Extending the VGRAPH Algorithm for Robot Path Planning

Alade Tokuta
Department of Mathematics & Computer Science
North Carolina Central University
Durham, NC 27707
tokuta@sci.nccu.edu

ABSTRACT

This paper presents an O(**n**log**m**) method to incorporate start and goal points of a robot into the roadmap of a two-dimensional workspace to form a VGRAPH. A VGRAPH Point Incorporation Algorithm(**VPIA**) incorporates a point in free-space into a roadmap. This VPIA divides the free space around an obstacle vertex into an ordered set of areas. A search is used to determine the containment of the point. Containment implies visibility of the point from the vertex. A point is incorporated by determining its visibility from all obstacle vertices. The VPIA is inherently parallel and the implementation can reduce this complexity from O(**n**log**m**) to O(log**m**). Most existing techniques for incorporating points into a roadmap perform in $O(n^2)$. The roadmap's data structure is modified to support the VPIA. The VPIA, employing the VGRAPH approach should enhance the worst case runtime of find-path algorithms closer to real-time for 2-D workspaces cluttered with obstacles. Sample VGRAPH diagram generated by an implementation of the VPIA is shown in Appendix A.

Keywords: visibility graph, path planning, VGRAPH

1.0 INTRODUCTION AND BACKGROUND

Providing a robot with the ability to move safely from one point to another in a workspace with obstacles is called ***the find-path problem***. The requirement of the find-path problem is defined as that of finding the shortest path for a robot, in moving from a given start point to a given goal point, while avoiding collisions with obstacles.

1.1 The VGRAPH Approach

Various techniques are used in building a search space graph for a robot's workspace. This paper focuses on the vertex-graph(VGRAPH) technique [Lozan79, Morav81, Thomp77]. In the VGRAPH method the obstacles are expanded by the size of the robot and the robot is reduced to a point[Morav81]. Lozano-Perez [Lozan83] presented a technique for Cspace obstacles which is used to form the VGRAPH on which the search for an optimal path is performed. This paper assumes a workspace with polygonal Cspace obstacles and a point robot.

The VGRAPH, **G(N,E)**, is a graph with a set of nodes N, and a set of edges E. Each obstacle vertex v in the workspace is represented by a node, N_v, in N. The start and goal points p_s and p_g are also represented by the nodes, N_s and N_g. Each edge in E links a pair of nodes in N whenever the vertices or points represented by the node pair are visible to each other or whenever they are the end vertices of an obstacle edge. The VGRAPH represents a discrete subset of collision-free paths from p_s to p_g. Each path is made up of a sequence of line segments. The VGRAPH contains the shortest path from p_s to p_g in its discrete subset [KantZ86,Wangd74]. To find an optimal path from p_s to p_g (assumed one not visible to the other), the VGRAPH is searched using a graph-search algorithm such as the A*[HartN68].

1.2 Building the VGRAPH Using a Roadmap

We are concerned with navigating robots between varying start and goal points among static obstacles. In this environment, a VGRAPH may be regarded as having two parts: a permanent part formed from the obstacle vertices, and a dynamic part formed from the start and goal points. The permanent part consists only of the set of obstacle vertex nodes $\{N_v\}$ and the edges $\{E'\}$ on which they are incident. It is fixed

for a static workspace and is referred to as the roadmap for the workspace. We assume the roadmap contains space for both N_s and N_g in its data structure, but the edges at N_s and N_g are omitted. The dynamic part of the VGRAPH changes whenever p_s or p_g changes and a new VGRAPH of the environment must be constructed when p_s or p_g is altered.

Because the roadmap is fixed, the VGRAPH is built in *two phases*. The *first phase* is a pre-processing phase which builds the roadmap. This phase is executed once and is not repeated in subsequent solutions of the find-path problem when start and goal positions vary. This first phase has been shown to be executable in $O(n^2)$ time [Guiba85, Welzl85], where n is the total number of obstacle vertices. The *second phase* of the VGRAPH building process integrates N_s and N_g into the roadmap. This is the phase that must be repeated, and is a runtime operation which should be ideally achieved in real-time. When all links at N_s and N_g are specified, the roadmap is called the VGRAPH. The second phase of the roadmap building process plays a key role in the real-time performance of the navigating robot. It contributes part of the robot's reaction time.

1.3 Runtime of the Find-path Algorithm

Some methods for integrating N_s and N_g into the roadmap run in $O(n^2)$ time complexity. One disadvantage of the VGRAPH method, noted [Singh87], is that every time source and destination points are changed, the visibility graph has to be recomputed. This requires $O(n^2)$. However, the plane-sweep algorithm [Fujim91, Prepa85] of $O(\mathbf{n}\log\mathbf{n})$ can be used to incorporate N_s and N_g into the roadmap. However, its order of complexity cannot be easily enhanced, while the time complexity of the VPIA can be enhanced from $O(\mathbf{n}\log\mathbf{m})$ to $O(\log\mathbf{m})$ by parallel processing.

The incorporation of start and goal point is not the only runtime operation in solving the find-path problem. The graph-search operation is equally critical in impacting real-time performance. The A* algorithm is optimal and performs in $O(n^2)$. Thus, the overall runtime complexity of the find-path solution remains in $O(n^2)$, although use of the $O(\mathbf{n}\log\mathbf{m})$ VGRAPH Point Incorporation Algorithm, described, enhances the worst case runtime of find-path algorithms, that employ the VGRAPH approach, for 2-D workspaces cluttered with obstacles. The worst case run-time complexity of the find-path algorithm will remain unchanged, but its worst case run-time is reduced.

2.0 VISIBILITY TO AN OBSTACLE VERTEX

The robot's workspace assumed to be a two-dimensional region with non-intersecting stationary convex polygonal obstacles. The ***Workspace Vertex Set*** $\{V_w\}$ consist of the union of the ***Obstacle Vertex Set*** $\{V_o\}$ and includes the vertices of the workspace boundary. The size of V_o ($|V_o|$) is n. Similarly, the ***Obstacle Edge Set*** $\{E_o\}$ is the set of obstacle edges, and the ***Workspace Edge Set*** $\{E_w\}$ is the union of E_o and the boundary edges of the workspace. We denote an obstacle as P_i if the point i lies on its boundary. We asuume the workspace contains m configuration space obstacles and the robot, is a point robot [Lozan83]. The free-space visible to each vertex in V_o can be specified as follows: we denote by $V_{wv} \subset V_w$ those vertices visible to v. For each vertex z ($z \neq v$) in V_{wv}, a directed line segment vz is drawn from v to z. Figure 1 is a set of line segments emanating from v and satisfying the following three conditions:

1) Each line segment is directed towards a vertex in V_{wv}.
2) None of the line segments divides any of the obstacles.
3) Each line segment is the longest possible.

Any coincident line segments must be replaced by a single equivalent line segment from v. Each line segment radiating from v is termed a ***visibility line*** of v. Each pair of adjacent visibility lines is associated with a triangular area with

apex at v, between them. The base of the triangle is the first edge segment visible when looking outward from v between the pair of adjacent visibility lines of the triangle. Each unoccupied triangular area is called a ***visibility triangle*** (v_Δ_v) of v.

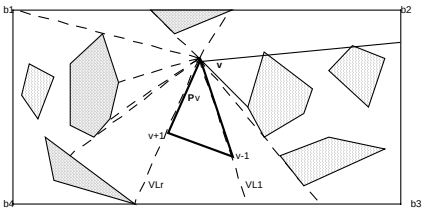


Figure 1: Example of a Robot's workplace. Visibility lines for an obstacle vertex v.

The union of all these visibility triangles represents the total free-space area within which any point is visible to v. By definition, a visibility triangle cannot enclose part of an obstacle. v_Δ_v may degenerate to a point or a line segment. This visibility triangle is called a ***Degenerate Visibility Triangle*** (d_Δ_v) and occurs when obstacles touch or when a workspace boundary vertex is in contact with a vertex v of obstacle P_v It also occurs as in figure 2 when the vertex of one obstacle is in contact with another obstacle or workspace boundary. That is, for some **v** ε $\mathbf{P_v}$ the visibility lines defined by VL_k and VL_{k+1} are collinear with some **e** ε **B** and it constitutes the base of the corresponding visibility triangle of **v** which has all its edges collinear.

3.0 A DATA STRUCTURE FOR THE VPIA

No visibility line should leave v at a blind angle, and all the visibility lines are ordered. If a ine segment leaves v at an angle such that it cuts P_v, then that angle is called a blind angle of v. All vertices in the blind angle direction are obstructed from the view of v by P_v. We denote the first visibility line of a vertex v which runs along the edge of the obstacle P_v in the direction v->v-1, as VL_1. This corresponds to the visibility line in the counterclockwise direction (negative half edge) when looking out across P_v, from v at a blind angle. The second, along an edge of P_v, in the direction v ->v+1 (figure 1), is denoted VL_r (r is the total number of visibility lines of v) and is the visibility line to the clockwise side looking out from v at a blind angle. The visibility lines are numbered from VL_1 to VL_r. Each subsequent visibility line VL_i is measured as a displacement from VL_{i-1} .

3.1 The Visibility Triangle Array (VARRAY)

A ***Visibility Triangle Array*** (**VARRAY**) of r cells stores the angular metric of the visibility lines in their order of non-decreasing values. Each cell becomes a structure which also consists of additional data fields described next.

We denote as VT_k some visibility triangle of v between VL_k and VL_{k+1}. Every VT_k has a visibility line VL_k on its clockwise side. We associate each cell VARRAY[k] to its corresponding visibility triangle VT_k. Note that VL_r is followed by the blind angles of v.

To complete definition of VT_k, and to facilitate rapid use of VARRAY in its determination of the visibility of points in free space from v (figure 3), the final element of a record becomes the problem of determining whether a given point, p, is within VT_k. Let U_k be the inward pointing unit normal to the base of VT_k, then p visible to v iff:

$$U_k \cdot ap \geq 0$$

When VT_k is degenerate, 'a' is coincident with v and U_k is set to point in a direction perpendicular to VL_k (Figure 2) for consistency. Once θ(vp) (the angle at which v_p leaves v, has been determined to be between $\theta(VL_k)$ and $\theta(VL_{k+1})$, only U_k and 'a' are needed to determine the visibility of p from v. Each structure VARRAY[k] comprise of three fields: {$\theta(VL_k)$, value of 'a', and value of U_k}. The data structure of VARRAY described determines the visibility of a point in free space from one obstacle vertex v.

Similar VARRAY structure for every v in V_o, determines the visibility of a point p from each vertex in V_o. Because $|V_o| = n$, n VARRAYs are needed to store all the v_Δ_V's. The final data structure is of the form: DSTR: Array[1..n] of VARRAY.

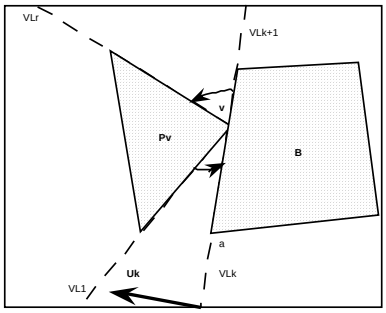


Figure 2: A degenerate visibility triangle VTk showing the BaseIntersect 'a' and base of degenerate visibility triangle

The variable, DSTR, thus contains a VARRAY for each obstacles vertex and defines all the visibility triangles for all $v \in V_o$.

4.0 SPACE COMPLEXITY OF THE VPIA DATA STRUCTURE

A Visibility Triangle record in VARRAY has O(1) space complexity. VARRAY has space complexity of O(n). Since the DSTR must store O(n) VARRAYs, DSTR has space complexity of $O(n^2)$. This order of space complexity is the same as that of the roadmap. We show later that although the VGRAPH Point Incorporation Algorithm adds $O(n^2)$ space complexity to the find-path problem, its use of the already existing $O(n^2)$ roadmap data structure proves to be very convenient.

5.0 THE VGRAPH POINT INCORPORATION ALGORITHM

Using DSTR, an algorithm is specified to determine the visibility of any point p in free-space from the vertices in V_o. The VPIA comprise of three steps which are applied to each vertex v in V_o.

1) Determine which of the visibility triangles of v are candidates for containing the point p.
2) Determine if p is contained in any of the candidate triangles found in step 1.
3) If the point p is contained in a visibility triangle, add the edge N_vN_p to the roadmap.

These steps executed for all obstacle vertices, incorporates the point p into the roadmap by *linking* a node N_p of p to *those* nodes representing an obstacle vertex from which p is *visible*. We now outline these three steps in more detail:

STEP 1: ***Finding Candidate Visibility Triangles***

In figure 3 are shown visibility triangles VT_k of an obstacle vertex v. For a point p, a line segment is drawn from v to p. It emanates from v at an angle θ(vp) such that:

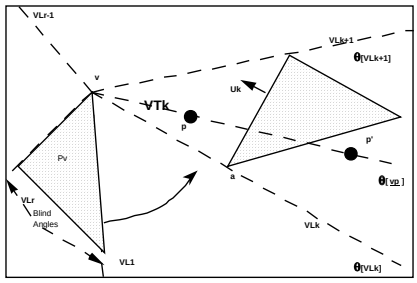


Figure 3: Point containment by visibility triangle. Typical visibility triangle VTk bounded by its two visibility lines VLk and VLk+1. A point **p** in the θ[vp] direction can be in VTk or outside VTk (**p'**). **a** is the intersection of VLk and the base of VTk.

a) $\theta(VL_k) \le \theta(vp) \le \theta(VL_{k+1})$ $(1 \le k \le r-1)$

b) $\theta(vp) > \theta(VL_r)$

With (a), VT_k becomes a candidate visibility triangle; (b) implies θ(vp) is a blind angle. θ(vp)is an adjusted angle (adjusted whenever $|\theta(vp)| < |\theta(VL_1)|$ to keep θ(vp) consistent with the angles assigned to the visibility lines of v. VT_k becomes a candidate for $\theta(VL_k) \le \theta(vp) \le \theta(VL_{k+1})$; a binary search of VARRAY for the first k such that $\theta(vp) \le$ VARRAY[k].VL_kAngle locates candidate visibility triangles in VARRAY. If no such k exists, then θ(vp) is a blind angle and thus no candidate visibility triangle exists, else VT_{k-1}

is the candidate. Furthermore, if θ(vp) = VARRAY[k].VL_kAngle and k < r, then VT_k serves as a second candidate in case when p proves to be outside VT_{k-1}. Finally, if θ(vp) = VARRAY[1].VL_kAngle, then only VT_1 will serve as a candidate, else if θ(vp) = VARRAY[r].VL_kAngle, in which case only VT_{r-1} serves as a candidate. In the worst case, a search takes O(log**n**) time.

STEP 2: ***Containment of p within Visibility Triangle***
Step 1 gives a maximum of two visibility triangles of v as candidates to be checked for containment of p. If the first contains p, then the second may be ignored. If neither contains p, then p is not visible to v.

VT_k will contain p iff p lies between or on the base of VT_k and v. (Figure 6). It is assumed that p is within VT_k if p lies on the base of VT_k. The following scalar product expression summarizes the condition for containment of p by VT_k:

VARRAY[k].NormalVector . (p - VARRAY[k].BaseIntersect) ≥ 0

STEP 3: ***Linking Node Representing p to the Roadmap***
This last step is trivial, and is not done if p was determined to be invisible to v.

With the above steps executed for all v in V_o, the point p is *incorporated* into the roadmap. This is the VGRAPH Point Incorporation Algorithm. Now, given that p_s and p_g are obstructed from each other's view, then the VPIA is applied twice; once with p_s, then again with p_g. This yields the VGRAPH.

6.0 COMPLEXITY OF THE VPIA

Step 1 is O(log**n**) time. Steps 2 and 3 are both O(1). Over n, yields the total time complexity of the VPIA as O(**n**log**n**). Thus, the order of time complexity involved with the VPIA in the solution of the find-path problem is O(**n**log**n**).

The computation of the visibility of the point p from a given vertex does not use or depend upon data generated in the computation of the visibility of p from any other vertex. The computation of the visibility of p from each of the n vertices in V_o can be done simultaneously. Thus, unlike the plane-sweep algorithm, the VPIA can be executed over n processors in O(log**n**) worst case time complexity.

7.0 USING THE ROADMAP DATA STRUCTURE FOR THE VPIA DATA

7.1 Basic Roadmap and VGRAPH Data Structure
We refer to a VGRAPH or roadmap node as a vnode. The basic roadmap has n nodes, the VGRAPH n+2 nodes. Each vnode potentially has one link to each other vnode in the VGRAPH. We therefore regard a vnode as an array of n+1 pointers with up to n-1 pointers used in the roadmap and the other two pointers at each vnode are for possible links to N_s and N_g.

There are two other pieces of data usually stored at the vnode: the first is its obstacle vertex, and the second is an estimate on the lower bounds of the cost of going from that vnode to the goal vnode N_g, i.e. h(n) in the expression:

$$f(n) = g(n) + h(n)$$

of the A* algorithm [HartN68]. The third item is an array of n+1 pointers to vnodes. These pointers specify the edges of the VGRAPH at the vnode.

The roadmap is constructed for each vnode N_v such that pointers are set to point to all other roadmap nodes whose workspace vertices are visible from v. At each roadmap node, all pointers pointing to other vnodes occupy the lower set of positions in the array of vnode_pointers. Other pointers in array cells that follow are set to nil.

7.2 Modifying the Roadmap Data Structure
There is a one-to-one mapping of the vnode_pointers at N_v, to the visibility lines at the vertex v. In all, the array of vnode_pointers at each vnode is n+5 pointers long, taking into

account the visibility lines, if any, to the workspace boundary corners, as well as the source-destination pair.

The resulting complete match between vnode_pointers and VARRAY visibility lines suggests that the data structure of every vnode_pointer can be modified to store the information stored by its corresponding visibility triangle in VARRAY. The following modification of the vnode_pointer data type achieves just that:

```
vnode_pointer = Record
BaseIntersect: point;
VLkAngle: real;
NormalVector: point;
Npointer: Pointer to vnode;
End;
```

We note that vnode_pointer is now a (visibility triangle) structure with a field called Npointer. Npointer points to a vnode. Every visibility line at a vertex v will have a matching Npointer at N_v. These modifications to the roadmap data structure discussed do not affect the $O(n^2)$ time complexity required to build the roadmap. After constructing the roadmap, the following data must be generated for each vnode_pointer record at every vnode N_v of the roadmap:

a) **VL_kAngle**: At most n-1 pointers of vnode-pointers at vnode N_v is used to define edges at vnode N_v in the initial roadmap. In the worst case, generating the VL_kAngle for a vnode_pointer record is max{O(1), O(n)} = O(n) time complexity.

b) **BaseIntersect**: VL_k is at least a segment emanating from v at VL_kAngle from (a), and directed toward the boundary. It is determined in O(1) using vcoord and VL_kAngle. The line segment may then be intersected with E_w in O(|Ew|) = O(n). When VT_k is not degenerate, some $e \in E_w$ may generate multiple intersections with line segment at BaseIntersect (figure 4). For m obstacles, and because a line segment may intersect a maximum of four edges of the obstacle, in the worst case 4m edges must be sorted in O(mlogm) to resolve the intersection. In addition, O(m) scan of intersections is done to determine the intersection in the positive half edge and hence the edge supporting the base of VT_k. The total worst case time complexity in finding the BaseIntersect for a vnode_pointer record is O(n) +O(**m**log**m**).

c) **NormalVector** : Only O(1) time is needed to calculate NormalVector after finding BaseIntersect in (b).

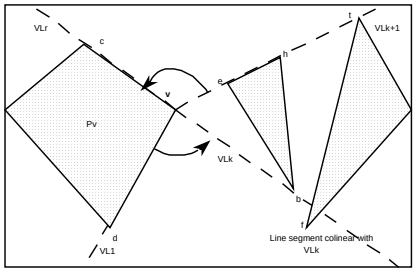


Figure 4: Multiple edges through BaseIntersect. Edges be, bh, ft may intersect the line segment at BaseIntersect 'b'. The correct base-support for VTk is 'be'. It subtends the smallest internal [to VTk] angle to vb.

The VL_kAngles calculated in (a) take O(n) time. Generating (a), (b) and (c) for every vnode_pointer record at all roadmap nodes followed by the adjustment of the VL_kAngles require worst time complexity of $O(n^3 + n^2\mathbf{m}\log\mathbf{m})$.

7.3 Substitution of Roadmap for VPIA Data Structure

The vnode data structure now contains the VARRAY data structure except that the vnode_pointer records at vnode do not have the same order as the visibility lines in VARRAY. This is rectified with use of an O(**n**log**n**) quicksort (of the array of vnode_pointer records) on the VL_kAngles of the vnode_pointer records (that match visibility lines) at each vnode. This is the final *pre-processing* step to convert the roadmap data structure to that needed by the VPIA. All the vnode_pointer arrays in the roadmap are sorted in $O(\mathbf{n}^2\log\mathbf{n})$ time.

Thus, after the basic roadmap has been generated in $O(n^2)$ time, it must be further pre-processed in $O(n^3 + n^2\mathbf{m}\log\mathbf{m})$ time to convert it to the data structure needed by the VPIA. This $O(n^3 + \mathbf{n}^2\mathbf{m}\log\mathbf{m})$ time complexity is

acceptable here, as we are concerned with an efficient *runtime* algorithm. Both processes of building and modifying the basic roadmap to correspond to the required VPIA data structure are pre-processing steps, done only once for a given workspace. Once this roadmap has been so formed and modified, solving the find-path problem for varying start and goal points uses the same roadmap and involves only the VPIA followed by an appropriate graph-search algorithm at runtime. The pre-processing steps do not affect the runtime complexity of the find-path algorithm.

To reuse N_s and N_g every time p_s and/or p_g changes, old VGRAPH links at N_s and N_g are removed to execute new links. This is done in O(n) time.

8.0 FROM O(*n*log*n*) TO O(*n*log*m*)

The VPIA algorithm can be modified to run in O(**n**log**m**). Figure 5 shows visibility line which does not form a tangent to the first obstacle which it touches (after leaving v) is eliminated.

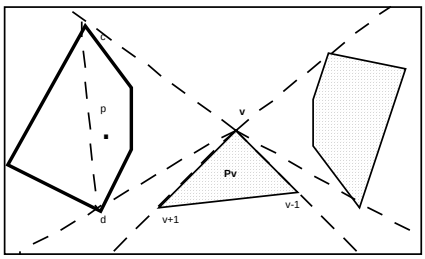


Figure 5: Tangential Visibility Lines at Vertex v

A visibility triangles VT_k is similarly defined as before with the base defined by the two points at which VL_k and VL_{k+1} touch the obstacle (or workspace boundary edge) visible when looking out from v between VL_k and Vl_{k+1}, e.g. triangle *vcd* in Figure 8.

The same VPIA algorithm described previously is used to incorporate a given point in free space into the roadmap. The visibility triangles in this form are fewer and the data structure used is efficient. Each visibility line must be tangent to at least one obstacle, and there can be no more than 2m visibility lines at v. The binary search done by the VPIA on VARRAY of length 2m. The time complexity of that binary search is O(log**m**). Total worst case time complexity of the VPIA is now O(**n**log**m**).

9.0 IMPLEMENTATION RESULTS and CONCLUSIONS

This paper addressed the importance of rapid incorporation of start and goal points into a roadmap, and why this operation is crucial in determining the real-time performance of a navigating robot which uses the VGRAPH approach to solve the find-path problem in a static workspace.

A VGRAPH Point Incorporation Algorithm (VPIA) which performs in O(**n**log**m**) worst case complexity was presented and was applied to integrate start and goal points in a variety of example workspaces. Sample results for an example workspace is shown in the Appendix. This worst case complexity is an improvement over the $O(n^2)$ worst case time complexity of most other algorithms. The VPIA also promises to perform in O(log**m**) time on a parallel machine. This is preferable to the inherently sequential O(**n**log**n**) plane-sweep algorithm which may also be used for determining the visibility of obstacle vertices from a point in the workspace.

The O(**n**log**n**) version of the VPIA was implemented in Turbo Pascal on an IBM PC. The modifying and pre-processing of the roadmap (sections 7.2 and 7.3) were performed simultaneously with the building of the roadmap itself. This resulted in a significant reduction in the pre-processing time.

The lowering in the order of worst case time complexity for n and m large, in find-path algorithms using the VPIA should result in more efficiency and bring closer to real-time performance than existing point incorporation methods that cause the algorithms to run in their worst case times.

We showed how the VPIA can be conveniently supported by modifying the roadmap data structure, so as not to create the need for maintaining two separate data structures when using the VPIA in solving the find-path problem. This is in spite of the fact that the VPIA adds $O(n^2)$ space complexity to the solution of the find-path problem. For 2-D find-path problems the VPIA provides an efficient method for integrating the start and goal points of the robot into the roadmap.

10. LIST OF REFERENCES

[Fujim91] Fujimura,K: *Motion Planning in Dynamic Environments*, Spring-Verlag Tokyo 1991.

[Guiba85] Guibas,L and J. Hershberger, Computing the Visibility Graph of n Line Segments in $O(n^2)$ time, *Th. Comput. Sc.*26, pp 13-20, 1985.

[HartN68] Hart,P.E., N.J.Nilsson, and B. Raphael, A Formal Basis for Heuristic Determination of the Minimum Cost Paths*, IEEE Trans.Sys. Sci. Cyber.,* vol. SSC-4, no.2, pp. 100-107 1968.

[Khati86] Khatib,O,Real-time Obstacle Avoidance for Manipulators and Mobile Robots,*Int'l Journal of Robotic Research,* vol. 5, no.1, pp 90-98, 1986.

[KantZ86] Kant,K and Steven W. Zucker, Towards Efficient Trajectory Planning: The Path-Velocity Decomposition*, Int'l Journal of Robotics Research,* vol.5, no.3, pp 72-89, 1986.

[Lozan79] Lozano-Perez,T and M.A Wesley, An Algorithm for Planning Collision-free Paths Among Polyhedral Obstacles, *Comm. ACM*, vol.22, no.10, pp. 560-570, 1979

[Lozan83] Lozano-Perez,T, Spatial Planning: A Configuration Space Approach, *IEEE Trans. Comput*., vol. C-32, no.2, pp 108-120, 1983.

[Morav81] Moravec,H, Rover Visual Obstacle Avoidance, in Proc.7th *Int'l Joint Conf. Artificial Intelligence*, pp 785-790, 1981.

[Prepa85] Preparata,F.P,and M.Shamos (1985*),Computational Geometry:An Introduction Springer-Verlag, New York,* 1985.

[Singh87] Singh, J. Sanjiv and Meghanad D. Wagh, Robot Path Planning Using Intersecting Convex Shapes: Analysis and Simulation*, IEEE Journal of Robotics and Automation*, vol. RA-3, no.2, 1987.

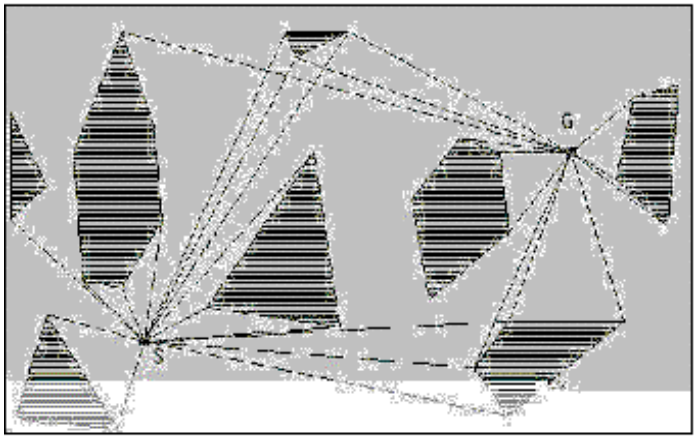
[Thomp77] Thompson,A.M., The Navigation of the JPL Robot,*in Proc. 5th Int'l Joint Conf. Artificial Intelligence*, pp 749-757 1977.

[Vital96] Vital,S, An O(nlogn) VGRAPH Algorithm in Robot Path Planning, MS Thesis, University of South Fl., 19996

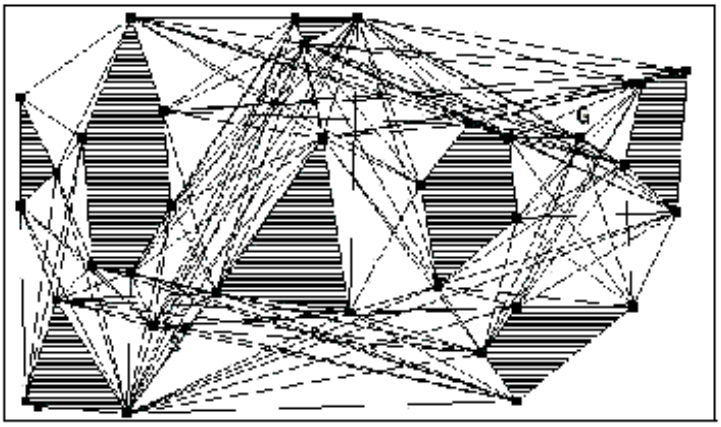
[Wangd74] Wangdahl,G.E., Pollock,S.M., and Woodward, J.B., Mininum Trajectory Pipe Routing, *J. Ship Res*. 18(1),pp. 46-49, 1974).

[Welzl85] Welzl,E, Constructing the Visibility Graph of n Line Segments in $O(n^2)$ Time, *Info. Processing Lett*., vol.20, pp 167

APPENDIX A



A1: Visibility links due to Incorporation of Start-Goal Pair



A2: VGRAPH of Workspace