Visible Decomposition: Real-Time Path Planning in Large Planar Environments

Oded Maron
Artificial Intelligence Lab
NE43-755
Massacusetts Institute of Technology
Cambridge, MA 02139
oded@ai.mit.edu

Tomás Lozano-Pérez
Artificial Intelligence Lab
NE43-836a
Massacusetts Institute of Technology
Cambridge, MA 02139
tlp@ai.mit.edu

Abstract

We describe a method called Visible Decomposition for computing collision-free paths in real time through a planar environment with a large number of obstacles. This method divides space into local visibility graphs, ensuring that all operations are local. The search time is kept low since the number of regions is proved to be small. We analyze the computational demands of the algorithm and the quality of the paths it produces. In addition, we show test results on a large simulation testbed.

1 Introduction

There is rapidly growing interest in the area of large-scale distributed simulations of physical systems. These simulations have many applications in planning, testing and training. In many cases, these simulations involve simulated agents that interact with human users. Producing these simulated agents requires solving many of the classic problems of artificial intelligence and robotics in environments that are more predictable than the "real world" but which are often very complex, since there is little interest in simulating simple environments.

The particular problem we faced was to create a real-time motion planning mechanism for the simulated forces (ModSAF) in the SIMNET military simulation environment [Brock *et al.*, 1992]. Because this is a simulation system, the problems of uncertainty in sensing and control so prominent in robotic motion planning are essentially non-existent. Also, in our case, it is sufficient to consider translational motion in the plane. However, these simulations run on a world in which every tree, boulder, building, river, and swamp are mapped. There are normally thousands of obstacles (each with many vertices) in a typical simulated world. In addition, the agent whose path is being planned is actually a group of vehicles. The various formations of this group have different sizes and result in different radii for the configuration space obstacles. Therefore, the configuration space actually has a third dimension — the radius of the formation. If there are m different formations, there are m times as many vertices in the configuration space. The number of obstacle vertices is now in the millions. Another characteristic of these domains is that some of the obstacles change over time so it must be possible to update the model on a nearly continuous basis. Furthermore, the performance requirements for path planning are very stringent. The planner needs to be able to continuously replan a path for one or more vehicles as the user is dragging his mouse to specify the origin and goal vertices. Therefore, "real-time" in this problem does not mean seconds or minutes but milliseconds.

We chose to focus on motion-planning methods that are deterministic and guaranteed (unlike potential fields), primarily because we felt that users would find it disconcerting to be faced with "erratic" behavior on the part of their simulations. Essentially all the deterministic and guaranteed motion planning methods involve three steps: (1) computing a configuration space (C-space) for the moving body, which in the case of a translating body represented by a circle is simply an expanded version of the obstacles, (2) representing the free-space outside of the C-space obstacles by a graph and (3) finding the shortest path in the graph. Since the first and third steps are well understood [Latombe, 1991], the variability among path-planning methods is limited to the second step. There are three broad classes of approaches that have been taken to represent the free-space for path-planning problems in the plane.

- The earliest and possibly most studied method for path-planning in the plane is the *visibility graph* approach [Nilsson, 1984, Lozano-Pérez and Wesley, 1979, Asano *et al.*, 1986]. This approach has the advantage of producing the shortest collision-free path but has very bad running times for large problems. In particular, the simple versions of the algorithm have poor asymptotic running-time, $O(n^3)$, which in practice means that they perform well for small problems (since there is so little implementation overhead) but poorly for the large problems of practical interest. The optimal algorithms that have better asymptotic running-times, $O(n^2)$, on the other hand are so complex that they perform very poorly for small problems and, at least in our limited experience, require so much space that they are also completely impractical for large problems. Furthermore, the cost of changing the obstacles in the optimal algorithms is empirically very high.

- Another class of methods is based on the "generalized" Voronoi diagram [Ó'Dúnlaing and Yap, 1982, Canny and Donald, 1988]. These methods can support very fast path planning since they produce small graphs and, although more difficult to implement than simple visibility graph methods, are quite practical. Nevertheless, they have one property that is undesirable for our purposes, namely that the paths one gets can be far from optimal. This occurs especially in open areas, since the Voronoi diagram is always maximally distant from obstacles. Another (smaller) disadvantage is that the algorithms for updating the generalized Voronoi diagram when obstacles change are a bit complex and numerically sensitive.

- The other classic approach to motion planning in the plane (and elsewhere) is the *cell-decomposition* approach [Brooks and Lozano-Pérez, 1985] where the space outside of the C-space obstacles is divided into (usually) convex cells that are completely outside the obstacles and which are then connected into a graph. Many versions of this approach exist, some based on hierarchic decompositions and others on line-scanning methods. These methods can be efficient even for large problems. The usual difficulty is, again, how to get "good" paths. If the cells are few and therefore big, one needs a strategy to find paths inside them. If the cells are small enough that one can simply move through their centers, then there are too many cells to be efficient.

The situation seems to boil down to this: methods that can compute the free-space graph efficiently tend to produce paths that can be far from the optimal. The ones that can produce optimal or nearly optimal paths (visibility graphs or fine-scale cell-decomposition) are too slow for the huge problems of interest in simulation. We have no doubt that any of the three approaches described above can be extended into a method capable of dealing with the requirements of our problem. In this paper, we show how to generalize the visibility graph approach.

The method we present, *Visible Decomposition*, divides the configuration space into a grid of regions. These regions include both obstacles and free space in general, so this is not a cell-decomposition method in the usual sense. Each grid region has its own local visibility graph. These graphs, being local, are small enough that the simple methods for constructing visibility graphs are quite adequate. Therefore, building the initial visibility graphs, updating the graphs and planning paths within a cell are all quite fast. We simply make the global visibility graph be the union of the local visibility graphs. One has to add "additional" points along the boundary of the regions to guarantee global connectivity but this is relatively simple. The disadvantage of this method is, predictably, that the paths are no longer optimal. Below, we show that by adding enough extra points along the boundary of the regions one can get good paths, which can then be further smoothed.

In the rest of the paper, we present the Visible Decomposition algorithm and analyze its performance. We then present results of running it on large problems. Finally, we conclude with a discussion of its merits and disadvantages.

2 Visible Decomposition

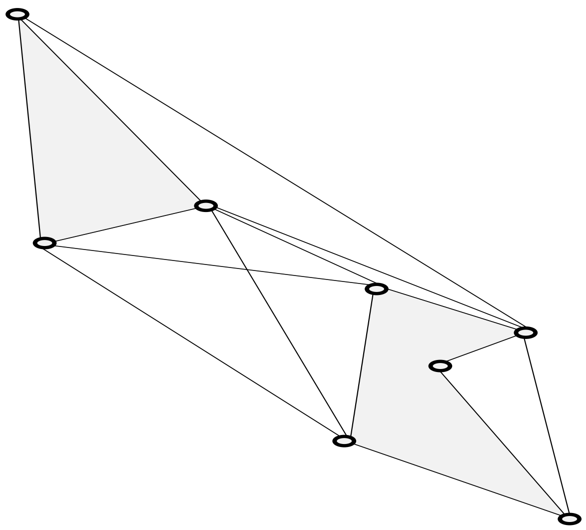
A *visibility graph* is a graph where the nodes are the vertices of the polygonal obstacles, and an edge is drawn from one vertex to another if the edge is "visible". An edge is visible if it does not intersect any obstacles. Once the static graph is constructed, goal and origin vertices are added and their visible edges are computed. This graph can then be searched for a shortest path between the origin and goal. An example of a visibility graph is shown in Figure 1.

In summary, the advantages of using a visibility graph for motion-planning is that it is a simple, well-understood method which returns optimal paths in 2D configuration space. One disadvantage is that the paths it generates hug the obstacles, and in many situations we want the path to stay at a respectable distance from the obstacles. In addition, for a complex world with a large number of obstacles, the visibility check becomes very expensive — we need to check if the candidate edge intersects any of the obstacle edges in the world. This causes the construction of the static visibility graph to be slow, and more importantly dynamic changes such as adding a goal vertex or adding a new obstacle are too slow to be done in real time.

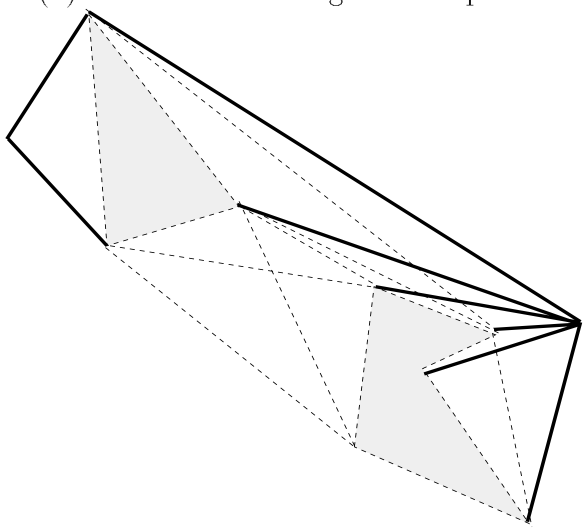
The main purpose of the Visible Decomposition algorithm is to make all operations on the visibility graph (except for search) local operations. This is done by dividing the configuration space into regions, each of which maintains its own collection of vertices. The global graph maintains the edges that connect the vertices. Regions share vertices that lie on the region boundaries so that the global graph will stay connected.

Specifically, the following is an algorithm for computing a visible decomposition, given a set of obstacles and a grid of regions. Notice that the obstacles need not be convex, and

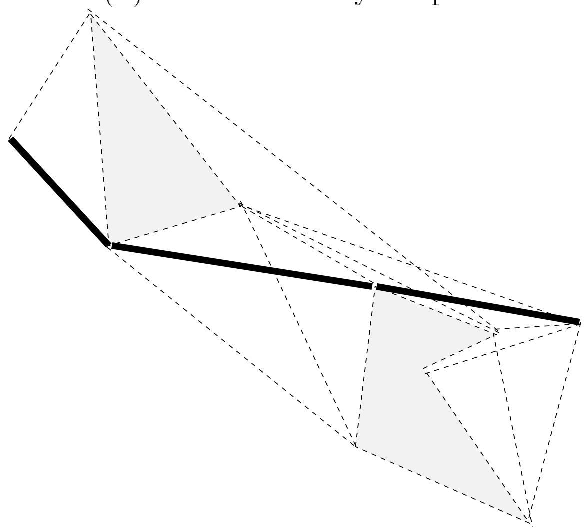
(a) Obstacles in configuration space



(b) Static Visibility Graph

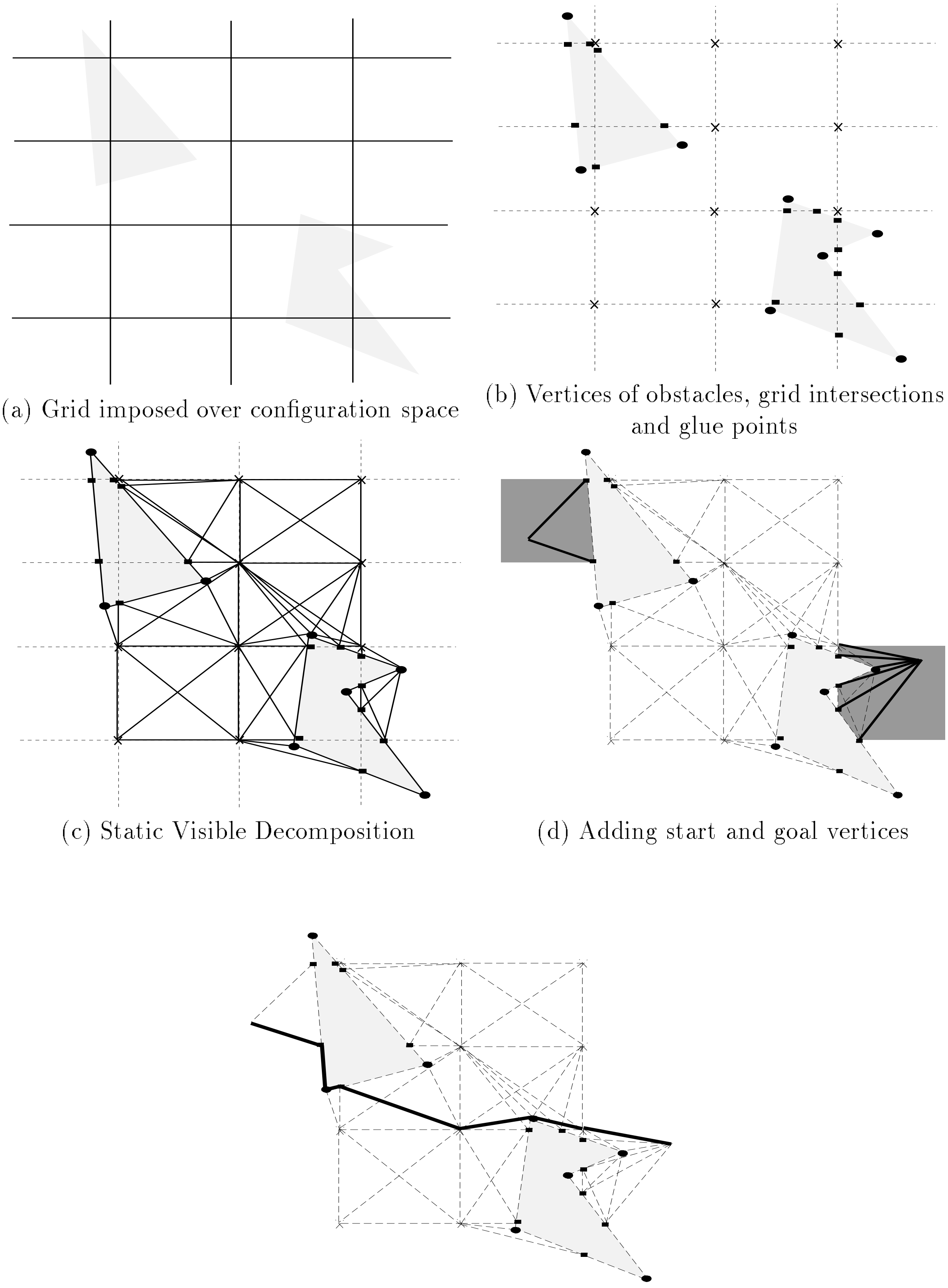


(c) Adding start and goal vertices



(d) A path through the visibility graph

Figure 1: An example of path planning using a Visibility Graph



(a) Grid imposed over configuration space

(b) Vertices of obstacles, grid intersections and glue points

(c) Static Visible Decomposition

(d) Adding start and goal vertices

(e) A path through the global graph

Figure 2: An example of path planning using Visible Decomposition

the grid does not need to be regular and can be represented as an arbitrary decomposition of the configuration space.

- For every edge in every obstacle, check for intersection against grid lines. For each of the two endpoints and each intersection point, add it to the region that it belongs to. The intersection points are shared by more than one region.

- For every region, construct the visibility graph for points in that region, including shared points. In other words, for every pair of points in the region that are "visible", add an edge between them in the global visibility graph. The usual visibility graph speedups such as checking for tangency before visibility still apply. [1]

- In addition to the points shared by two regions when an obstacle overlaps those regions, we need "glue points" in order to make sure that the set of local visibility graphs is globally connected. Otherwise, empty regions can disconnect sections of the global visibility graph. At a minimum, we need to add a point to the corners of every region. These points are connected locally as discussed above.

An example illustrating this computation is shown in Figure 2. Part (a) shows the grid and the regions it generates on the configuration space. In part (b), the circles are the original vertices of the obstacles, the squares are the intersections of the obstacle edges with the grid, and the crosses are the glue points added to make sure that all regions are connected. In part (c), we find the visible edges for each region, and since regions share either glue points or grid intersection points, the resulting graph is connected.

Once the static visibility graph is constructed, we can perform operations such as adding a vertex to the graph, adding an obstacle, or removing an obstacle using only local computations. For example, to add a start or goal vertex:

- Find the region that contains the vertex, and find the visible edges from the vertex to the *local points* maintained by that region. Add the vertex and those edges to the visibility graph.

To add an obstacle:

- Update the visibility graph for those regions that overlap the obstacle. The simplest method is simply to recompute the visibility graphs in those regions from scratch. More elaborate methods are possible, but we have not felt the need for them.

In order to plan a path, we insert the start vertex and the goal vertex, and then search the global graph for a path between the two. The insertion operations have been speeded up considerably by the decomposition of the configuration space, but the search is potentially slowed down by the addition of grid intersection vertices and glue points. In fact, the more we decompose the space, the closer we move toward the Cell Decomposition approach in which most cells are uniform (either empty or full) and we look for a path through them.

[1] If an edge in a visibility graph is not tangent at both endpoints, then it can not be part of the shortest path.

Unfortunately, as we move toward that approach, the search cost increases. We attempt to analyze the necessary computations and explore this tradeoff in the next section.

An example of path planning is shown in Figure 2. In part (d), we highlight the regions where the origin and goal points are located. We only attempt to add visible edges to points within those regions. Part (e) shows the shortest path through that graph. As can be seen, the number of vertices is slightly increased over the one shown in Figure 1 and the path is slightly longer, but the amount of computation needed to insert the origin and goal vertices is much smaller since only edges to the vertices in the highlighted regions need to be examined.

2.1 Analysis

The analysis in this section is an average worst case of the Visible Decomposition algorithm. We compare it with the running time of the global visibility graph approach. Some empirical comparisons are shown in Section 3.

The key computation in these algorithms is the line intersection routine. This is a basic routine which is called repeatedly, and it is relatively expensive for such a basic operation. Therefore, the cost of an operation will be measured by how many line intersection checks it performs. The time will grow with respect to n, the number of points in the set of obstacles. There are two parts to the analysis: computation of the static visibility graph, and inserting a new vertex to the graph. The former is less important since this computation only needs to be performed rarely and its results can then be stored. The latter is what needs to be performed repeatedly in real-time. Incremental updates on obstacles (addition or deletion of an obstacle) also need to be performed quickly.

For global visibility graphs, the simple method of inserting a point into the graph is $O(n^2)$ since there are n possible visibility edges eminating from that point, each of which needs to be checked for intersection with the n edges that make up the obstacles in the world. Then, computing the entire static graph is $O(n^3)$ since we need to insert all n points, each of which takes $O(n^2)$. This is the naive implementation, and there are methods which are theoretically faster [Asano *et al.*, 1986]. However, they tend to be complicated and are slower in practice than the naive method.

For the analysis of the Visible Decomposition approach, we assume that the points are on average uniformly distributed through the configuration space. This assumption is in general not true since even if the obstacles are uniformly distributed, their vertices are not. In addition, we assume that we decompose the world into a $k \times k$ grid, and each of the regions is equal size.

Therefore, there are n/k^2 points contributed by the original obstacles in each region. However, in the worst case each edge of every polygon intersects all $2k$ grid lines, giving us an extra $O(nk)$ points in the configuration space. The glue points get swallowed up (since $n \gg k$), and there are actually about $O(\frac{n+nk}{k^2}) \sim O(n/k)$ points per region. [2]

The initial static part of Visible Decomposition performs k^2 visibility graph constructions, with each construction building on $O(n/k)$ points. Since each construction takes $O((\frac{n}{k})^3)$, the entire initialization takes $O(\frac{n^3}{k})$. This is a factor of k faster than the traditional visibility

[2]Since most obstacles usually do not fall into the worst case category, the number of points per region is usually closer to $O(n/k^2)$ than to $O(n/k)$.

graph approach.

Inserting a point using Visible Decomposition uses the same analysis as for the traditional approach, except that there are only $O(n/k)$ points to try to connect the point to, and likewise only $O(n/k)$ edges that the potential visible edge can intersect. Therefore, inserting a point is done in $O(n^2/k^2)$, a speedup of k^2 over the visibility graph approach.

Ideally, we would increase k until there were less than 1 point per region on average. Unfortunately, the number of total points (due to grid intersections and glue points) in the global graph would increase as $O(nk)$. A search technique such as Dijkstra's shortest path works in $O(nk \lg(nk))$. Clearly, there is a value of k for which the extra expense incurred in search is greater than the time saved in inserting the point. The optimal value for k minimizes the cost of insertion and search. Despite some abuse of notation, we can solve for when the derivative of this sum with respect to k is zero. We find that the optimal value for k is $O(n^{\frac{1}{3}+\epsilon})$, for arbitrarily small ϵ. Therefore, the optimal number of regions grows slowly with the complexity of the world.

In Section 3, we will show that this analysis holds in the real world and that empirically the insertion time drops dramatically, the search time rises slowly, and the optimal value for k is small with respect to n.

2.2 Path Length

One of the advantages of the traditional Visibility Graph approach is that the paths it generates are of optimal length. This property is not conserved by the Visible Decomposition method since paths that go through more than one region must use the points on the region boundary as "portals". Intuitively, as the number of regions or the number of additional glue points grows, the paths get closer to their optimal length, at the expense of additional computation. In this section we will show that in fact, the path length does not converge to the optimum as the number of regions grow. The path length does converge to optimum as the number of glue points grows. These claims will be proved in this section, and empirical evidence is shown in section 3.

2.2.1 Increasing the number of regions

For simplicity, the analysis will be done on an empty, square world with no obstacles. However, the basic results hold on a non-empty world as well. With no partitioning, the average distance between two points in a square with sides of length s is [Santaló, 1976]

$$OPT = s\frac{\sqrt{2} + 2 + 5\log(1+\sqrt{2})}{15} \tag{1}$$

We would like to compare this to the average distance between two points in a unit square with the following restriction: there is a $k \times k$ grid imposed on the square and if the points fall in different regions, the path between them must go through the grid intersections. There are three cases: (a) the points fall in the same region, (b) the points fall on the same row or column of regions (but not in the same region), (c) the points are not in the same region nor in the same row or column. We need to find the probability ($\Pr\{\cdot\}$) of each of the three cases happening, and the average distance ($AvgD(\cdot)$) between the two points in each case. Once we have those, we can compute

$$\lim_{k\to\infty} \frac{\Pr\{a\}AvgD(a) + \Pr\{b\}AvgD(b) + \Pr\{c\}AvgD(c)}{OPT} \tag{2}$$

Finding the probabilities is easy:

$$\Pr\{a\} = \frac{1}{k^2} \tag{3}$$

$$\Pr\{b\} = \frac{2k-1}{k^2} \tag{4}$$

$$\Pr\{c\} = 1 - (\Pr\{a\} + \Pr\{b\}) = \frac{k-2}{k} \tag{5}$$

If the points fall in the same region, then we can use Equation 1 to say that the average distance between two points in square of size $\frac{1}{k} \times \frac{1}{k}$ is

$$AvgD(a) = \frac{1}{k}\frac{\sqrt{2} + 2 + 5\log(1+\sqrt{2})}{15} \tag{6}$$

If the points fall in the same row or column of regions, then we can assume without loss of generality (i.e. the average distance does not change) that they fall in the same row and and that the first point lies in a region with a lower x-coordinate than the second point. The distance between the two points consists of three parts: the distance from the first point to its region's upper-right or lower-right corner (b_1), the distance from the second point to its region's upper-left or lower-left corner (b_3), and the distance between the region corners (b_2). An example of this situation is shown in Figure 3.

The average sum of b_1 and b_3 can be lower bounded by 0. [3] The average distance between the two region corners is

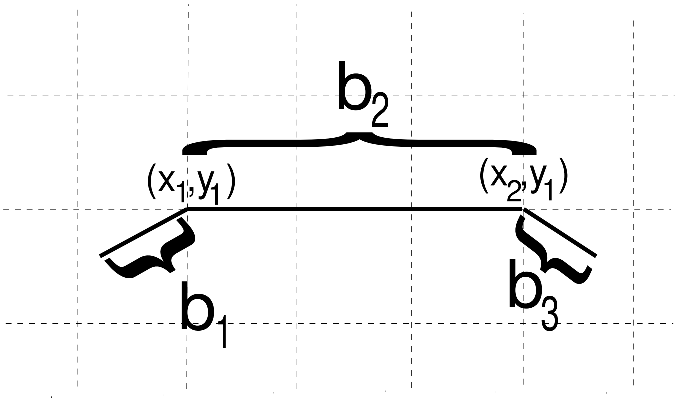
$$E[b_2] = \frac{1}{k}\frac{\sum_{x_1=0}^{k-2}\sum_{x_2=x_1+1}^{k-1} x_2 - x_1 - 1}{\sum_{x_1=0}^{k-2}\sum_{x_2=x_1+1}^{k-1} 1} = \frac{k-2}{3k} \tag{7}$$

Therefore,

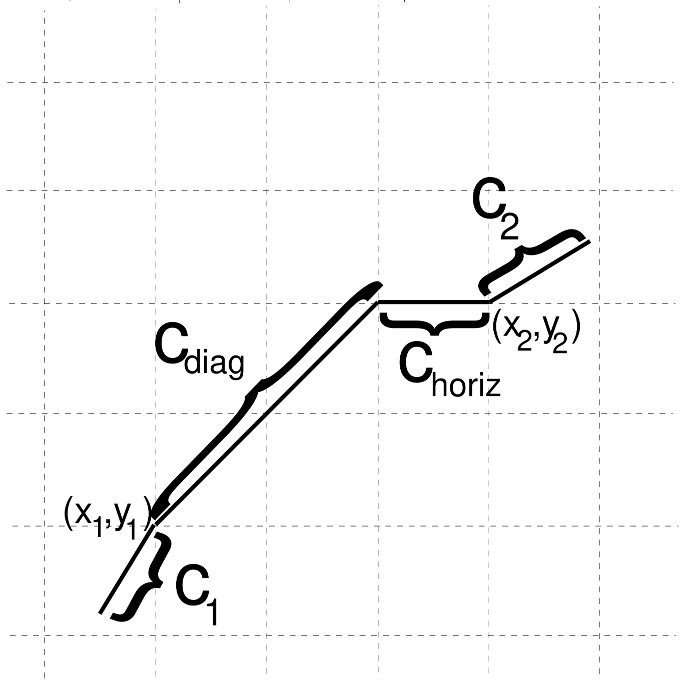
$$AvgD(b) = E[b_1 + b_2 + b_3] > \frac{k-2}{3k} \tag{8}$$

If the points do not fall in the same row or column of regions, then we can assume without loss of generality that the first point is in a region which is "below" the region that the second point falls in. The distance between the two points consists of four parts: the distance between the first point and its region's upper right corner (c_1), the distance between the second point and its region's lower left corner (c_2), the distance along the diagonal path from the lower region to the upper's region row or column (c_{diag}), and the distance up the row or column to the second region (c_{horiz} or c_{vert}). An example of this situation is shown in Figure 3.

[3] It is actually about $1.3/k$, but we want to show that even a lower bound on the average Visible Decomposition path is higher than the optimal path length. In addition, as $k \to \infty$, case (c) dominates.



Case (b) of the analysis, in which both points fall in the same row or column. The path between them must consist of three parts: two segments from the points to their respective region corners, and a (potentially zero-length) segment between the region corners.



Case (c) of the analysis, in which the points do not fall in the same row or column. The path between them must consist of a diagonal section, a vertical or horizontal section, and two segments from the points to their respective region corners.

Figure 3: Two cases from the analysis of average Visible Decomposition path length

The average lengths of c_1 and c_2 are independent, and since the average distance to every corner must be equal, the average length to the corner is the radius of the region. In other words, $E[c_1] = E[c_2] = \frac{\sqrt{2}}{2k}$.

Now we calculate $c_{diag} + c_{horiz} + c_{vert}$: the path from the first region's upper right corner (x_1, y_1) to the second region's lower left corner (x_2, y_2), where $x_1 < x_2$ and $y_1 < y_2$. The length of the path along the diagonal is $c_{diag} = \frac{\sqrt{2}}{k}(\min(x_2 - x_1, y_2 - y_1) - 1)$. The length of the vertical and horizontal part of the path (at least one of which must be zero) is $c_{vert} = \frac{1}{k}(y_2 - y_1 - \min(x_2 - x_1, y_2 - y_1))$ and $c_{horiz} = \frac{1}{k}(x_2 - x_1 - \min(x_2 - x_1, y_2 - y_1))$, respectively. The average distance of the path along the diagonal and up the row or column is therefore

$$\begin{aligned} E[c_{diag} + c_{vert} + c_{horiz}] &= \frac{\sum_{x_1=0}^{k-2} \sum_{y_1=0}^{k-2} \sum_{x_2=x_1+1}^{k-1} \sum_{y_2=y_1+1}^{k-1} c_{diag} + c_{vert} + c_{horiz}}{\sum_{x_1=0}^{k-2} \sum_{y_1=0}^{k-2} \sum_{x_2=x_1+1}^{k-1} \sum_{y_2=y_1+1}^{k-1} 1} \\ &= \frac{(k-2)((4+3\sqrt{2})k^2 + (2-6\sqrt{2})k + \sqrt{2} - 2)}{15k^2(k-1)} \end{aligned} \tag{9}$$

and therefore,

$$\begin{aligned} AvgD(c) &= E[c_{diag} + c_{vert} + c_{horiz} + c_1 + c_2] \\ &= \frac{(k-2)((4+3\sqrt{2})k^2 + (2-6\sqrt{2})k + \sqrt{2} - 2)}{15k^2(k-1)} + \frac{\sqrt{2}}{k} \end{aligned} \tag{10}$$

We now have all the components for finding the average path length between two points where the path must go through the grid intersections. $\Pr\{a\}AvgD(a) + \Pr\{b\}AvgD(b) + \Pr\{c\}AvgD(c)$ is at least:

$$\begin{aligned} &[-20 + 3\sqrt{2} - 5\log(1+\sqrt{2}) + (53 - 3\sqrt{2} + 5\log(1+\sqrt{2}))k + \\ &\quad (-8\sqrt{2} - 29)k^2 + (-4 - 3\sqrt{2})k^3 + (4 + 3\sqrt{2})k^4] \;/\; 15(k-1)k^3 \end{aligned} \tag{11}$$

As we make the grid finer and finer (i.e. as $k \to \infty$), Equation 11 converges on $\frac{4+3\sqrt{2}}{15} \approx 0.55$, about 5.5% longer than the average optimal path (Equation 1) which is about 0.521. The intuition behind this is that as the number of subdivisions increases, the deviations from the optimal path get smaller and smaller, but the number of deviations gets larger and larger.

Both the grid path and the optimal path scale linearly with the length of the side of the square. This analysis was done in an empty world, and adding obstacles to the world helps the grid path. That is because any intersection of an obstacle with a grid line generates an additional "portal" point to cross from region to region. However, the convergence behavior described in this section persists, and the length of the average Visible Decomposition path still does not match the optimal path length.

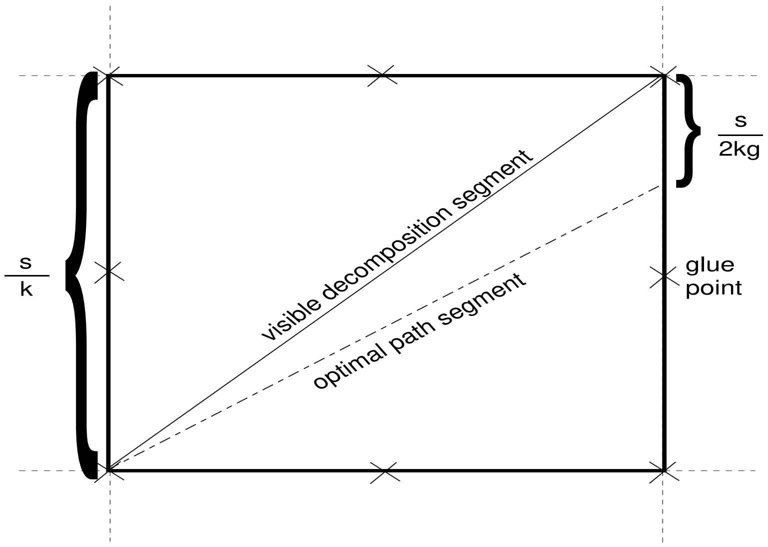


Figure 4: Worst case difference between the optimal path segment and the visible decomposition path segment. In this diagram, the number of glue points per edge,g, is 2.

2.2.2 Increasing the number of glue points

As the number of glue points increases, the average path length *does* converge to the optimal average. To show this, we again consider an empty square world of size $s \times s$, subdivided into $k \times k$ regions (in this case k is held constant). There are g glue points on every side of each of the k^2 regions and they are uniformly spaced.

If two points are picked at random, they will be on average $k/3$ region-rows away from each other and $k/3$ region-columns away from each other. In other words, the line connecting them will go through $2k/3$ regions on the average.

We will now discuss the worst case difference between the optimal path segment and the visible decomposition path segment that goes through a particular region. The worst case occurs when (1) the optimal path segment starts at the lower left corner of the region and crosses the right edge of the region as it exits, and (2) the visible decomposition path segment is the longest possible segment in the region — the diagonal. This situation is depicted in Figure 4. Since the edge of the region is of length s/k, the diagonal is of length $s\sqrt{2}/k$. We know that the optimal segment hits the right edge of the region at most $1/2g \cdot s/k$ below the top right corner. Since there are g glue points (one of them is the corner), there is another glue point on the right edge, s/gk below the corner. Therefore, if the optimal path segment crosses the right edge more than $s/2gk$ below the corner, then the visible decomposition path segment would not go through the corner glue point, but through the glue point below the corner.

We can now compute the distance of the optimal path segment in that region. It is

$$\frac{s}{2kg}\sqrt{8g^2 - 4g + 1} \tag{12}$$

As the number of glue points increases ($g \to \infty$), Equation 12 converges to $s\sqrt{2}/k$, which is the worst case length of the Visible Decomposition segment. Therefore, the difference between the optimal and visible decomposition path segments converges to zero. The total difference of the path length also converges to zero since the path goes through only $2k/3$ regions on the average.

2.3 Path smoothing

While adding glue points is a good way to decrease path length, it is very memory intensive since many new points need to be added to the visibility graph. An alternative approach is to smooth the path after it is generated. The smoothing techniques described in this section do not add new points, but they are asymptotically inefficient (i.e. global) in time. However, the expense incurred in practice is negligible.

We rely on the triangle inequality in order to smooth the path. The unsmoothed path is a list of vertices $(v_1, \ldots, v_n)$, where the first and last vertices are the origin and goal. For every triple of vertices (v_i, v_{i+1}, v_{i+2}) for $i = 1 \ldots n-2$, check if $dist(v_i, v_{i+1}) + dist(v_{i+1}, v_{i+2}) > dist(v_i, v_{i+2})$. If so, and the edge from v_i to v_{i+2} is visible, then remove the vertex v_{i+1}.

This algorithm constitutes one smoothing pass. We can pass over the list of vertices repeatedly until there is no change. The expensive and non-local part of the algorithm is the visibility check. It is possible that we need to check for intersection of every obstacle in the world with every segment of the path. In practice however, even multi-pass smoothing can be done in real time.

This smoothing technique is an instance of filtering the path with a narrow filter. It is possible to extend it to filters of growing widths, and recursive schemes. For example, trying to smooth triple $(v_1, v_{n/2}, v_n)$, and then the two triples $(v_1, v_{n/4}, v_{n/2-1})$ and $(v_{n/2}, v_{3n/4}, v_n)$, and so on.

3 Results

The Visible Decomposition algorithm was implemented and tested on various terrains from the ModSAF database. The complexity of these worlds varied from 150 to over 2000 obstacles, and the obstacles themselves ranged from well defined tree canvases to rivers to swamps. The configuration space is two-dimensional, but there are multiple copies of it since there are many different widths that the collection of vehicles can assume. The following results are from experiments on a 150 obstacle world with about 6500 vertices. This relatively small problem (by the standards of these simulations) is used to enable us to do extensive testing in finite time. The algorithm has also been run on the larger databases and achieves the desired interactive performance, that is, a user can drag a point over the whole terrain while the a collision-free path is continuously being found (from scratch) from the origin to the point being dragged.

The graph in Figure 5 shows the CPU time on an SGI Indigo2 (in centiseconds) spent on inserting the goal and start vertices as k grows (the world is divided into a $k \times k$ grid). It also shows the time spent on searching the resulting graph using both Dijkstra's shortest path and A*. The topmost point, $k = 1$, represents the traditional visibility graph approach, and it can be seen that as k increases, the insertion time decreases quickly until the total

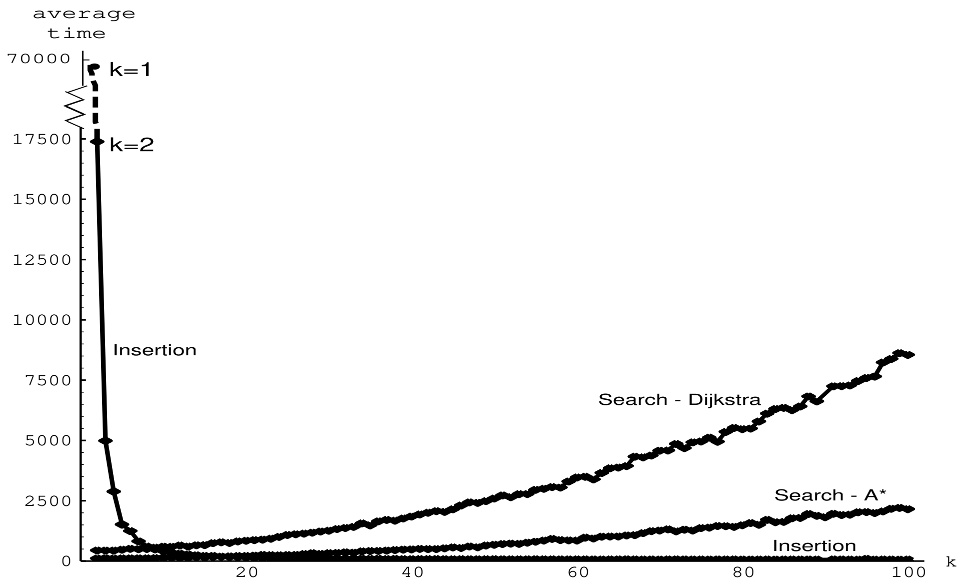


Figure 5: Time for vertex insertion and search as k increases. $k = 1$ is the global visibility graph approach.

planning time is dominated by the search. In addition, it is clear that the value of k for which both insertion and search is minimized is low. The timing is averaged over 500 random paths for each value of k.

The graph in Figure 6 shows how the length of paths generated by Visible Decomposition compares with the optimal paths produced by the traditional visibility graph method. We chose 500 random pairs of start and goal vertices, and used this set for measuring the average path length for increasing values of k. The optimal path length is the base line at the bottom. The top line shows results without smoothing. The second line shows results with one-pass triangle inequality smoothing. The third line shows results with multiple-pass triangle inequality smoothing, as described in Section 2.3.

In the example shown in Figure 2, there is one glue point associated with each region; namely, one in the lower left corner (the region's neighbors provide for the other corners). This is the minimum required for connectivity, but doesn't lead to very good looking paths. In the experiments described above, there are 3 glue points for each region — the bottom left corner, the midpoint of the left wall, and the midpoint of the floor.

As can be seen, the paths generated using Visible Decomposition are only about 1% longer than optimal, and can be made shorter using very simple smoothing techniques. However, as the analysis of the previous section showed, increasing the subdivision does not lead to convergence to the optimal path length.

Figure 7 shows how the average path length decreases as the number of glue points per region increases. The paths are the same set as the ones used for Figure 6. The number of regions was held constant at 15×15, and the number of glue points per region was increased from 1 to 60. Increasing the number of glue points results in convergence to the optimal path length, agreeing with the analysis. However, there is extra computation as can be seen in Figure 8. The top line shows the search time using Dijkstra as the number of glue points

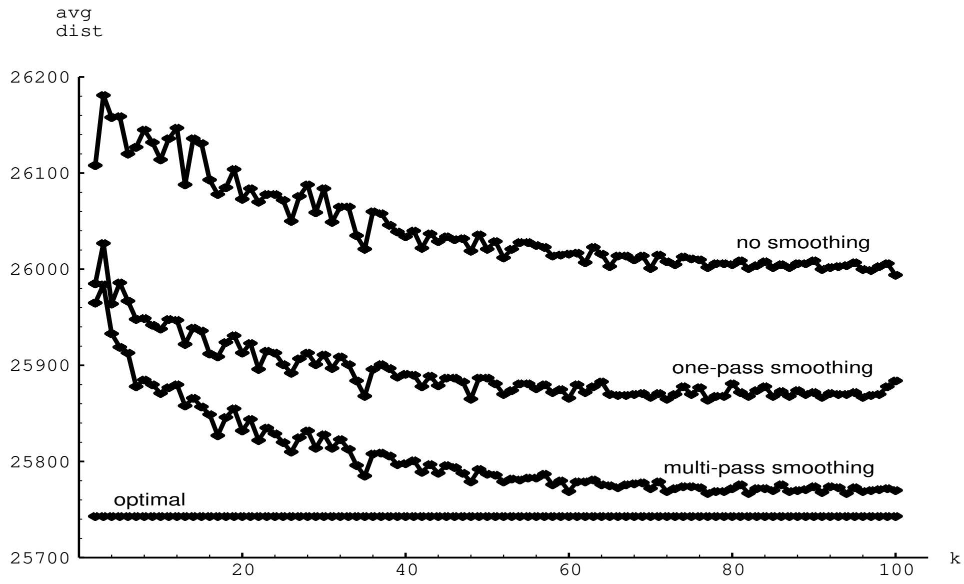


Figure 6: Average path length as k increases. The base line is the optimal $k = 1$. The top plot uses Visible Decomposition. The second plot smoothes using 1-pass triangle inequality. The third plot smoothes using multiple-pass triangle inequality.

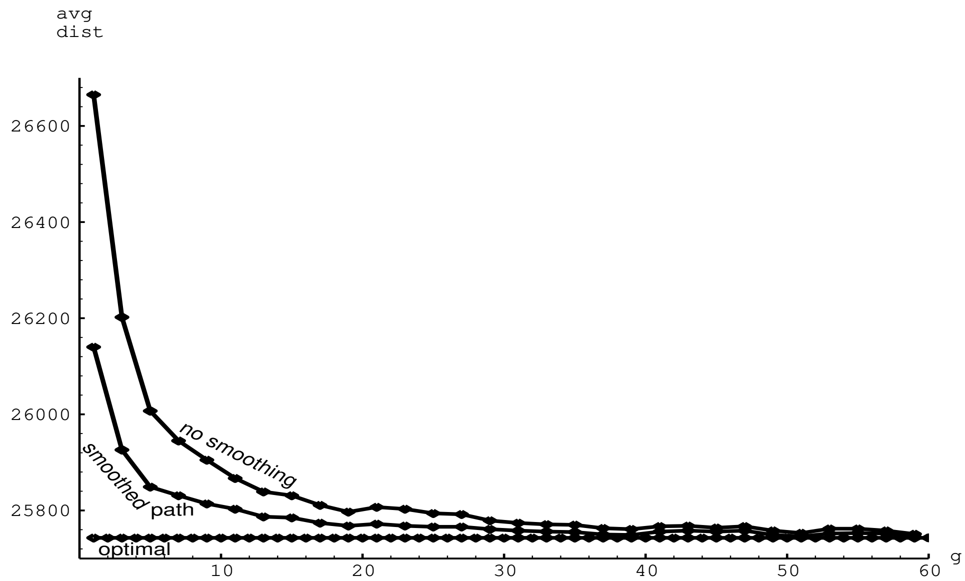


Figure 7: Average path length as g, the number of glue points, increases. The base line is the optimal path length with $k = 1$. The decomposition is held constant at $k = 15$.

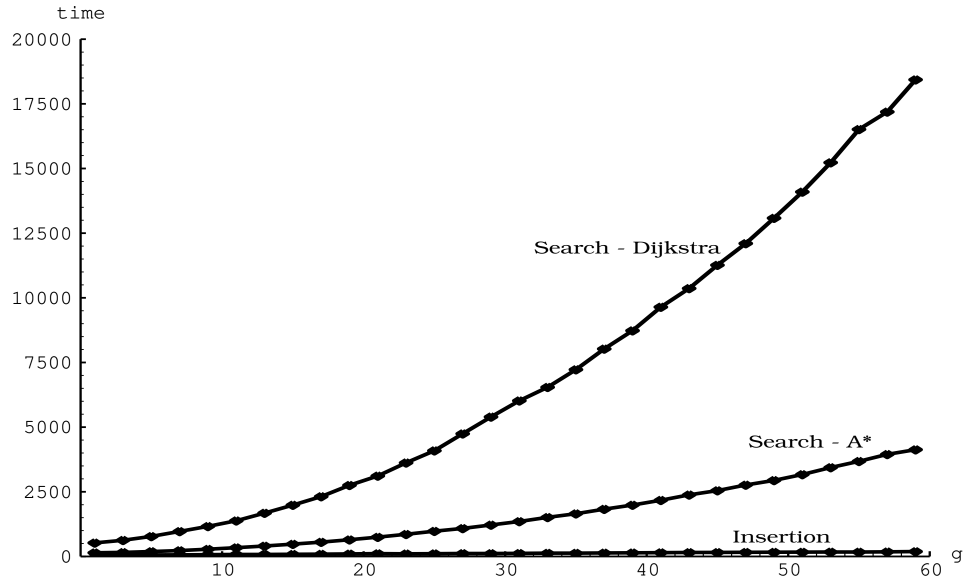


Figure 8: Average computation time as g, the number of glue points, increases. The decomposition is held constant at $k = 15$. The top line shows the time taken by Dijkstra, the middle line by A*, and the bottom line by inserting the origin and goal vertices.

increases, the second line shows the search time using A*, and the bottom line shows the time taken to insert the origin and goal vertices.

4 Conclusions

In this paper we have discussed a technique called Visible Decomposition for local construction and update of visibility graph planning. The method represents a tradeoff between the visibility graph approach and the cell decomposition approaches. The tradeoff is parameterized by the number of local regions. We analyze a decomposition into a world of $k \times k$ squares, and show that the optimal value for k only grows as $O(n^{\frac{1}{3}+\epsilon})$, where n is the number of vertices of all the obstacles in the world. This means that the number of regions grows very slowly compared to the size of the world.

We have shown that the method is well suited for real-time planning in complex, planar environments. We have also shown that the paths generated by using Visible Decomposition are not much longer than the optimal paths, and can be made to converge by smoothing and adding glue points.

Future extensions should include variable density decomposition with more regions in the areas with more obstacle vertices.

References

[Asano *et al.*, 1986] Takao Asano, Tetsuo Asano, Leonidas Guibas, John Hershberger, and Hiroshi Imai. Visibility of disjoint polygons. *Algorithmica*, 1:49–63, 1986.

[Brock *et al.*, 1992] D.L. Brock, D.J. Montana, and A.Z. Ceranowicz. Coordination and Control of Multiple Autonomous Vehicles. In *Proc. of IEEE International Conference on Robotics and Automation*. IEEE, 1992.

[Brooks and Lozano-Pérez, 1985] R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. *"IEEE Transacations on Systems, Man and Cybernetics"*, SMC-15(2):224–233, 1985.

[Canny and Donald, 1988] J. Canny and B. Donald. Simplified voronoi diagrams. *Discrete and Computational Geometry*, 3:219–236, 1988.

[Latombe, 1991] Jean-Claude Latombe. *Robot motion planning*. Kluwer Academic, 1991.

[Lozano-Pérez and Wesley, 1979] Tomás Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22:560–570, 1979.

[Nilsson, 1984] Nils J. Nilsson. Shakey the robot. SRI TN-323, SRI International, 1984.

[Ó'Dúnlaing and Yap, 1982] C. O. Ó'Dúnlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6(1):104–111, 1982.

[Santaló, 1976] L. A. Santaló. *Integral Geometry and Geometric Probability*. Encyclopedia of Mathematics and its applications. Addison-Wesley, 1976.