A visibility graph based method for path planning in dynamic environments

Hrvoje Kaluđer*, Mišel Brezak ** and Ivan Petrović**
* TEB Elektronika d.o.o,
Vončinina 2, 10000 Zagreb, Croatia
hrvoje.kaluder@teb-elektronika.hr
** Faculty of Electrical Engineering and Computing,
University of Zagreb,
Unska 3, 10000 Zagreb, Croatia

Abstract - Computational geometry is very important for solving motion planning problems. Visibility graphs are very useful in determining the shortest path. In this work, a modified Asano's algorithm is implemented for determining the visibility polygons and visibility graphs. Implementation is done using the CGAL library. Although the principle for determining visibility graphs is rather simple, the procedure is very time and space consuming and the goal is to achieve lower algorithm complexity. The algorithm consists of two steps: first, angular sorting of points is done using the dual transformation, and second, visibility between the points is determined. Testing of the algorithm is done on two polygonal test sets. The first is made of squares, uniformly and densely distributed. The second is made of triangles, randomly and sparsely distributed. Results show a cubical complexity of the algorithm, depending on the number of reflex points. The main advantage of this method is that it can be applied in dynamical environments (environments that change in time). It is not required to perform the calculation for all points on the map. Instead, the graph can be refreshed locally so it is very practical for online use.

I. INTRODUCTION

Path planning is one of the main problems in robotics. Goal is to find a collision-free path amidst obstacles for a robot from its starting position to its destination.

Using visibility graphs for determining the shortest path is very practical and intuitive. The visibility graph of a set of nonintersecting polygonal obstacles in the plane is an undirected graph whose vertices are the vertices of the obstacles and whose edges are pairs of vertices such that the open line segment between each two vertices does not intersect any of the obstacles.

Main problem in designing the visibility graph is determining the visible portions of map. This operation is time and space consuming, even for modern computers. To determine the visibility of polygons many algorithms have been developed. They all have something in common - great computational complexity.

Let Q be a polygon with n vertices. Let $P = \{s_1,..,s_m\}$ be a set of m points in Q; the points in P may lie both on the boundary of Q and in the interior of Q. The visibility graph of P in Q, denoted $VG_Q(P)$, is the graph whose nodes are the points P and whose edges connect pairs of nodes that see one another within Q (i.e., the segment joining the points lies within Q). See Figure 1 for an example.

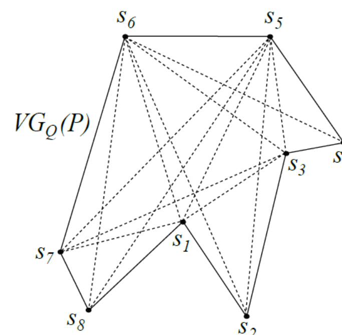

Figure 1. Visibility graph $VG_Q(P)$ of polygon Q with vertices s_i

In this paper, the algorithm for designing visibility graph in two-dimensional configuration space is implemented. It is based on modified Asano's algorithm [1] and implemented using the CGAL Library [4].

Related work. Basic and most intuitive (naive) algorithm has $O(n^3)$ time.. For n polygonal reflex vertices there are n^2 possible visibility connections between them ($O(n^2)$ time). It takes $O(n)$ time for detecting if two points are *intervisible* (they see each other).

$O(n^3)$ time means that algorithm becomes significantly time consuming for larger number of vertices. So more advanced and faster algorithms were developed.

Lee [8] managed to construct the visibility graph $VG_Q(V)$ in $O(n^2 \log n)$ time, using a radial sweep about each vertex Q. Welzl [5] and Asano et al. [1] improved the time bound to $O(n^2)$, which is worst-case optimal but not output sensitive. For general polygons (with holes), Overmars and Welzl [9] obtained a relatively simple algorithm running in $O(k \log n)$ time and $O(n)$ space, where k is the number of edges in the visibility graph. Then Gosh and Mount [2] developed optimal technique of

planar scanning using triangulation and funnel splits running in $O(k + n \log n)$ time and $O(k)$ space.

Dual transformation and algorithms for sorting points and determining the visibility polygon are presented in section II. In section III algorithm for determining the visibility graph with pseudo code is given. In section IV results of simulation on test polygonal sets and real map are shown. The paper ends with a conclusion.

II. VISIBILITY POLYGON

A. Dual transformation

In the Cartesian plane, a point has two parameters (x- and y-coordinates) and a (non-vertical) line also has two parameters (slope and y-intercept). We can thus map a set of points to a set of lines, and vice versa, in an one-to-one manner. This mapping is called dual transformation.

Let $p=(p_x, p_y)$ be a point in original plane (x-y plane). We say that line p^* is dual transformation of point p and described as:

$$p^* := (y = p_x x - p_y).$$

Let $l : y = m x + b$ be a line in original plane. We say that point l^* is dual transformation of line l as given:

$$l^* := (m, -b).$$

See Figure 2 for example.

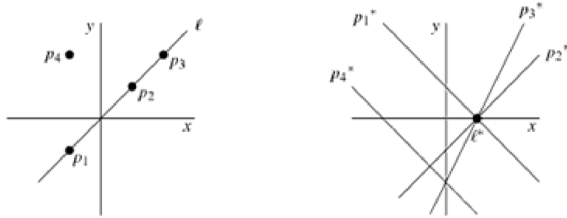

Figure 2 Original and dual plane - dual transformation principle

B. Sorting points using dual transformation

Let q be a point of view and p_i (i=1,…,N) points we want to sort by slope regarding to q. In first step we map the points p_i in original plane to lines in dual plane. These lines form an arrangement in dual plane. This preprocessing step takes $O(n^2)$ time and space. Insertion of each new line in dual arrangement takes $O(n)$ time. Important is to notice that line connecting points p and q in original plane corresponds to the point of intersection of two lines, q^* and p^*, in mapped plane.

Hence, the ordering by slope of the lines connecting q and p_i (i = 1, …, N) corresponds to the ordering by x-coordinate of the points of intersection of the line q^* with lines p_i^* (i = 1, …, N) in the mapped plane. Since we have the arrangement at hand, this ordering can be obtained in $O(n)$ time.

Procedure is shown in Fig.3.

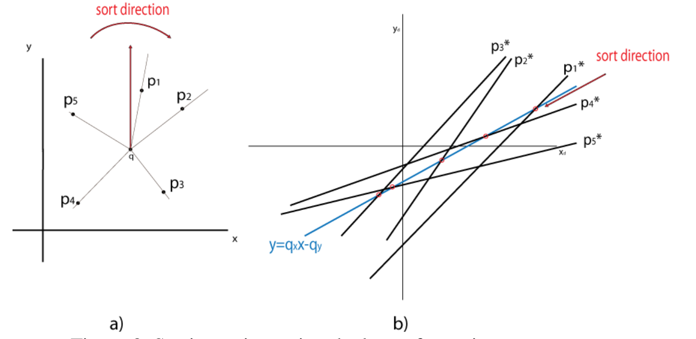

Figure 3. Sorting points using dual transformation
a) original plane b) dual plane.

C. Determining the visibility polygon

Determining the visibility polygon is done via scan lines and segment splitting. This method uses polar scan line to sweep through the sorted list of segments to find the visible sub-segments.

In first step, we define starting scan line (extending vertically up from view point q). Then, we place a ray emerging from q to each of N previously sorted points p_i. We start from point q and move along the ray. If we encounter a *right side* segment (segment extending right of the ray) we store it and move to next ray. Otherwise, we continue moving along the ray, until there are no more intersections or we encounter a *right side* segment. We repeat this procedure for each of N rays, detecting the first segment we encounter while moving along the ray (starting from q). See Figure 4 for example.

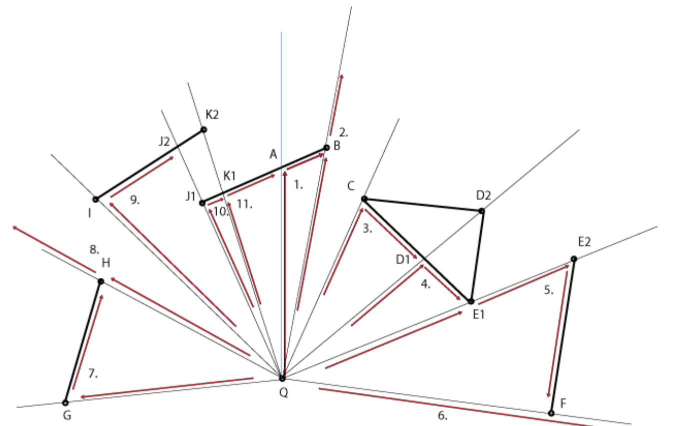

Figure 4 Determinig the visibility polygon - procedure in steps

III. VISIBILITY GRAPH

In case of determining the visibility graph algorithm for determining visibility polygon is slightly modified because it is not necessary to know which segment or sub-segment each view point “sees”. Only visibility of edge points of segments is important.

So upper algorithm is modified in way it doesn’t detect segment ray encounters; only ray directly encountering one of p_i points matters. For simplification, it is not necessary to process all points p_i, only *reflex* ones (vertices at which the internal angle of polygon is grater then 180 degrees).

The information whether a reflex point is visible or not is stored in binary visibility matrix (1 - visible, 0 - not visible) whose dimension is N x N (for n reflex points). Each row of matrix presents one visibility vector (n^{th} row presents visibility vector of points viewed from n^{th} point).

Problem is how to store information in proper field of matrix, because for each query point q points p_i are differently sorted. It is solved using the additional information added to vertex object in CGAL – pointer to the exact element in visibility matrix.

Also, the visibility is bidirectional phenomenon which means it is required only to check it once per pair of points (Ex. If point A "sees" B then B "sees" A also). This means it is not necessary to fill all fields in visibility matrix (n x n) – only above or below the diagonal (values on diagonal are equal to 1).

Pseudo code is given below. Input is array of edge points P (vertices of all polygons on map). Output is visibility matrix (VG).

```
0   BEGIN
    1   RP = DetermineReflexPoints(P);
    2   RPdual=DualTrans(RP);
    3   S = DetermineSegments(P);
    4   ArrD = CreateArrangement(RPdual);
    5   Arr = CreateArrangement(S);
    6   InitVG(); EnumarateVertices(Arr,RP);
    7   For i = 1 to length(RP)
            a.  Qdual = RPdual(i); Q= RP(i);
            b.  He=*Locate(ArrD,Qdual);
            c.  [P1,P2] =FindNeighbours(Arr,Q);
            d.  SortedPoints= Sort(ArrD,he);
            e.  IsVisible(Arr,&VG,Q,SortedPoints);
    8   END
```

Steps 1- 6 are preprocessing steps executed only once for given input data set. Step 7 is repeated for each reflex point on map. Functions used above are implemented using CGAL Library and defined as follows:

DetermineReflexPoints(p): Input parameter is array of all points on map p and output is array of reflex ones.

DualTrans(p): Input parameter is array of points p and output is array of lines, which represents dual mapping of those points.

DetermineSegments(p): Input is array of points p ordered as follows: counter clockwise for outer polygon and clockwise for inner polygons. From array of points an array of segments is formed (segments representing the polygon edges).

CreateArrangement(s): Input is array of segments or lines s and output is an arrangement of those. Insertion is done using the CGAL *insert()* function.

EnumarateVertices(arr,p): Input is arrangement *arr* (type: *Arrangement_2*) and array of points p. Output is arrangement with vertices enumerated in order they appear in array p.

***Locate(arr,S):** Input is arrangement *arr* and line S. Line is inserted in arrangement and pointer to that line is returned as output (type: *Halfedge_handle*).

FindNeighbours(arr,P): Function takes point P and arrangement *arr* (map) as input and returns predecessor and successor of vertex P in arrangement *arr.*

Sort(arr,he): Function takes pointer on line *he* and arrangement *arr* as input and returns array of sorted points.

IsVisible(arr,&VG,P,p): Function takes arrangement *arr* (map), pointer to visibility matrix *VG*, view point Q and array of points sorted about Q as input. Row with index corresponding to index of Q (in array of reflex points) will be modified.

InitVG(): Initialization of visibility matrix (to 0).

Theoretically, shown algorithm runs in $O(e\, n^2)$ time where e is average number of sorted points (that can be visible).

Even though sorting is done in O(n) time, the complexity is increased due to way functions and objects are implemented in CGAL. They are not optimized for tasks such as locating and directly accessing vertices and lines/segments/rays in arrangement, thus one has to locate them iteratively, by searching through all elements (points, segments etc.).

IV. RESULTS

Testing of the algorithm is done on two polygonal test sets. The first is made of squares, uniformly and densely distributed. The second is made of triangles, randomly and sparsely distributed. Results are given below.

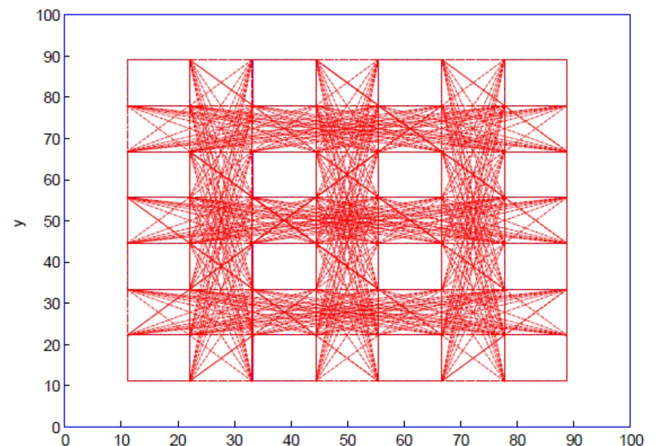

Figure 5 Visibility graph for polygon made of 16 uniformly distributed squares

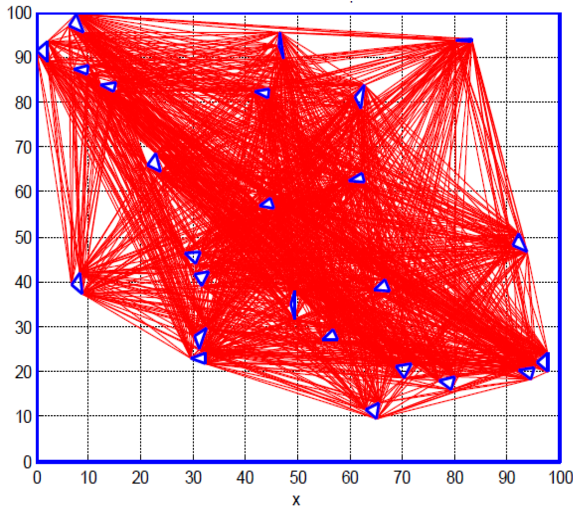

Figure 6 Visibility graph for polygon made of 75 randomly distributed squares

Execution time has been measured. Data is given in table below:

TABLE 1 DURATION OF ALGORITHM EXECUTION (SQUARES)

Number of reflex vertices N_{rp}	Preprocessing time T_{pred} [s]	Main loop time of execution (sorting and determining VG) - T_{alg} [s]	Total time of execution T_{uk} [s]
64	0.294	3.198	3.492
144	1.498	28.924	30.422
256	4.755	160.264	165.019
400	12.524	653.514	666.038 ≈11min

TABLE 2 DURATION OF ALGORITHM EXECUTION (TRIANGLES)

Number of reflex vertices N_{rp}	Preprocessing time T_{pred} [s]	Main loop time of execution (sorting and determining VG) - T_{alg} [s]	Total time of execution T_{uk} [s]
30	0.0793	0.5549	0.6342
75	0.4407	5.2668	5.7075
150	1.7679	29.5604	31.3283
225	3.8039	90.7532	94.6671
300	6.6481	192.98	199.6281
399	11.8844	466.957	478.8414

Results are shown graphically:

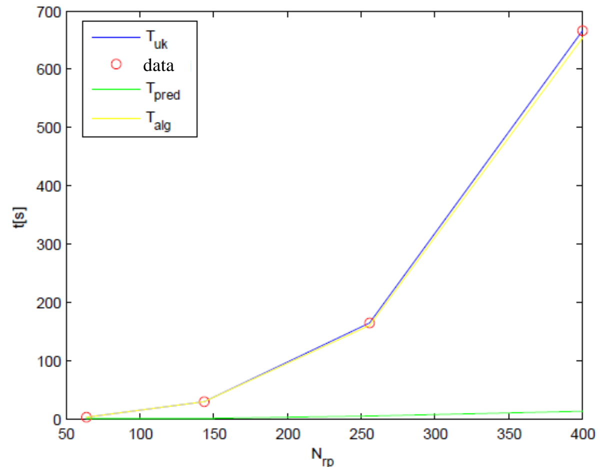

Figure 7 Duration of execution of algorithm (square polygons)

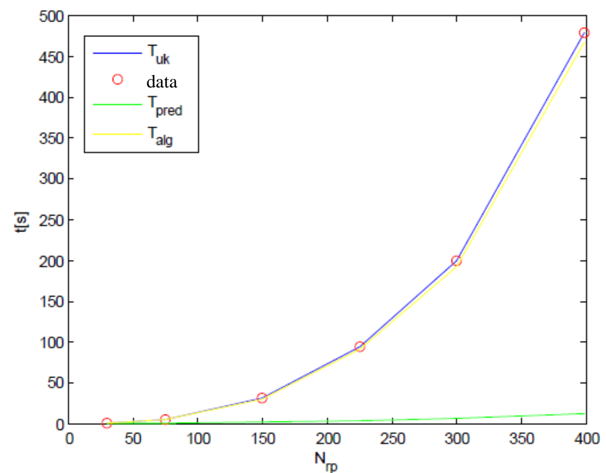

Figure 8 Duration of execution of algorithm (triangle polygons)

Results show cubical dependency of time of execution on number of reflex points ($k\ n^3$). Even though execution gets rapidly slower as number of points increase, it is noticeably faster on sparse maps (for sparsely distributed triangles).

The algorithm was also tested on real map (N_{points} = 740):

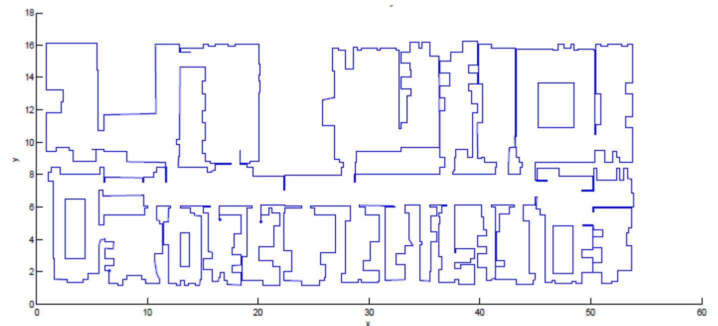

Figure 9 Test map

Results are shown in Figure 10.

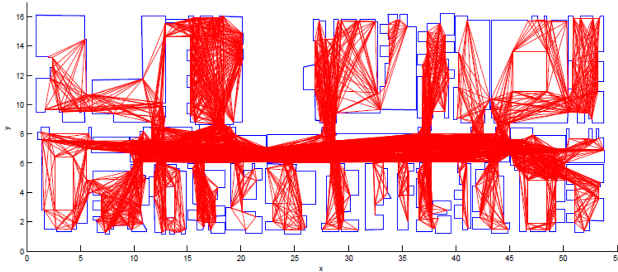

Figure 10 Visibility graph of test map

Preprocessing time: 11.1 s.
Total execution time: 630.15 s.

V. Conclusion

Results show a cubical complexity of the algorithm, depending on the number of reflex points. It can be rather time consuming for larger number of points. The reason is in using the CGAL Library, which is not optimized for visibility graphs (problem is in direct accessing objects in arrangements and moving along the arrangement).

Algorithm is noticeably faster on sparse maps which can be very useful in using the visibility graph for path planning on open spaces with sparse objects.

The main advantage of this method is that it can be applied in dynamical environments (environments that change in time). It is not required to perform the calculation for all points on the map. Instead, the graph can be refreshed locally so it is very practical for online use.

To reduce execution time it is necessary to write a library optimized for computing visibility graphs. It should be designed for direct accessing objects in arrangements and for standard operations with arrangements (vertices, segments, rays etc.).

References

[1] T. Asano, L. J. Guibas, J. Hershberger, and H. Imai. “Visibility of disjoint polygons”, Algorithmica, 1:49-63, 1986.

[2] S.K Ghosh and D.M.Mount, “An output sensitive algorithm for computing visibility graphs”, SIAM Journal on Computing, Vol.20. No.5, pp. 888-910, 1991.

[3] M. Brezak, “Localization, motion planning and control of mobile robots in intelligent spaces”, PhD Thesis, University of Zagreb, Faculty of Electrical Engineering and Computing, 2010.

[4] CGAL Manual, CGAL Open Source Project, release 3.6, http://www.cgal.org/Manual/3.4/doc_html/cgal_manual/contents.html, 20th March 2010.

[5] E. Welzl, “Constructing the visibility graph for n-line segments in $O(n^2)$ time, in Information Processing Letters”, vol. 20, pp. 167-171 , 1985.

[6] J. Kitzinger, “The Visibility Graph Among Polygonal Obstacles : a Comparision of Algorithms”, University of New Mexico, 1993.

[7] E.W Dijkstra, “A note on two problems in connection with graphs”, Numerische Mathematik, vol. 1 , pp. 269-271, 1959.

[8] D.T. Lee, “Proximity and reachability in the plane”, Technical port, Dept. Elect. Engineering, Univ. Illinois, Urbana, IL, 1978.

[9] M. H. Overmars and E. Welzl., “New methods for computing visibility graphs”, in Proc. 4th Annu. ACM Sympos. Comput. Geom., pp. 164-171, 1988.