moving.

We then build on these data structures to compute the view of the scene as an observer moves along a line segment. We use that to compute the boundary shadow points in a scene illuminated by a linear light source. And finally, we show how to use the shadow points to determine the different views of the light source by using string manipulation only.

The class design presented here separates the difficulty depending on a client's needs. For example, a client requiring a view computation in the static case will not have to use an implementation having more features available than he asked for nor waste time during running to take care of the extra data structures.

The figure captions, when read in sequence, will provide a quick overview of the problems addressed.

2 PREVIOUS WORK

Visibility problems received much attention in the geometry community. In some of the relevant earlier work [Edels83], an algorithm was presented to compute the view from a point. We briefly describe that algorithm when we use it in Section 4. The visibility complex is a data structure that encodes visibility relationships in the plane and which has been used successfully for a number of applications [Pocch96]. The different views of a scene were studied in a data structure called the aspect graph [Plant86, Gigus91].

After some pioneering work in the area of the computation of the shadow boundaries [Nishi85], the area became more mature which resulted in a number of algorithms to compute what is called the discontinuity mesh [Lisch92, Heckb92, Stewa94], and also heuristics based on voxels to compute the static [Drett94] and the dynamic [Losco97] discontinuity mesh.

Planar problems were shown to be important parts of the problem in space [Ghali96a]. These problems were addressed both directly [Ghali96a] and using the visibility complex [Riviè97]. It is useful to note that using a set of line segments as an input in visibility computations in the plane is undesirable as the sidedness of a point on a line segment has to be addressed. It is also harder to embed the planar case in the solution to the problem in space. Dynamic problems in the plane were tackled to update form factors in a scene with moving objects [Orti96] and to update the view from a moving point in a static scene [Ghali96b]. Also, the computation of critical points on a line segment was studied in [Mulmu91, Bern94]. We use their terminology in contrasting opaque and transparent visibility.

The trapezoidal decomposition [Mulmu94] is a data structure that can answer many dynamic queries in the plane. However, we describe here an alternative which does not require the implementation of more than search trees and lists. Another data structure which offers the essence of visibility in an elegant formulation is the visibility graph [Overm88, Ghosh91]. Despite its simplicity, the incidence relationship of vertices in this graph does not facilitate its use in this application domain.

3 CLASS HIERARCHY

Figure 1 shows the class design for the problems discussed in this paper. Two symbols from the Booch notation [Booch94] are used: the cloud shape represents a class and the line segment represents a relationship between two classes. The class adjacent to the circle *contains* one or more instances of the class at the opposite endpoint of the line segment. The presentation in this paper follows the class design shown.

Details about the classes addressing visibility and the shadow points from a point light source are given in Sections 4, 6, and 7; classes addressing the maintenance of the view and the shadow points are discussed in Sections 8 and 9; and the classes addressing the view and shadows from a line segment are discussed in Sections 10, 11, and 12. The transparent view is discussed in Section 5.

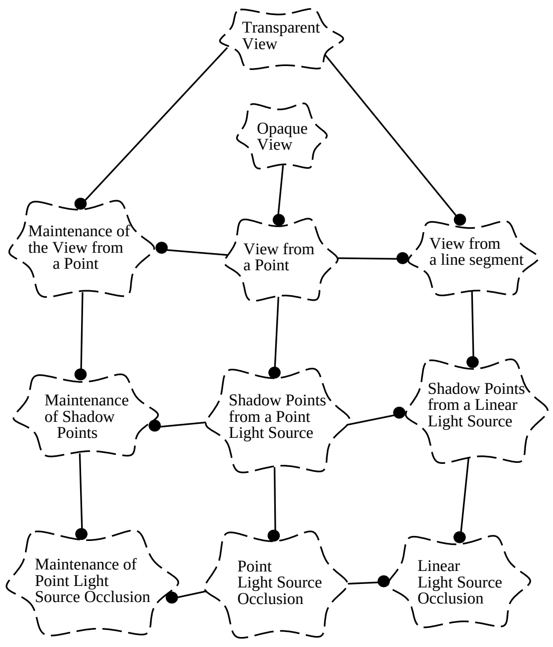
Transparent View
Opaque View
Maintenance of the View from a Point
View from a Point
View from a line segment
Maintenance of Shadow Points
Shadow Points from a Point Light Source
Shadow Points from a Linear Light Source
Maintenance of Point Light Source Occlusion
Point Light Source Occlusion
Linear Light Source Occlusion

Figure 1: Class design for the problems discussed in this paper.

4 OPAQUE VISIBILITY

Given a set S of polygons and a point P in the plane, we describe how to build a data structure, called the opaque visibility cycle and perform a set of operations on it. We present this Abstract Data Type and shall see shortly its applications.

The visibility cycle consists of a sequence of the visible edges and vertices from a viewpoint positioned at P. See Figure 2.

In addition to functions for generating an opaque view and reporting it, the opaque visibility ADT supports operations to search for a vertex, deleting a vertex–edge pair, and inserting a vertex–edge pair following/ preceding a vertex. For a set of polygons of n vertices, it is possible to compute the opaque view in worst–case optimal $O(n \log n)$ time [Edels83]. The essential step is that it is possible to merge two opaque visibility cycles in linear time in their size. To be able to locate a vertex or an edge in the opaque visibility cycle, we build two search trees, one containing the indices of the vertices present in the visibility cycle and pointing to the location of each vertex in the linked list (and similarly for the edges).

5 TRANSPARENT VISIBILITY

If the polygons in S are made of a translucent material, a viewer situated at P will be able to see through

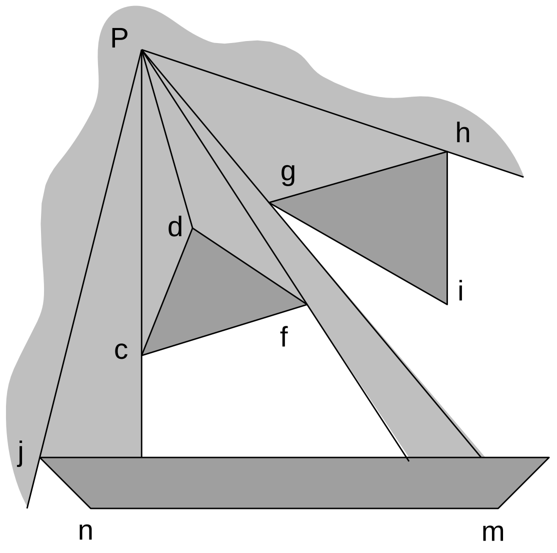

Figure 2: A viewer sees a subset of the edges defining the polygons shown. The polygons are highlighted in dark in the Figure. The opaque visibility cycle consists of a sequence of edge and vertex indices. It is identified in light grey and consists of the sequence of vertices $jcdfgh$ in addition to the interleaving edges.

the polygons. The *transparent visibility cycle* is a data structure that stores the ordering sequence of the polygon vertices as seen from P. It also stores one level of visibility *below* each polygon vertex [Bern94]. Imagine a ray going from P to a polygon vertex v. This ray may intersect an edge farther than v. We will refer to this edge as *below*(v) and to the ray starting at v and ending at that edge (if any) as the *light ray* at v. In practice, it is convenient to assume that a special circle of infinite radius surrounds the input. If no polygon edge is intersected by the ray PV, we store a special index indicating that the infinite circle is intersected. See Figure 3.

The transparent visibility cycle is stored in an array with each entry holding the index of a vertex and the index of the vertex *below*(v). The transparent visibility cycle ADT supports operations to compute the transparent view and to swap a pair of vertices in the view.

6 SHADOW POINTS UNDER A POINT LIGHT SOURCE

Assume that a point light source positioned at P illuminates the polygons in S. We say that an edge in S is in *shadow* if no point on the edge can see the light source P. We also say that an edge is *in light* if all points on the edge can see the light source at P. Finally, we say that a point on a segment in S is a *shadow point* if a neighbouring polygon point on one side can see P while one on the other side cannot. We are interested in reporting the set of shadow points in a scene consisting of P and S. We will not report the coor-

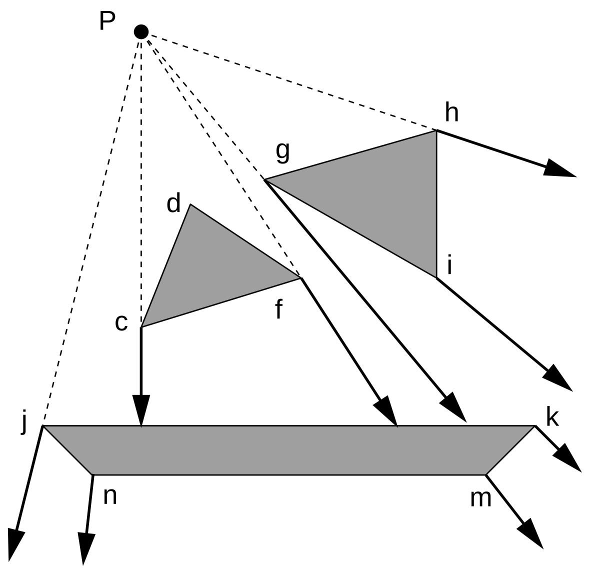

Figure 3: The transparent visibility cycle: each vertex v stores a reference to the edge *below*(v). The reference is shown here by the arrows, called the light rays. If no edge intersects the light ray, a special marker for a circle at infinity is stored.

dinates of these points, instead, we will report the index of the edge e on which it lies, a pointer to the light source at P and the vertex v belonging to a polygon in S that *generated* this shadow point. See Figure 4.

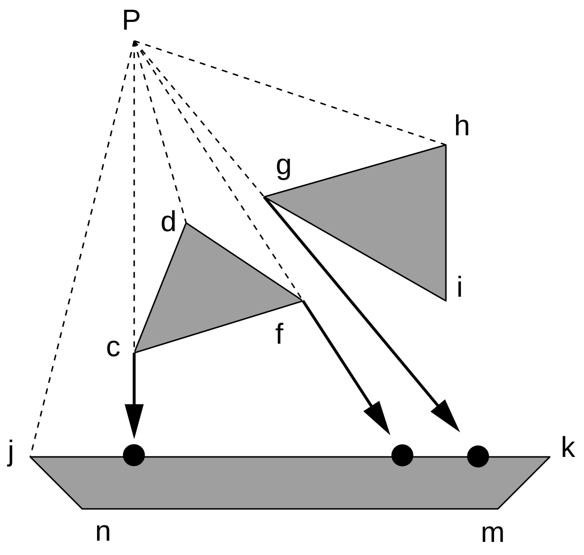

Figure 4: The points identified by the circles are the shadow points in the scene. For each such point, the neighbourhood on one side can see the light source P whereas the neighbourhood on the other side cannot. The shadow points are in one–to–one correspondence with the vertices in the opaque visibility cycle.

We are only concerned with the points that lie on the outside of each polygon in S, thus when we talk about a point on a polygon, we are referring to a point that lies in the vicinity of an edge of a polygon and on the outside.

For a polygon $p \in P$, let *edges*(p) and *vertices*(p) be the set of edges and vertices, respectively, of p. Also, let *polygon*(v) (*polygon*(e)) be the polygon in which a vertex v (edge e) is a member.

We store the shadow points in an array A. There is an entry for each edge e in *edges*(S) and each stores a search tree T. Each node in T stores a vertex index. The vertex with that index casts a shadow from the light source P and an edge adjacent to v. These entries are inserted in T in their sorted order by distance from one of e's endpoints.

After computing the opaque view from point P, we can easily report the shadow points. Let v be a vertex in the opaque visibility cycle and let p be *polygon*(v). We perform the following tests to find the shadow point, if any, resulting from the vertex v. Note that there are at most as many shadow points as there are vertices in the opaque visibility cycle.

- If the two adjacent edges to v in the opaque visibility cycle are both in *edges*(p), then v does not generate a shadow point (e.g., vertex d in Figure 4).
- If one of the two adjacent edges is the infinite circle, we indicate that there is a shadow point on that circle (e.g., vertex j in Figure 4; the light ray is not shown).
- Otherwise, let edge e be the one of the two edges adjacent to v in the opaque visibility cycle that is not in *edges*(p). Insert a shadow point in the tree T (e.g., vertex c in Figure 4).

To generate a list of the shadow points, for each entry in A, we output the list of vertices stored in each edge tree. The resulting set of vertex–edge tuples is the required set of shadow points. The number of shadow points is at most linear in the number of vertices.

7 POINT LIGHT SOURCE OCCLUSION

Generating the set of shadow points in a scene does not tell us which portions of the polygons in S are lit from the light source at P and which portions are not. Imagine that the polygons in S define the plan of a three dimensional model (as in a maze) and that a light source is positioned somewhere in the model within the limits of the extruded walls. We would be interested in building two sets of edge portions in 2D: those visible from P and those that are not, because we can then send both sets to a hardware renderer, one with the light parameter enabled and the other with it disabled. This results in an easy method to render accurately the shadows in such a scene. See Figure 5.

We will start by reporting the set of edges that are in light. We simply scan the opaque visibility cycle and check for each edge if its two adjacent vertices are also adjacent in the scene. If they are, then the edge is entirely in light (e.g., edge gh in Figure 5).

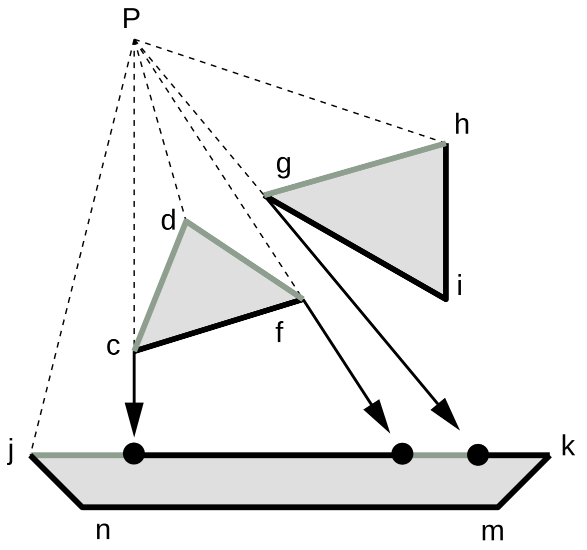

Figure 5: Using the shadow points and the opaque visibility cycle, we can easily determine the edge fragments which are lit by P (dark grey) or in shadow (black).

We then output the *fragments* of edges that are in light. For each edge e in the shadow points array, we output a list of consecutive vertex pairs in the tree T, starting and ending with the endpoints of e. We alternatively store each consecutive pair in the list of fragments in light or in the list of fragments in shadow. We identify whether one of the extreme fragments is in light by searching for the corresponding endpoint vertex in the opaque visibility cycle. The remaining unprocessed edges are in shadow.

8 MAINTENANCE OF THE VISIBILITY FROM A POINT UNDER A MOVING OBJECT

We now consider the case when a polygon in the set S moves in the plane. We will assume that such a motion is given interactively using an input device and that the device generates the motion events at a high rate. This assumption means that the change in the visibility from the viewpoint P is small, which allows us to design an efficient dynamic algorithm.

We will use the ADT's built for both the opaque and the transparent visibility cycles. Also, we will store an edge array E with an entry for each polygon edge in S. This entry stores a doubly–linked list (with head and tail pointers) of the vertex indices whose light ray falls on the edge. For an edge e, let *vlist*(e) be the list of such vertices in sorted order around the viewpoint P.

For a polygon $p \in P$, recall that the sets *edges*(p) and *vertices*(p) are the edges and vertices, respectively, of p. The events we handle are identified as an *elongation* if a light ray becomes longer after the event is processed and as a *shortening* if a light ray becomes shorter. The name of each event also describes whether the light ray that is affected belongs to a *static* polygon, or to a *dynamic* polygon. If a polygon p moves, we detect the following events: (see Figure 6). Due to space constraints, we present the details for the first case only.

I. Elongation of a Static Light Ray: For each edge e in *edges*(p) that is facing P, we check if the light ray of each vertex in *vlist*(e) still intersects e. Assume that p has moved such that the light ray of a vertex $v' \in$ *vlist*(e) no longer intersects e. Of the two vertices adjacent to e, let v be the one closer to the light ray of v'. Perform the following two operations:

- Copy the value of the edge $e' =$ *below*(v) to *below*(v').
- Delete v' from *vlist*(e). This takes constant time since v' is either at the head or the tail of the list.
- If v' is found in the opaque visibility cycle, insert v and e' in it.
- Swap v and v' in the transparent visibility cycle.

II. Elongation of a Dynamic Light Ray: For each vertex v in *vertices*(p), we check if the light ray of v still intersects the same edge. If it does not, then its light ray will "fall" on a new edge e. To determine e, let vertex v' be the one to be crossed by the light ray. Then we have $e =$ *below*(v').

III. Shortening of a Static Light Ray: For each vertex v in *vertices*(p), do the following: Let e' be the edge *below*(v) and let v' be the vertex adjacent to v in *vlist*(e') and in the direction of motion of p. Check if v has intersected the light ray of v'. This case arises when v' is closer than v to P (the other case is handled below).

IV. Shortening of a Dynamic Light Ray: For each vertex v in *vertices*(p), do the following: Let e be the edge *below*(v) and let v' be the vertex adjacent to v in *vlist*(e) and in the direction of motion of p. In this case, v is closer than v' to P (the other case is handled above). Check if the (moving, hence the name) light ray of v is intersected by the vertex v' and update accordingly.

After checking for these four cases and performing the necessary updates on the data structure, the new view from point P is readily available in the opaque visibility cycle. This update takes time proportional to the number of light rays incident to *edges*(p).

9 MAINTENANCE OF SHADOW POINTS UNDER A MOVING OBJECT

We detect each of the preceding four cases and perform one of the following updates.

Elongation of a Static Light Ray: If v' is found in the opaque visibility cycle, delete the shadow point on e and insert it on e'.

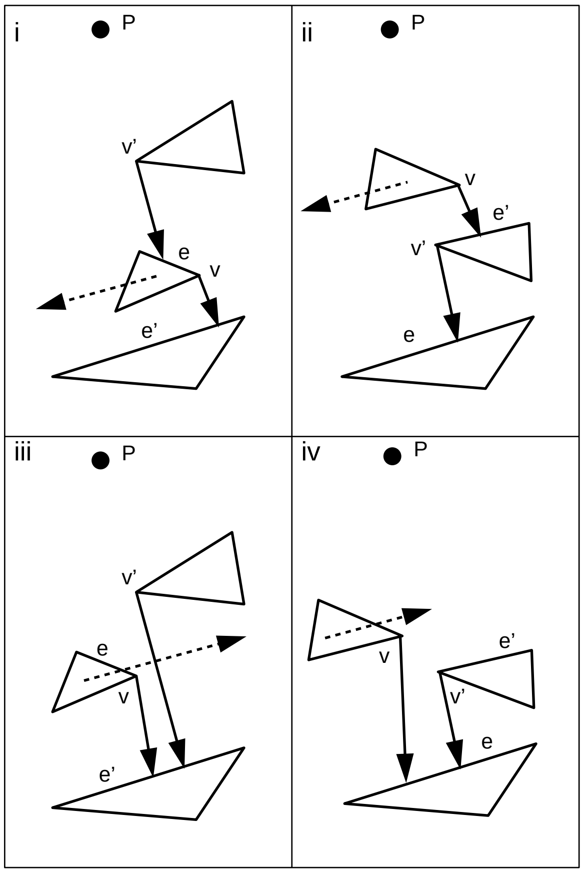

Figure 6: The four cases show the relative position of two vertices and their light rays (arrows). Each case can be detected and a few operations performed to update the view. The dashed arrows show the direction of motion of the polygon.

Elongation of a Dynamic Light Ray: If v is found in the opaque visibility cycle, delete the shadow point on e' and insert it on e.

Shortening of a Static Light Ray: If v' is found in the opaque visibility cycle, delete the shadow point on e' and insert it on e.

Shortening of a Dynamic Light Ray: If v is found in the opaque visibility cycle, delete the shadow point on e and insert it on e'.

If any of the tests above resulted in an update of the shadow points, we maintain the list of edge fragments in light and in shadow by regenerating the two lists of fragments for the edges e and e'.

10 VISIBILITY FROM A LINE SEGMENT

Consider the scene with a set S of polygons in which a viewpoint moves along a line segment L. We are interested in identifying the set of critical points on L. A point is *critical* if the view of the scene in the neighbourhood of the point in one direction is different from the view in the other direction. A solution of the problem in space was described in [Bern94]. We describe here a simplification of that algorithm for the planar case which yields a more efficient solution than in space.

Parameterise L with t such that $t = 0$ at one endpoint and $t = 1$ at the other. We will maintain the transparent visibility cycle H of S as seen from a viewpoint V moving from $t = 0$ to $t = 1$. Let Q be a priority queue that stores the locations along L at which critical points may occur. See Figure 7.

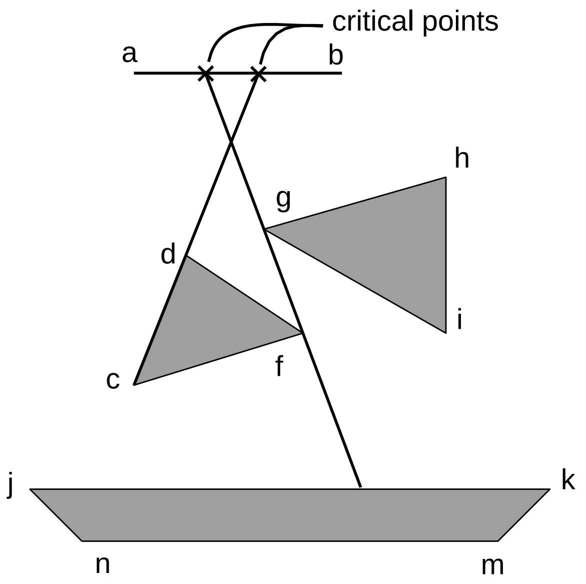

Figure 7: A viewer observes a scene from a line segment ab. As the viewer moves from a to b, the view seen remains the same until the viewer reaches one of the critical points shown. it is possible to use the transparent visibility cycle to detect this change in the view.

Compute both the opaque and the transparent visibility cycles at $V_{t=0}$. Initialise Q by computing the locations t along L at which the line joining each adjacent pair of endpoints in the transparent visibility cycle intersects L and discarding those that do not lie in $[0, 1]$. Each item in Q stores:

- A real number t in $[0..1]$ indicating the position along L at which a critical point may occur.
- The index of two vertices which give rise to that critical point.

For each item in Q with smallest t, we extract it then do the following operations. Let v_1 and v_2 be the two vertices stored in that item. We perform the following operations:

- Swap v_1 and v_2 in the transparent visibility cycle.
- Update *below*(v_1) and *below*(v_2) depending on the relative position of v_1 and v_2 with respect to L.
- Search for v_1 and v_2 in the opaque visibility cycle and update if necessary.

11 LINEAR LIGHT SOURCE SHADOW POINTS

Consider the scene consisting of a set S of polygons in which a linear light source is positioned at the line segment L. We say that a polygon edge in the scene is in shadow if no point on the edge can see any point on the light source L. A polygon edge is in light if all points on the edge can see all the points on the light source L. Finally, we extend the definition of a shadow point as follows. We say that a point on a polygon edge is a shadow point if the structure of the light source L seen from the neighbourhood of the point on one side is different from that seen on the other side. Here also, we do not report the coordinates of the shadow points, but we report instead the indices of the two vertices that gave rise to the shadow point.

It is easy to see that the set of shadow points resulting in such a scene consists of the following two sets of points:

- The shadow points resulting from each endpoint of L.
- A shadow point resulting from each critical point on L.

In the case of a point light source, a shadow point on an edge e was identified by two vertices: the point light source P, and a vertex in S. Two vertices are also needed to identify a shadow point in the case of a linear light source. However, either both vertices are in S or one is in S and the other is an endpoint of the light source L.

For a set S illuminated by a point light source, a single edge *below*(v) identifies a line segment in the plane. The region on one side of this line segment is in light and the region on the other is in shadow. On the other hand, if S is illuminated by a linear light source, there could be more than one line segment incident to v around which the portion seen of L changes. (We describe this more carefully in the next section.) For each vertex v, we store a sequence *belowList*(v) sorted around v. Each entry in the sequence describes the vertex closer or on the light source L defining the boundary, in addition to the edge on which the shadow point falls. See Figure 8.

To determine this set of shadow points, we position a point light source at each endpoint of L and find the set of shadow points. We then compute the set of critical points on L. For each critical point, we determine the corresponding shadow point. We insert this point into *belowList*(v) by sorted angle around v. Of the two vertices $\{v_1, v_2\}$ that give rise to a shadow point on e, the vertex in which *belowList*$()$ is stored is the farther from the light source.

12 LINEAR LIGHT SOURCE OCCLUSION

In a scene S illuminated by a linear light source L with endpoints a and b, we would like to partition the edges

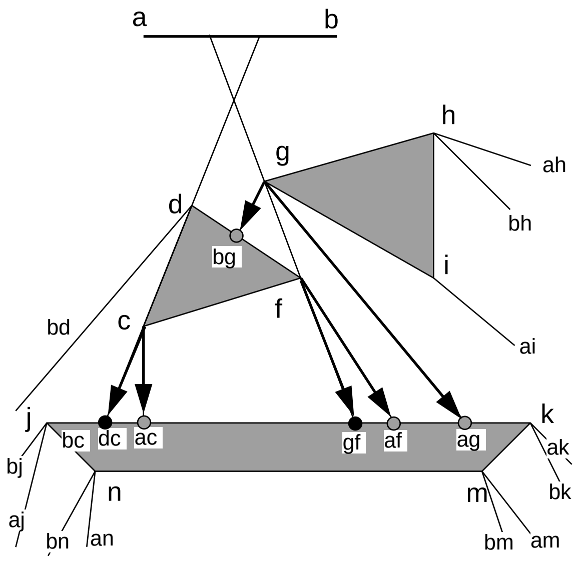

Figure 8: If a linear light source L is positioned at the line segment ab, the two endpoints of L, vertices a and b generate shadow points exactly as if they were point light sources (shadow points in grey). The critical points on ab determine another set of shadow points (black). The union of these sets of shadow points is the set of shadow points that result when this scene is illuminated by the linear light source ab.

in S into fragments such that all points on one fragment see the same portion or portions of the light source. The visible portion of the light source L can be described by a string. In the example shown in Figure 9, the string is empty $(-)$ if L is not visible and it is ab if all of L is visible. The string may be formed by more than one part. For example, the string $ac-db$ means that two portions are visible from L: a portion from a to c and another from d to b. This string gives important information about the computation of the irradiance and its derivative [Heckb92, Arvo94]. We use ideas of incremental updates of views [Gigus91, Stewa94] but describe a technique which uses only string manipulation.

We proceed as follows: given the source view string (for conciseness, we say here "the string") at a vertex of a polygon in S, we would like to determine the strings at all fragments of the polygon. Each edge e in the polygon stores *vlist*(e) in sorted order along e. Each shadow point in *vlist*(e) contains a pair of vertices (one of which possibly belonging to the light source L). If we know the string on one side of the shadow point, it is possible to determine its value on the other side. If the shadow point is defined by xy, then we perform the following tests:

- If the string at hand is also xy, then the result of the update is the empty string and the light source ceases to be visible.
- If the string contains an occurrence of x (denoting a vertex), we replace that occurrence by y and vice–versa. Note that the string cannot contain more than one occurence.
- If the string contains neither x nor y, we append the string xy. This would mean that two (or more) portions of the light source are visible and an object in the scene is occluding the view in between.

If the string is known at a vertex on one polygon, then it can be determined at all points on the polygon: for a vertex v, consider *belowList*(v), the list of shadow points generated. By doing the string substitution described above on that list in order, we can determine the string on one edge of a polygon if it is known on another in the neighbourhood of the vertex v. Similarly, the two lists *belowList*(v) and *vlist*(e), for a vertex v and an edge e, allow us to find the string on one vertex if we know its value at the fragment of an edge. See Figure 9.

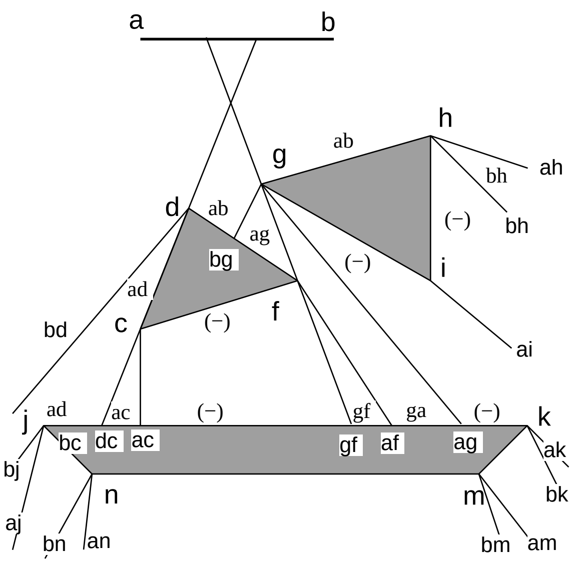

Figure 9: Unlike the case of a point light source in which each edge fragment was marked either *lit* or *in shadow*, the edge fragments that result from illumination by a linear light source are marked by the "signature" of the portion of L that each can see. A string is used to describe this signature. String manipulation suffices to determine the signatures from all edge fragments in the scene if it is known at one vertex in the scene.

13 CONCLUSION

We described a hierarchy of problems to solve visibility and shadow problems in the plane. For each problem in the hierarchy, we gave enough detail so that the reader can conceive an implementation both in a short period of time and using the power of abstraction and encapsulation of an object–oriented language. From a practical point of view, studying these problems in space is more interesting than studying them in the plane. Part of the beauty of this problem domain, however, is that solutions to these problems in space use many of the components of the solutions in the plane [Ghali96a]. To embed the classes and algorithms described in this paper as part of a system handling these problems in space, one has to abstract what constitutes a polygon in the plane. In a 3D system, a polygon would be defined by the intersection of a plane and a polyhedron.

References

[Arvo94] James Arvo. The irradiance Jacobian for partially occluded polyhedral sources. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 343–350. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[Bern94] M. Bern, D. Dobkin, D. Eppstein, and R. Grossman. Visibility with a moving point of view. *Algorithmica*, 11:360–378, 1994.

[Booch94] G. Booch. *Object–Oriented Analysis and Design with Applications*. Addison–Wesley, Readings, Massachusetts, second edition, 1994.

[Drett94] 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.

[Edels83] H. Edelsbrunner, M. H. Overmars, and D. Wood. Graphics in Flatland: A case study. In F. P. Preparata, editor, *Computational Geometry*, volume 1 of *Adv. Comput. Res.*, pages 35–59, London, England, 1983. JAI Press.

[Ghali96a] S. Ghali and A. J. Stewart. A Complete Treatment of D1 Discontinuities in a Discontinuity Mesh. In *Proceedings of Graphics Interface '96*, pages 122–131, San Francisco, CA, May 1996. Morgan Kaufmann.

[Ghali96b] S. Ghali and A. J. Stewart. Incremental update of the visibility map as seen by a moving viewpoint in two dimensions. *Eurographics Workshop on Computer Animation and Simulation*, pages 1–11, August 1996.

[Ghosh91] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM J. Comput.*, 20:888–910, 1991.

[Gigus91] Z. Gigus, J. Canny, and R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):542–551, June 1991.

[Heckb92] P. Heckbert. Radiosity in flatland. *Eurographics*, 11(2), 1992.

[Lisch92] D. Lischinski, F. Tampieri, and D. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics & Applications*, pages 25–39, November 1992.

[Losco97] C. Loscos and G. Drettakis. Interative high–quality soft shadows in scenes with moving objects. *Eurographics*, 16(3), 1997.

[Mulmu91] K. Mulmuley. Hidden surface removal with respect to a moving point. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 512–522, 1991.

[Mulmu94] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[Nishi85] T. Nishita and E. Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 23–30, July 1985.

[Orti96] Rachel Orti, Stephane Riviere, Fredo Durand, and Claude Puech. Radiosity for Dynamic Scenes in Flatland with the Visbility Complex. In *Computer Graphics Forum*, volume 15, pages C237–C248, September 1996.

[Overm88] M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 164–171, 1988.

[Plant86] W. Plantinga and C. Dyer. An algorithm for constructing the aspect graph. In *Proc. 27th Symp. Foundations of Computer Science*, pages 123–131, 1986.

[Pocch96] M. Pocchiola and G. Vegter. The visibility complex. *Internat. J. Comput. Geom. Appl.*, 6(3):279–308, 1996.

[Riviè97] "Stéphane Rivière". *Calculs de visibilité dans un environnement polygonal 2D*. PhD thesis, Université Joseph Fourier – Grenoble I, 1997.

[Stewa94] A. 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, Aug 1994.