3D Visibility Graph

Mojtaba Nouri Bygi*

Department of Computer Engineering
Sharif University of Technology

Mohammad Ghodsi†

Department of Computer Engineering
Sharif University of Technology
IPM School of Computer Science, Tehran

Abstract

The visibility graph is a fundamental geometric structure which is useful in many applications, including illumination and rendering, motion planning, pattern recognition, and sensor networks. While the concept of visibility graph is widely studied for 2D scenes, there is not any acceptable equivalence of visibility graph for 3D space.

In this paper we explain some reason for this absence. Then we try to find a new way to define geometric structure in 3D space. Following our new way, we easily define a new structure called 3D visibility graph which we believe is the natural way to extend visibility graph in 3D scenes. We show how to compute it in an acceptable time.
keywords: computational geometry, visibility graph, 3D visibility.

1 Introduction

Problems involving the visibility of objects have arisen in several areas of computer science, for examples, graphics, VLSI layout, motion planning, and computational geometry. Frequently the underlying structure of the visibilities is critical and a graph can be created that condenses this structure information into a more usable form.

The *visibility graph* is a fundamental geometric structure useful in many applications, including illumination and rendering, motion planning, pattern recognition, and sensor networks.

Consider the path planning problem in a 2D polygonal scene. The *visibility graph* is defined as follows: The nodes are the vertices of the scene, and an arc joins two vertices A and B if they are mutually visible, i.e. if the segment [AB] intersects no obstacle. It is possible to go in straight line from A to B only if B is visible from A. The start and goal points are added to the set of initial vertices, and so are the corresponding arcs (see Figure 1). Only arcs which are tangent to a pair of polygons are necessary.

It can be easily shown that the shortest path between the start point and the goal goes through arcs of the visibility graph. The rest of the method is thus a classical graph problem.

This method can be extended to non-polygonal scenes by considering bitangents and portions of curved objects. In the latter case we will have a *tangent visibility graph (TVG)*. The set of vertices in this graph is $\mathcal{O}$, the convex objects of the scene. Furthermore any common tangent of two objects $O_1, O_2 \in \mathcal{O}$ whose endpoints can see each other correspond to an edge $\{O_1, O_2\}$ of the TVG. Both visibility graph of polygonal scene and tangent visibility graph

*nouribaygi@ce.sharif.edu

†ghodsi@sharif.edu, This author's work has been partly supported by IPM School of CS (contract: CS1385-2-01).

of smooth object can be computed in optimal $O(k + n \log n)$ time. Here n in the number of polygons or objects and k is the size of visibility graph.

In this paper we review the main concepts of visibility graph in section 2.1. In section 2.2 we outline the most famous algorithms for building visibility graph and in section 2.3 we enumerate some of its application in computational geometry. In section 3 we specify some of the reasons that lead to complexities of visibility relations in 3D. In section 4 we introduce a new structure called 3D visibility graph that plays the role of visibility graph in 3D scenes. In section 4.5 we give an algorithm for building 3D visibility graph that runs in $O(n^3 \log n)$ time.

2 Preliminaries

In this section we review the visibility graph an its construction algorithms and applications.

2.1 Visibility Graph

The concept of a visibility graph is widely studied in computational geometry.

Consider a collection $\mathcal{O}$ of pairwise disjoint objects in the plane. We are interested in problems in which these objects arise as obstacles, either in connection with visibility problems where they can block the view from another geometric object, or in motion planning, where these objects may prevent a moving object from moving along a straight line path. The visibility graph is a central object in the context of these problems. For polygonal obstacles the vertices of these polygons are the nodes of the visibility graph, and two nodes are connected by an arc if the corresponding vertices can see each other. Lee described the first non-trivial algorithm for computing the visibility graph of a polygonal scene with a total of n vertices in $O(n^2 \log n)$ time [2]. Ghosh and Mount presented an optimal $O(k + n \log n)$ algorithm, where k is the number of arcs of the visibility graph [13].

In the case of smooth convex objects, the visibility graph is referred as *tangent visibility graph (TVG)*. The set of vertices of this graph is $\mathcal{O}$. Furthermore any common tangent of two objects $O_1, O_2 \in \mathcal{O}$ whose endpoints can see each other correspond to an edge $\{O_1, O_2\}$ of the TVG. (Note that there are at most 4 edges between two vertices.)

2.2 Construction Algorithms

In this section we review some algorithms developed for computing the visibility graph of a 2D scene.

D.T. Lee in his 1978 Ph.D. dissertation [11] wrote about the first nontrivial solution to the visibility problem running in $O(n^2 \log n)$ time. In the mid-to-late 1980s a series of $O(n^2)$ papers appeared. In 1985, E. Welzl described a technique based on an arrangement of the dual of the vertices followed by a topological sort to order the vertex pairs in $O(n^2)$ time [12]. This technique is used in other computational geometry problems as well. Welzl's technique requires $O(n^2)$ working space. It works for a set of line segments and can be adapted for sets of polygons. Edelsbrunner and Guibas later improved the working storage of the topological sweep to $O(n)$ [14]. About the same time, Asano, et al. [15] offered two other versions with arrangements also requiring $O(n^2)$ space: the first via triangulation and the second via scan-lines and segment splitting. These techniques construct the polar order one vertex at a time as opposed to Welzl's technique that produces a good permutation (not strictly sorted, but good enough) among all vertex pairs at once. Asanos technique can also handle dynamic updates in $O(n)$ time. In 1988, a paper by Overmars and Welzl describes yet another $O(n^2)$ technique that does not need to calculate the dual arrangement and uses only $O(n)$ working space. The paper also describes a second algorithm running in $O(|e| \log n)$ time and $O(n)$ space, where $|e|$ is the number of edges in the visibility graph. It is efficient for sparse graphs, i.e. when $|e| = O(n)$. Of course, all of the $O(n^2)$ algorithms are optimal when the graph is dense, in other words when $|e| = O(n^2)$.

Towards the end of the appearance of the $O(n2)$ papers, two output-sensitive approaches became known [13, 16]. Ghosh and Mount show a planar-scan technique using triangulation and funnel splits to achieve $O(|e| + n \log n)$

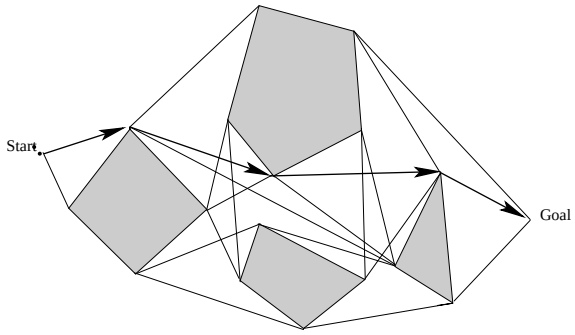

Figure 1: Path planning using the visibility graph.

time bounds [13]. The other work by Kapoor and Maheshwari essentially achieve the same time bounds using corridors which are based on a triangulation dual. With both of these, the time is basically bounded by the number of edges in the visibility graph, but it can be dominated by the time it takes to triangulate (which optimally is $O(n \log n)$ for a polygon with simple polygonal holes). Thus, both of these approaches are optimal for graphs with density as low as $|e| = O(n \log n)$ but no lower.

Here we represent three different algorithms for computing the visibility graph of a scene consist of n polygons. First we describe naive (i.e. trivial) approach which runs in $O(n3)$ time. After that, Lees algorithm is described which runs in $O(n^2 \log n)$ time, followed by Overmars and Welzl's method (being one of the more elegant $O(n^2)$ algorithms).

2.2.1 Naive Algorithm

A simple solution to the problem would be to just look at every edge to see if it blocks/interferes with a given pair of vertices. If none interfere, then the two vertices are visible to each other (otherwise not). Of course, to produce the entire visibility graph, the procedure loops through every pair of vertices. The time analysis is simple also: there are $\binom{n}{2}$ pairs of vertices which is $O(n^2)$ and there are $O(n)$ edges (one for every vertex) so this means the total time is $O(n^3)$. As for storage, the algorithm requires $O(n)$ working space (at least to store the input), and if the visibility graph is stored - not just reported - then it requires $O(|e|)$ memory.

2.2.2 Lee's Algorithm

The algorithm attributed to D. T. Lee represents the first nontrivial solution running in $O(n^2 \log n)$ time [11]. The basic idea is simple: for each vertex, sort the other points in angular order around it, then visit each one keeping track of the order of intersected edges made by the scan-line. If the visited point is associated with the first edge in this ordered list, then it can be reported. Otherwise, it must be obscured by some other edge appearing before it (with respect to the center) and so would not be reported. Of course, the edge list must handle inserts and deletes in $O(\log n)$ time which means using optimal sorting (of which many are available).

Figure 2 shows the intuitive idea. The edge-list here would be $\{5, 2, 1, 4, 3\}$ - the order of intersecting edges from the center along the scan-line. Of course, in reality, the scan-line only stops at vertices (not in the middle of edges).

What happens at each vertex visit depends on the polygonal edges associated with that vertex. There may be two inserts, two deletes, or an insert and a delete. Figure 3 shows these situations with vertices marked a, b, and c with edges marked 1-10. Collinear points are handled as follows: if several points lie along the same scan-line, the order is determined by the distance from the center. In figure 3, vertex a would be visited, followed by vertex b, followed by vertex c. Before a is visited, the edge-list would be $\{5, 9, 10, 3, 6, 2\}$. When a is handled, both its edges are deleted, so the edge-list afterwards would be $\{5, 3, 6, 2\}$. When b is handled, both its edges are inserted, so the edge-list be-

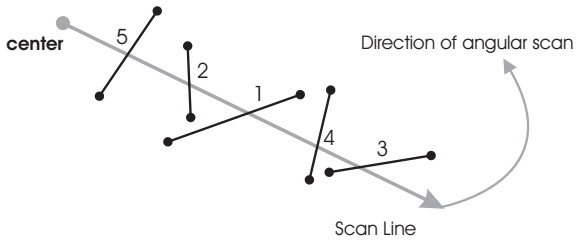

Figure 2: Example of Lee Scan with Edge-List.

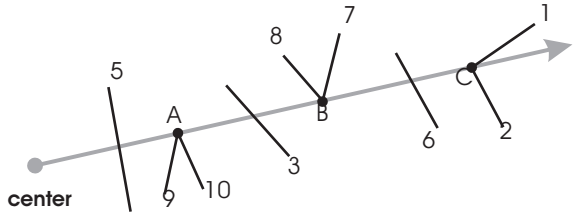

Figure 3: Basic Cases in the Lee Scan.

comes $\{5, 3, 8, 7, 6, 2\}$. When c is handled, one edge is deleted and the other is inserted. The edge-list at the end would be $\{5, 3, 8, 7, 6, 1\}$.

Time analysis: there are $(n-1)$ vertices to be visited for each of n^2 centers. At each of the $(n-1)$, it takes $O(\log n)$ for the search/insert/delete, thus making the time for one scan $(n-1)O(\log n) = O(n \log n)$. The time for all n scans would then be $nO(n \log n) = O(n^2 \log n)$.

Space analysis: in the worst case, there may be $O(n)$ edges in the edge-list at any one given time, but no more. The angularly sorted list also requires $O(n)$ storage during one scan, but can be freed after the particular center has completed. Of course, in order to store the visibility graph, it takes $O(|e|)$ space.

2.2.3 Overmars and Welzl's Algorithm

Welzl originally published a paper[12] describing a technique based on a topological sort of the dual arrangement of segments in a plane. Because it effectively sorts all $\binom{n}{2}$ pairs of vertices, it runs in $O(n^2)$ time (as opposed to Lees $O(n^2 \log n)$ algorithm which scans one vertex at a time with a sort of all other points at each step). The space required is $O(n^2)$. This was later improved by Edelsbrunner and Guibas to $O(n)$ space [14]. Asano, et al. [15] has one version of this based on triangulation and set-union and another based on scan-lines and splitting. The Overmars and Welzl paper [17] represents a practical version without using dualization[1]. Instead, it is based on the concept of rotation trees.

The idea is simple: for each vertex, a scan-line is kept which runs from - $\pi/2$ to $\pi/2$ hopping from vertex to vertex in its path. During the main loop, it appears that all of the scan-lines are proceeding simultaneously. In fact, there are exact rules about determining the next vertex to process, and some vertices may finish their scan before others.

To understand the rules about finding the next vertex, the rotation tree must be understood. A rotation tree is a rooted planar tree where each vertex is a node and points to its parent. There are two special nodes: $+\infty$ and $-\infty$, where $-\infty$ is infinitely below and just to the right of all regular points, and inf is infinitely above and just to the right of all regular points. Initially, all vertices point to $-\infty$ as their parent and $-\infty$ points to $+\infty$. Also stored is the rightmost child (if a node is a parent), and its right and left siblings (if they exist). The ordering of children is by slope: the one with the smallest slope is the leftmost.

The loop that examines all pairs simply takes the rightmost leftmost leaf as the next segment to process and then reattaches it to the tree (while maintaining the property of being a rotation tree). It can reattach to the left of its parent or to the tangent of the chain above it. When a vertex attaches to $+\infty$, it is finished. The loop continues when all points have attached to $+\infty$.

2.3 Applications of the Visibility Graph

The visibility graph problem itself has long been studied and has been applied to a variety of areas. A common use for it has been for finding the shortest path[1]. Exploiting the fact that the shortest path consists of arcs of the visibility graph, one can find the shortest path by running Dijkstra's algorithm [3] on it. The shortest path has been used in robot motion planning. This was identified in 1979 in Lozano-Perez and Wesley's work [5]. The visibility graph can also be used to solve the art gallery problem by finding the minimum dominating set of the visibility graph (NP-hard). More recently, visibility has been used in pursuer-evader problems. Finally, the visibility complex, which contains more information than the visibility graph, has been used in illumination problems [9].

3 2D Visibility versus 3D Visibility

Things are very different when we turn to 3D scenes. The path planning method described above does not generalize simply to 3D where the problem has been shown to be NP-complete [6, 7]. Furthermore, in 3D, the term "visibility graph" often refers to the abstract graph where each object is a node, and where arcs join mutually visible objects. This is however not the direct equivalent of the 2D visibility graph.

We enumerate here some points which make that the difference between 2D and 3D visibility can not be summarized by a simple increment of one to the dimension of the problem. This can be more easily envisioned in line space. Recall that the atomic queries in visibility are expressed in line-space (first point seen along a ray, are two points mutually visible?).

First of all, the increase in dimension of line-space is two, not one (in 2D line-space is 2D, while in 3D it is 4D). This makes things much more intricate and hard to apprehend.

A line is a hyperplane in 2D, which is no more the case in 3D. Thus the separability property is lost: a 3D line does not separate two half-space as in 2D.

A 4D parameterization of 3D lines is not possible without singularities [4].

Visual events are simple in 2D: bitangents lines or tangent to inflection points. In 3D their locus are surfaces which are rarely planar (IEEE or visual events for curved objects) [4].

All these arguments make the sentence the generalization to 3D is straightforward a doubtful statement in any visibility paper.

4 3D Visibility Graph

As stated in section 3, we can not simply extend the concept of visibility graph from 2D to 3D. The problem is that the traditional definitions of visibility cannot be used for defining a meaningful and applicable structure as visibility graph. For one thing, the number of rays between two objects might be countless. For another, it seems impossible to represent the visibility relations of a 3D scene in a planar graph.

These problems lead us to reconsider the basic concepts of visibility in 3D space. In 2D space, we say that two points are mutually visible or see each other if there is a straight line not intersecting any other part of the configuration from one object to the other. If we want to use this concept in 3D, instead of using the line, we must use the plane. The reason is that the main property which explains why 3D visibility is much harder than in 2D, is *separability* property: In 2D, a line separates the plane into two half-planes. No such property holds in 3D

[1] Some work has been done for finding the partial visibility graph where only the tangents around obstacles are included since the shortest path would not need other visibility edges to the obstacle [10].

because lines are no longer hyperplanes. This leads us to considering planes: in 3D, planes have this property, i.e. each plane separates the space into two half-space.

4.1 Some Definitions

Now we replace the role of the plane in 3D with that of line in 2D and change some basic definitions in visibility. first we review these definitions in 2D scenes:

- A ray is a half-line starting from a point
- A segment is a part of line bounded with two points in that line
- The view from a point in some direction is the first object intersecting the line starting from the point in the given direction.
- We say two points are mutually visible if the line segment defining by them are not intersected with any objects except possibly at the end points.

Now we give what we think is a suitable definition for visibility in 3D. As we said we try to replace the role of line in 2D with plane in 3D:

- A *pseudo-ray* from point p in the direction of plane A is an angle surrounded by two half-line in A starting at p. Notice that this pseudo-ray is no longer unique.
- We say that a pseudo-ray can *see* an object if the plane of the pseudo-ray intersect the object and the intersecting section lies within the angle of the pseudo-ray.
- A *pseudo-segment* in a plane is a simple polygon in that plane. We can imagine this pseudo-segment as the region surrounded by some pseudo-rays (angles). In the simplest form, a pseudo-segment is a triangle in the plane, consisting of three pseudo-rays with mutual common edges. In this case, another way to distinguish a pseudo-segment is by giving its ordered vertives. Notice that a line is a special pseudo-segment consisting of two pseudo-rays (angles) that lie on each other.
- We say that some points are mutually visible if they are all lie in a same plane and the pseudo-segment defined by them does not intersect with any object except possibly at these points.

We will use the above definitions for building our new data structure.

4.2 Pseudo-graph

As we mentioned before, because of intricate relations, it is not possible to represent the visibility relations of a 3D scene with an ordinary graph.

We introduce a new structure, called the *pseudo-graph*, which like an ordinary graph consists of vertices and edges, but each edge of it connects three vertices of the pseudo-graph. We can think of edges of a pseudo-graph as pseudo-segments connecting its vertices.

4.3 3D Visibility Graph

Now we define 3D tangent visibility graph as follows. Consider a collection $\mathcal{O}$ of pairwise disjoint smooth convex objects in a 3D scene. We define an structure called pseudo-graph, witch like a ordinary graph consists of vertices and edges, but each edge connects three vertices of the pseudo-graph. The set of vertices in this graph is $\mathcal{O}$, the convex objects of the scene. Any common tangent plane of two objects $O_1, O_2, O_3 \in \mathcal{O}$ whose tangent points are mutually visible correspond to an edge $\{O_1, O_2, O_3\}$ of the pseudo-graph. Note that there are at most 8 edges between three vertices.)

The size of 3D visibility graph of n objects, is proportional to the number of edges of it and is limited by $O(n^3)$ and $\Omega(n)$.

4.4 Application of 3D visibility Graph

We believe that this pseudo-graph has most of the properties that visibility graph is reputed for. For example consider the convex hull of a set of 3D objects. As we know, in 2D, the convex hull of a set of objects lies on the visibility graph of those objects. It can be easily

shown that the same property holds for the 3D visibility graph.

4.5 Construction

Here we present a construction algorithm for 3D visibility graph which runs in time $O(n^3 \log n)$. It is mainly an extension of Lee's algorithm for building visibility graphs to 3D scenes. For each pair of objects O_1 and O_2, sort all the planes which are tangent O_1 and O_2 and one other object in angular order with respect to a fixed plane tangent to O_1 and O_2. This is equivalent to sweeping the space with a rotating plane while remaining tangent to O_1 and O_2 and visiting each of the objects when the sweep-plane becomes tangent to it. We keep track of the order of intersected objects made by the sweep-plane. Like the Lee's algorithm, we maintain an ordered list which has current tri-tangents. When a new tri-tangent plane appears, we check it with other tri-tangents in the list and we insert it or remove it from the list.

The time needed for sorting the tangent planes is $O(n \log n)$, as there are $O(n)$ of these planes. Each of $O(n)$ inserting and removing operations takes $O(\log n)$ time. So each sweeping process takes $O(n \log n)$ time. As there are $O(n^2)$ pairs of objects, the total algorithms takes $O(n^3 \log n)$ time.

Although the algorithm given above is not naive, it is not optimal. We note that it might be possible to improve the running time to be $O(k + n^2 \log n)$, where k, $O(n^3)$, is the number of visibility edges of pseudo-graph, by using pseudo-triangulation based on the work of [10].

5 Conclusion

After reviewing the concept of visibility graph in 2D, we defined 3D visibility graph, which is an structure that plays the role of visibility graph for 3D scenes. We gave an algorithm for building it in time $O(n^3 \log n)$. We believe that 3D visibility graph can be used in many 3D geometric problems. For an example observe that the convex hull of a set of objects is laying on the 3D visibility graph of the scene.

As we mentioned in section 4.5, it might be possible to improve the running time to be $O(n^2 \log n + k)$, where k is the number of visibility edges of pseudo-graph, by using pseudo-triangulation based on the work of [10]. We also need to find some application for 3D visibility graph and our new visibility relations.

References

[1] B. Ben-Moshe, O. Hall-Holt, M. J. Katz, J. S. B. Mitchell. Computing the visibility graph of points within a polygon

[2] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. Computational Geometry: Algorithms and Applications, *Second, Revised Edition, 1998*, pp. 307-316.

[3] E. W. Dijkstra, A note on two problems in connection with graphs, *Numerische Mathematik, 1 (1959)*, pp. 269-271

[4] F. Durand. 3D Visibility: Analytical Study and Applications. *PhD thesis, University of Joseph Fourier*, 1997.

[5] T. Lozano-Perez, M. A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, *Communications of the ACM, 22 (1979)*, pp. 560-570.

[6] J. Canny and J. H. Reif. New lower bound techniques for robot motion planning problems. *In Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci.*, 1987.

[7] J. S. B. Mitchell and M. Sharir. New Results on Shortest Paths in Three Dimensions. *SCG04, June 911, 2004*, Brooklyn, New York, USA.

[8] M. Pocchiola and G. Vegter. The visibility complex. *Internat. J. Computational Geometry Appl.*, 1996. special issue devoted to ACM-SoSG'93.

[9] R. Orti, F. Durand, S. Riviere and C. Peuch, Using the visibility complex for radiosity computation, *in 1st ACM Workshop on Applied Computational Geometry, WAGC96, Philadelphia, PA, Springer-Verlag Lecture Notes in Computer Science, vol. 1148, (1996)*, pp. 177-190.

[10] M. Pocchiola, G. Vegter, Computing the visibility graph via pseudo-triangulations. *In Proceedings of the 11th Annual ACM Symposium on Computation Geometry, Vancouver, B.C., (1995)*, pp. 248-257.

[11] Lee, D. T., Proximity and reachability in the plane, *Ph. D. thesis and Tech. Report ACT-12, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana,* (1978).

[12] E. Welzl. Constructing the visibility graph of n lines segments in the plane. *Information Processing Letters, 20:167171*, 1985.

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

[14] Edelsbrunner, H. and L. Guibas, Topologically sweeping in an arrangement. *In Proceedings of the 18th Annual Symposium on Theory of Computing* (1986), pp. 389-403.

[15] Asano, T., T. Asano, L. Guibas, J. Hershberger, H. Imai, Visibility of disjoint polygons, *Algorithmic, 1* (1986), pp. 49-63.

[16] Kapoor, S. and S. N. Maheshwari, Efficiently constructing the visibility graph of a simple polygon with obstacles. *In SIAM Journal on Computing, Vol. 30, No. 3* (2000), pp. 847-871.

[17] Overmars, M. H., and E. Welzl, New methods for constructing visibility graphs. *In Proceedings of the 4th Annual ACM Symposium on Computational Geometry, Urbana, IL* (1988), pp. 164-171.