Local Approach to Dynamic Visibility in the Plane

Karel Nechvíle, Petr Tobola[1]

Department of Software Systems and Communications
Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno
Czech Republic
{kodl,ptx}@fi.muni.cz

ABSTRACT

This paper proposes the new method for maintaining of the view around a moving point in a static scene in the plane. The scene is composed of n line segments. To maintain the view efficiently, present algorithms avoid construction of the full arrangement of discontinuity lines and use an implicit description instead. On the contrary to previously published papers, we use the structure containing information only about discontinuity lines that are in the proximity of the current viewpoint position. We show how to create this local structure and we give two ways how to apply it in the context of previous work.

Keywords: visibility, discontinuity line, incremental update, duality, dynamic convex hull.

1 Introduction

Consider the following problem: A viewpoint moves in a scene composed of nonintersecting line segments in the plane. It is required to compute and report all visible line segments at successive positions along the viewpoint trajectory. We focus on the *on-line* solution to the problem, where the trajectory is not known in advance. This solution is attractive for present computer systems where the viewpoint path can be guided e.g. by a mouse, a data glove or an infrared tracking system.

As the change in the viewpoint position is small with respect to the scene, there is an amount of similarity between consecutive frames. The *frame-to-frame* coherency makes wasteful to render each frame from scratch, given that most of the information needed has already been computed for the previous frame. This fact enables a precomputing step which later allows quick updates.

The traditional solution is to preprocess the scene in such a way that the depth order can be computed for an arbitrary viewpoint. Binary Space Partitioning (BSP) is well known paradigm [Fuchs80]. The expected size of a 2D BSP tree is $O(n \log n)$ and the tree is rendered using a painter's algorithm [Berg97]. Unfortunately, the cost of traversal in the painter's algorithm can be prohibitive for our purposes.

If we get more familiar with the problem, we find that changes in visibility occur only at some locations in the scene. These places are called *discontinuity lines* or *visual events* and were intensively studied in 2D ([Welzl85, Chaze85, Ghosh87, Overm88]) and 3D computer graphics ([Gigus88, Bern90, Heckb92, Lisch92, Stewa94, Duran97]). For a scene comprised of n segments, the number of discontinuities is between $\Omega(n)$ and $O(n^2)$.

A possible approach to maintain the visibility could be to create a complete arrangement of all discontinuity lines. Such arrangement splits the plane into cells. The visibility is constant (the same for all points) inside these cells, see Figure 1. As the viewpoint moves, we have to track if the viewpoint crosses a cell boundary. After entering a new cell, visibility is updated using the information stored in this cell. However, the number of cells which is $O(m^2)$ for m discontinuity lines discourages us from using that approach for other than small scenes.

[1]This work was supported by the grant 201/98/K041 from GAČR

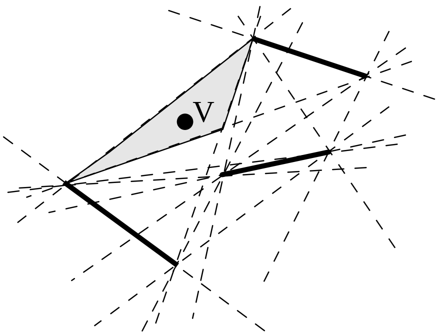

Figure 1: An arrangement of discontinuity lines in the plane.

A more promising way is to form an implicit description of the cell the viewpoint is currently in. There are two recent approaches known to solve the problem, an algorithm of Ghali and Stewart [Ghali96a] and another one of Rivière [Rivie97] (for polygonal scenes). Both algorithms derive their internal structures from the *visibility graph*. Rivière creates then the *visibility complex* [Pocch93], a structure that has the same complexity as the visibility graph, but contains more information. Ghali and Stewart employ a duality concept to gain algorithm efficiency.

We review algorithm of Ghali and Stewart (G&S algorithm) briefly in the following section. In section 3, we describe our data structure for storing only local information about the viewpoint position and demonstrate its utilization in section 4. In section 5, some changes in behaviour of the algorithm are shown when the local approach is used. The last section outlines directions and applications in future.

2 G&S Algorithm

First of all, we shortly introduce the G&S algorithm, because our approach is closely related to it. Full details may by found in original papers [Ghali96a], [Ghali96b]. The algorithm proceeds in the following steps:

1. Build the visibility graph for the given n segments. The nodes in the visibility graph are the segment endpoints. Two nodes are joined by an edge if and only if the line segment connecting the corresponding endpoints does not intersect the interior of any segment. The line supporting the two endpoints is called a *discontinuity line.* This graph can be constructed in $O(m + n \log n)$ time and $O(m)$ space for a graph with m discontinuities [Overm88]. Depending on the scene, m is bounded by $\Omega(n)$ and $O(n^2)$.

2. Given the set of discontinuity lines, preprocess it into a data structure that a viewer traversing the scene can query to determine whether any discontinuity has been crossed. For this purpose, Ghali and Stewart use a duality concept. The *duality transformation* maps a set of points to a set of lines, and vice versa, in a one-to-one manner. We can even do this in such a way that certain properties of the set of points are translated to certain other properties of the set of lines, e.g. the duality can be incidence and order preserving. The plane where the original data sit is called the *primal plane* and the one to which they map is called the *dual plane.*

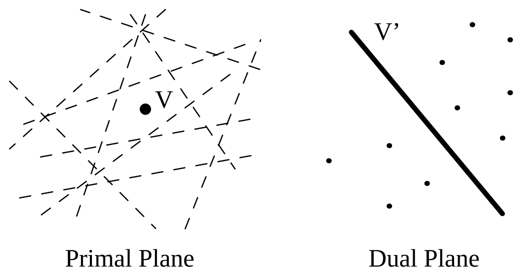

Figure 2: Schematic view of the situation in the primal and dual plane.

Thus, discontinuity lines are transformed into a set of points in the dual plane and the viewpoint V is transformed into a line V'. This line divides the collection of points into two sets, see Figure 2. As the viewpoint moves in the primal plane, the line is changing its position in the dual plane. Answering the question whether the viewpoint crosses a discontinuity line (in the primal plane) equals to the query if any point changes its position with respect to the line (in the dual plane). In order to make this decision quickly, Ghali creates a convex hull of the two sets of points and checks only points on the boundary. After preprocessing step in $O(m \log m)$ time and $O(m)$ space, it is possible to decide if the viewpoint crosses a discontinuity line in $O(\log m)$ or naive $O(k)$ time, where k is number of points on the boundary of the convex hull. Because the number of boundary points is small in practice the $O(k)$ criterion is often sufficient.

3. As long as no discontinuity line is crossed, the set of visible segments remains unchanged. If one is crossed, the data structure has to be updated. In the dual plane,

a point changes from one side of the separating line to the other. In this case, the point must be removed from one convex hull and inserted into the other one. To maintain a dynamic convex hull, Ghali&Stewart use the algorithm of Overmars and van Leeuwen [Overm81] which permits an insert/delete operation at a cost of $O(\log^2 m)$. The set of visible segments is *not always* updated when the viewpoint crosses a discontinuity line, although the convex hulls must be updated to maintain the description of the current cell. We will examine visibility changes in detail later.

3 Local Approach

The algorithm of Ghali&Stewart computes a set of *all* discontinuities to form an implicit description of the current cell. In our view, it uses the *global* information about a scene to extract the local visibility information. Instead, we propose the *local* structure which encodes only discontinuity lines that will be processed in the near future. The structure we suggest is similar to a trapezoidal map used for solving a point location problem [Berg97]. We call it the *topological map*.

3.1 Topological Map

Let us suppose that the scene consists of a set S of n nonintersecting line segments and an initial position of the viewpoint V is given. We introduce a simplification here that will make things easier for us and that is not essential for the rest of the algorithm as we show later. We assume that no two arbitrary endpoints and the viewpoint are colinear. The *topological map* is obtained by drawing two *extensions* from every endpoint of each segment, one extension going towards the viewpoint and the other one going in the opposite direction. The extensions stop when they intersect another segment of S or the viewpoint V. The topological map of S is the subdivision of the plane induced by S, the viewpoint V and extensions. Figure 3 shows an example. We will call faces of the subdivision *topological cells* or simply cells.

We call two cells c_1 and c_2 *adjacent* if they meet along a common extension. Because of our assumption, a cell has at most four adjacent cells, see Figure 4. This fact permits a very simple representation of the topological map based on adjacency. In addition to pointers to neighbouring cells, we store pointers to the left and right adjacent endpoints and pointers to the upper and lower bounding segments. If a cell has only one neighbouring cell on a side, both pointers point to this cell.

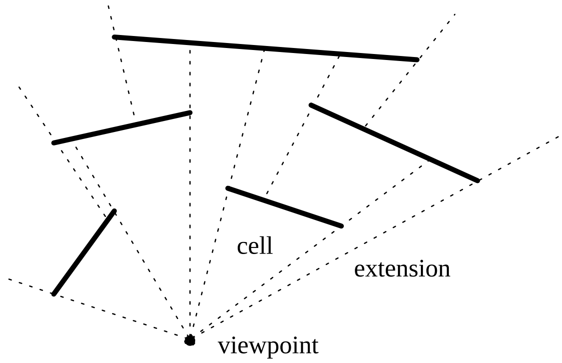

Figure 3: The topological map for a set of line segments.

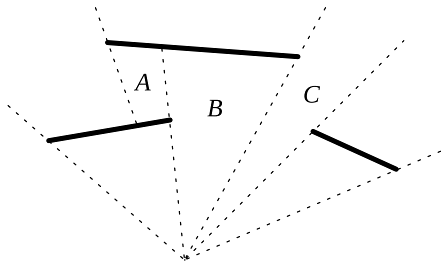

Figure 4: The possibilities in adjacency. Cell A has two neighbouring cells, cell B has three adjacent cells and cell C is adjoined to four other cells.

There are three types of cells, cells meeting at the viewpoint, interior cells inside the scene and open cells on a border. Obviously, if the cell touching the viewpoint is bounded from the upper site, the boundary is made of a visible part of a line segment. To maintain the visible line segments, we can record a cell adjoined to the viewpoint and using the adjacency obtain remaining cells. Due to our assumption, it can be easily shown that there is exactly $3n$ cells for n nonintersecting line segments. Using a radial sweep algorithm around the viewpoint, the topological map can be made at a cost of $O(n \log n)$ time and $O(n)$ space.

In the following section, we show how to maintain the visibility for a moving viewpoint using the topological map. We also justify our assumption about the endpoint positions.

3.2 Discontinuity lines and changes in visibility

Each cell of the topological map has two adjacent endpoints. If we connect these endpoints with a line, we create the *discontinuity line* arising from the cell, Figure 5. As the number of cells is linear, the number of discontinuity lines we get is also linear. They compose a subset of all discontinuity lines encoded within the visibility graph.

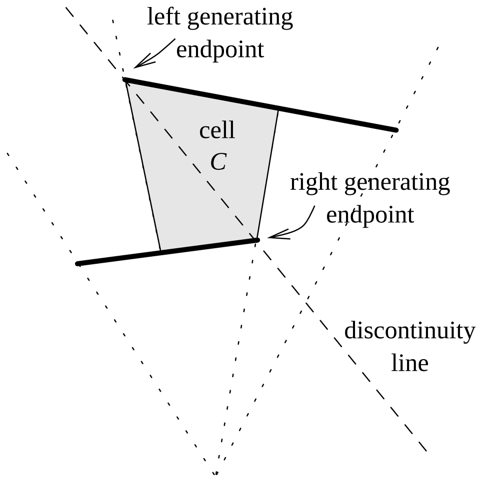

Figure 5: Discontinuity line generated by the cell C in the topological map.

When the viewpoint begins to move, we have to change the topological map accordingly. Fortunately, every movement does not need an update, only a movement changing the topology. Such situation arises when the viewpoint crosses some of the discontinuity lines. Crossing a discontinuity line corresponds to the change in the mutual position of segment endpoints in the generating cell. The number of cells remains constant after this change. When the viewpoint crosses discontinuity line generated by a cell, we update both adjacent cells and their corresponding discontinuity lines. There is only a small number of possibilities how to alter the topological map and they are all localized, see Figure 6.

We argue about the assumption that no two endpoints are collinear with the viewpoint now. If the assumption was not fulfiled, the cell might have an arbitrary number of adjacent cells. In such a case, we would lose simplicity of the data representation. However, we use the topological map in order to keep up the visibility from the viewpoint. Then, if two endpoints are in a cover, we see only the first one. Therefore, we can shift a bit the viewpoint in the right direction so that no two endpoints lie on a line and the correct visibility is conserved.

4 Using Local Approach in G&S Algorithm

In this section it will be shown how to employ the topological map in context of the G&S algorithm. The construction of a topological map

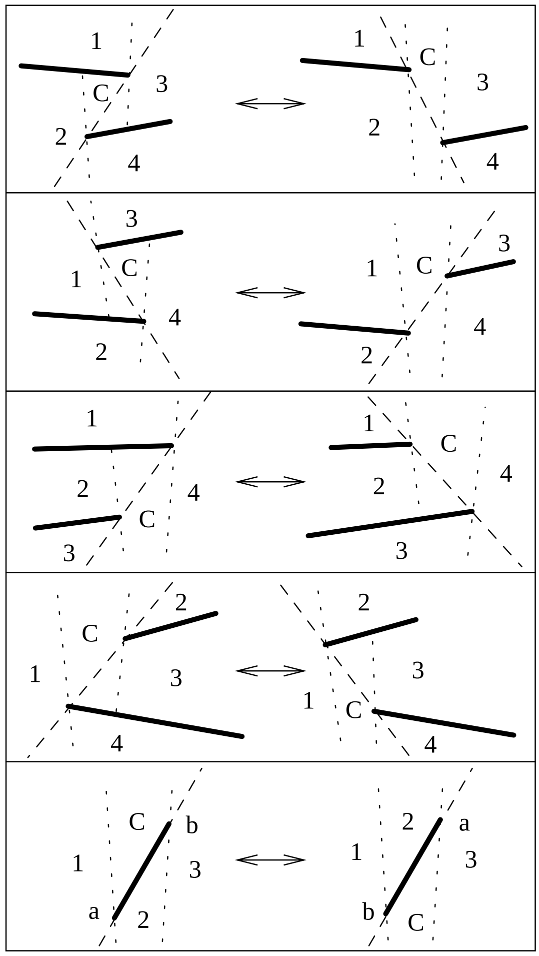

Figure 6: Possible changes in the topological map when crossing the discontinuity line generated by the cell C. Adjacent cells are labeled by numbers.

corresponds to the first step of the algorithm described in the section 2. Now, we demonstrate how to perform the remaining steps.

4.1 With the dual transformation

An obvious way how to do so is simply to follow the Ghali's approach. We dualize the discontinuity lines that are encoded in the topological map. Thus, we get a set of $O(n)$ points in the dual plane. The viewpoint is also dualized and the line we get splits the points into two sets. We make a convex hull for each set and use it to detect if some point changes its position with respect to the line. When a point crosses the line, we update the structures. To maintain the dynamic convex hulls, we apply an algorithm of Overmars and van Leeuwen [Overm81].

In Ghali's algorithm, the set of points in the dual plane is static and the convex hulls change only. The number of points is in $\Omega(n)$ and $O(n^2)$. In our approach, positions of points as well as the convex hulls are both changing. Note however, the number of points is constant and it is always in $O(n)$. Both algorithms need a time of $O(\log^2 n)$ to perform an update.

4.2 Without the dual transformation

We can use an alternative way to preserve the relation between the viewpoint and discontinuity lines. The convex hull is closely related to an *intersection of halfplanes* just due to the duality transformation [Berg97]. Thus, we can avoid duality transformation by solving the problem as a common intersection of a set of halfplanes in the primal plane.

A discontinuity line splits the plane into two halfplanes and the halfplane considered is that one the viewpoint lies in. In order to maintain the common intersection of halfplanes for a dynamic environment, we apply an algorithm presented in [Overm81]. The algorithm permits the construction of necessary data structures in $O(n \log n)$ time, using a space of $O(n)$. Answering the question if the new viewpoint position is inside the common intersection takes $O(\log n)$ time. As the boundary of the intersection is very small in practice, it is sufficient to compare the position against the boundary explicitly. After crossing a boundary, an update of the data structure is feasible at a cost of $O(\log^2 n)$ time. The approach to dynamic maintenance of intersection of halfplanes is in fact similar to that one of maintaining the dynamic convex hull. In this case the lines are recursively sorted by their slope, for details see the original paper [Overm81].

5 Frequency of Updates

When comparing the cost of creation and update, both algorithms look similar, but there is a difference we want to highlight. We have paid little attention to the arrangement yet. Now, we carefully examine the discontinuity lines involved.

Let us suppose the viewpoint is situated somewhere in the scene and let us consider only the visible segments. The Ghali and Stewart's algorithm is derived from the visibility graph. Within the visibility graph, all discontinuity lines connecting mutually visible endpoints of segments are contained. As we move "inside" the scene, no changes in topology actually occur. Apart from this, we have to update the structure every time we cross a discontinuity line. An extreme example would be a set of circularly arranged line segments, see Figure 7a. When using the topological map, only relevant lines are actually present, see Figure 7b. The difference between the number of lines crossed by each algorithm is dependent upon the scene and can not be exactly evaluated.

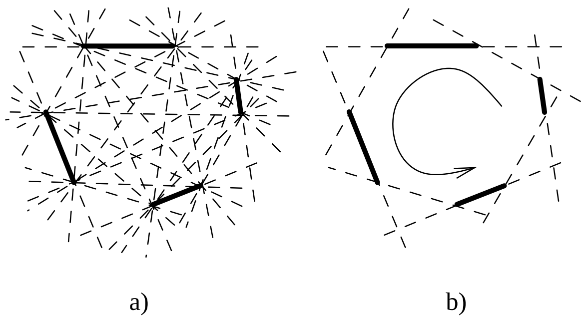

Figure 7: Discontinuity lines contained in the visibility graph (a) and in the topological map (b) for the viewpoint moving inside.

6 Implementation and Comparison

We have implemented the algorithm in C++ with help of the [LEDA] library. To avoid numerical problems, we adopted the following strategy. If all the input points have integer coordinates (or rational), then all computations are performed with rational numbers, i.e. exactly. If more than one discontinuity line has been crossed during the mouse motion, the distance of intersection points to the old viewpoint in the primal plane is compared to determine their order. In order to report the visibility cycle, we store the pointer to a cell adjoined to the view point. Using the adjacency information stored in the cells, we can visit all visible segments.

It is difficult to exactly compare the algorithm presented here to the algorithm of Ghali and Stewart. Their algorithm relies on the fact the set of discontinuity lines is static. That makes possible to use a simplified version of the dynamic convex hull algorithm. In our case, we have to employ a "full version" of the CH-algorithm. On the other hand, the simplified version limits further enhancements of G&S algorithm to dynamic environments.

As was stated in the previous section, both algorithms differ in a number of discontinuity lines involved. As the number can not be determined theoretically, we measured it in practise. We have performed tests with scenes comprising from 10 to 500 segments and a random walk through the environment, see Figure 8. We have found that the number is connected with the *average visibility density* which we defined as the ratio of the average number of visible segments during the path to the whole number of segments. As the average visibility density grows, the visibility graph contains more mutually visible endpoints and the G&S algorithm has to cross more discontinuity lines than algorithm based on topological map. In our tests, the G&S algorithm crosses from 10% to 50% more discontinuity lines than the presented algorithm, typically about 20%. When the path contains some loops or swirls, the difference even grows.

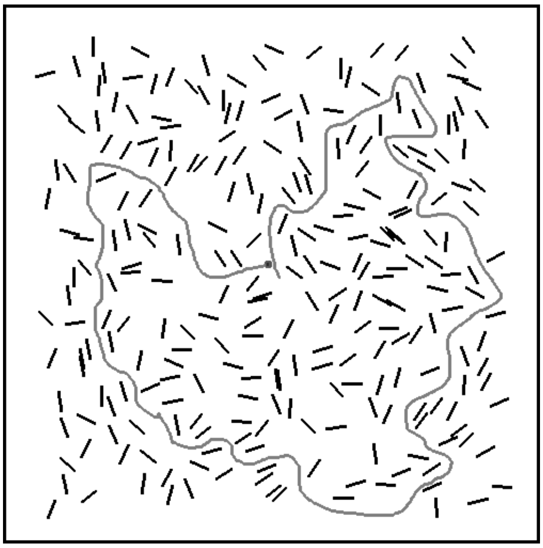

Figure 8: A screenshot of test scene with 256 input segments and a motion path

We have also compared the storage requirements of both algorithms. We recall, algorithm based on topological map is always $O(n)$, the complexity of the G&S algorithm is between $\Omega(n)$ and $O(n^2)$. Our tests indicated that the average visibility density is closely related to a multiplicative factor between memory requirements of both algorithms. For example, if the density is 20%, the G&S algorithm has to store about 20 times more discontinuity lines than the presented algorithm. This again follows from the fact the average visibility density corresponds to the number of discontinuity lines contained in visibility graph.

The time needed to initialize the structures was typically very small and we found it insignificant.

7 Conclusion and Future Work

We have shown a "local approach" to maintain a set of relevant discontinuity lines for visibility with a moving point of view. We have pointed out two ways how to use it in the context of original algorithm and stated their complexity.

Currently, we study if it is possible to use the local approach within the visibility complex paradigm too. We are also looking for some ways how to deal with a dynamic environment, where segments move in the scene, eventually are inserted into or deleted from. Possible extensions to 3D are also explored.

REFERENCES

[Berg97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, August 1997.

[Bern90] M. Bern, D. Dobkin, D. Eppstein, and R. Grossman. Visibility with a moving point of view. In David Johnson, editor, *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 107–117, San Francisco, CA, USA, January 1990. SIAM.

[Chaze85] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. In Joseph O'Rourke, editor, *Proceedings of the Symposium on Computational Geometry*, pages 135–146, Baltimore, MD, June 1985. ACM Press.

[Duran97] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 89–100. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[Fuchs80] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. volume 14, pages 124–133, July 1980.

[Gigus88] Z. Gigus, J. Canny, and R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. In *Second International Conference on Computer Vision (Tampa,, FL, December 5–8, 1988)*, pages 30–39, Washington, DC,, 1988. Computer Society Press.

[Ghosh87] S. K. Ghosh and D. M. Mount. An output sensitive algorithm for computing visibility graphs. In Ashok K.

Chandra, editor, *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 11–19, Los Angeles, CA, October 1987. IEEE Computer Society Press.

[Ghali96a] S. Ghali and A. J. Stewart. Incremental update of the visibility map as seen by a moving viewpoint in two dimensions. In *Seventh International Eurographics Workshop on Computer Animation and Simulation*, pages 1–11, August 1996.

[Ghali96b] S. Ghali and A. J. Stewart. Maintenance of the set of segments visible from a moving viewpoint in two dimensions. In *Proceedings of the Twelfth Annual Symposium On Computational Geometry (ISG '96)*, pages V3–V4, New York, May 1996. ACM Press.

[Heckb92] P. Heckbert. Discontinuity meshing for radiosity. *Third Eurographics Workshop on Rendering*, pages 203–226, May 1992.

[Lisch92] D. Lischinski, F. Tampieri, and D. P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, November 1992.

[Overm81] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166–204, October 1981.

[Overm88] M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proceedings of the Fourth Annual Symposium on Computational Geometry (Urbana-Champaign, IL, June 6–8, 1988)*, pages 164–171, New York, 1988. ACM, ACM Press.

[Pocch93] M. Pocchiola and G. Vegter. The visibility complex. In ACM-SIGACT ACM-SIGGRAPH, editor, *Proceedings of the 9th Annual Symposium on Computational Geometry (SCG '93)*, pages 328–337, San Diego, CA, USA, May 1993. ACM Press.

[Rivie97] S. Rivière. Dynamic visibility in polygonal scenes with the visibility complex. In *Proceedings of the 13th International Annual Symposium on Computational Geometry (SCG-97)*, pages 421–423, New York, June4–6 1997. ACM Press.

[Stewa94] A. J. Stewart and S. Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 231–238. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[Welzl85] E. Welzl. Constructing the visibility graph for n-line segments in $O(n^2)$ time. *Information Processing Letters*, 20(4):167–171, May 1985.

[LEDA] LEDA library, K. Mehlhorn, S. Näher, M. Seel, and C. Uhrig. `http://www.mpi-sb.mpg.de/LEDA`.