14

An $O(n^2)$ Shortest Path Algorithm For A Non-Rotating Convex Body

by John Hershberger and Leonidas J. Guibas

November 27, 1986

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

Systems Research Center

DEC's business and technology objectives require a strong research program. The Systems Research Center and two other corporate research laboratories are committed to filling that need.

SRC opened its doors in 1984. We are still making plans and building foundations for our long-term mission, which is to design, build, and use new digital systems five to ten years before they become commonplace. We aim to advance both the state of knowledge and the state of the art.

SRC will create and use real systems in order to investigate their properties. Interesting systems are too complex to be evaluated purely in the abstract. Our strategy is to build prototypes, use them as daily tools, and feed the experience back into the design of better tools and the development of more relevant theories. Most of the major advances in information systems have come through this strategy, including time-sharing, the ArpaNet, and distributed personal computing.

During the next several years SRC will explore applications of high-performance personal computing, distributed computing, communications, databases, programming environments, system-building tools, design automation, specification technology, and tightly coupled multiprocessors.

SRC will also do work of a more formal and mathematical flavor; some of us will be constructing theories, developing algorithms, and proving theorems as well as designing systems and writing programs. Some of our work will be in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. We also expect to explore new ground motivated by problems that arise in our systems research.

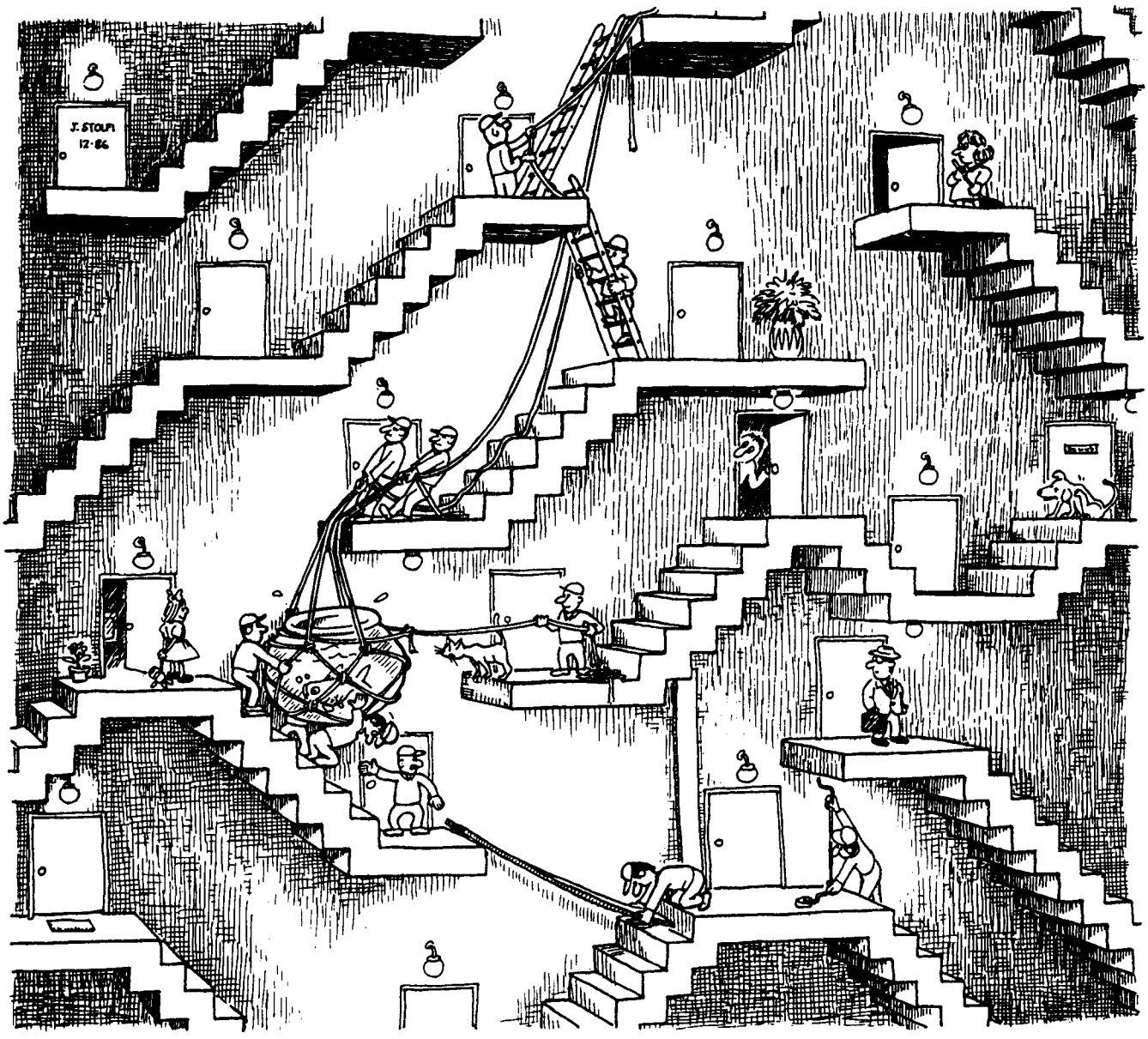
DEC is committed to open research. The Company values the improved understanding that comes with widespread exposure and testing of new ideas within the research community. SRC will therefore freely report results in conferences, in professional journals, and in our research report series. We will seek users for our prototype systems among those with whom we have common research interests, and we will encourage collaboration with university researchers.

Robert W. Taylor, Director

An $O(n^2)$ Shortest Path Algorithm for a Non-Rotating Convex Body

John Hershberger and Leonidas J. Guibas

November 27, 1986



John Hershberger is a member of the Computer Science Department at Stanford University, Palo Alto, California. His work was supported in part by US Army Research Office Fellowship number DAAG29-83-G0020.

Authors' abstract

We investigate the problem of moving a convex body in the plane from one location to another while avoiding a given collection of polygonal obstacles. The method we propose is applicable when the convex body is not allowed to rotate. If n denotes the total size of all polygonal obstacles, the method yields an $O(n^2)$ algorithm for finding a shortest path from the initial to the final location. In solving this problem, we develop some new tools in computational geometry that may be of independent interest.

John Hershberger and Leonidas J. Guibas

Capsule review

In this paper the authors consider the problem of finding the shortest path by which a convex plane body—called the *robot*—can be moved, without rotation, from one point in the plane to another while avoiding a collection of disjoint simple polygonal obstacles. Previous researchers have studied special cases of this problem in which the shape of the robot is a point, a convex polygon, or a disk, and have found shortest-path algorithms that run in time $O(n^2)$, where n is the total number of edges in the set of obstacles. The algorithm presented in this paper achieves the same time bound for robots of arbitrary convex shape.

The authors begin by observing that any path of the robot avoiding the obstacles corresponds to a path of a single point avoiding a collection of *fattened* obstacles, which are obtained by replacing each point in each of the original obstacles with an inverted copy of the robot. A shortest path from point α to point ω avoiding the fattened obstacles may then consist of two kinds of fragments: straight segments each of whose endpoints is either α, ω, or a point of tangency to a fattened obstacle, and portions of the perimeters of fattened obstacles joining these points of tangency. A shortest path for the robot can therefore be found by applying a graph-theoretical shortest-path algorithm to an *augmented path graph* whose edges are (approximately) those fragments listed above that are not blocked by fattened obstacles, and whose vertices are (approximately) α, ω, and the various points of tangency.

Unfortunately, the augmented path graph may have $\Theta(n^2)$ vertices and $\Theta(n^2)$ edges. Finding a shortest path in such a graph may require $\Theta(n^2 \log n)$ time. The meat of the paper is devoted to a sequence of constructions which, by taking advantage of the convexity of the robot, reduce the problem to one of finding shortest paths in a *coalesced path graph* having only a $O(n)$ vertices. Shortest paths in such a graph can be found in $O(n^2)$ time by using Fibonacci heaps to implement Dijkstra's algorithm. The work of building the augmented path graph and reducing it to the coalesced path graph is also achieved in $O(n^2)$ time, with the constant factor depending on the time required to perform certain primitive operations, such as constructing the common tangents of two translated copies of the robot.

Jim Saxe

Contents

1. Introduction

The Euclidean shortest path problem formalizes a common question: what is the easiest way to move an object from one location to another? An instance of the problem specifies a set of obstacles S and initial and final positions α and ω for the object. The desired solution is a path from α to ω that avoids the obstacles and has minimal Euclidean length, if any such path exists. There are many variants of the problem: the problem may be posed in two, three, or more dimensions; the obstacles may be subject to constraints (convexity, for example); and the moving object may have constraints on its shape or motion. This paper focusses on the two-dimensional case in which S is a set of disjoint simple polygons with a total of n vertices.

A number of recent algorithms find shortest paths among polygonal obstacles in the plane. The methods of Reif and Storer [RS] and of Asano et al. [AAGHI] take $O(n^2)$ worst-case time to find shortest paths for a moving point. The problem of finding shortest paths for a moving body of finite size is, however, more difficult. In the special case when the body is a non-rotating convex polygon of fixed complexity, a simple extension of the point-motion algorithms runs in $O(n^2)$ time. Baker and Chew propose an $O(n^2 \log n)$ algorithm to find shortest paths for a disk moving among polygonal obstacles [Ba][Ch]. Reif and Storer give an $O(n^2)$ solution to the same problem. Their method exploits the geometry of the obstacle space to run faster than $O(n^2)$ under some conditions; we discuss their approach in Section 8.

In this paper we solve a generalization of the disk motion problem. We show how to find shortest paths for a non-rotating convex body. The obstacles we consider are disjoint simple polygons, as in the work cited above; our method is novel because the moving body is not constrained to be polygonal or round, but only convex. The arbitrary shape of the convex body is not without its costs, but we defer discussion of these costs until Section 5.

Our approach uses two ideas from the literature to simplify the problem of finding shortest paths. The first simplification reduces the problem of moving a non-rotating convex object among polygons to that of moving a point (the object's center) among "fattened" versions of the obstacles. This idea is due to Lozano-Perez and Wesley [LPW]. The second simplification, due to Baker and Chew [Ba][Ch], reduces the problem of finding shortest paths for a point among fattened obstacles to that of finding a shortest path in a graph. Dijkstra's shortest path algorithm for graphs [AHU] can be used to solve the reduced problem.

The shortest path algorithm we present builds the graph of Baker and Chew, then simplifies it before invoking Dijkstra's algorithm. Each step takes $O(n^2)$ time and space. The visibility graph of the polygons, obtained using the algorithm of Asano et al. [AAGHI] or Welzl [W],

forms the basis for constructing Baker and Chew's graph. If the graph so constructed has $\Omega(n^2)$ nodes, then running Dijkstra's algorithm on it will take $\Omega(n^2 \log n)$ time, causing a bottleneck. To circumvent this potential problem, the algorithm reduces the graph to an equivalent one with only $O(n)$ nodes before running Dijkstra's algorithm. Using the Fibonacci heaps of Fredman and Tarjan [FT], Dijkstra's algorithm takes only $O(n^2)$ time when applied to the reduced graph.

2. Definitions

This section gives the special notations used in the paper. It also explains the idea of "fattening" obstacles mentioned in the introduction.

We represent individual points by lowercase letters and sets of points by uppercase letters. The segment between points p and q is $\overline{pq}$, and the distance from p to q is $|pq|$. This distance is usually the straight-line distance between the points, though we sometimes use $|pq|$ to refer to the distance from p to q along the boundary of a convex region on which both points lie.

We refer to the moving convex body as the *robot* and represent it by the letter A. The robot must avoid obstacles as it moves. These obstacles are disjoint simple polygons with a total of n vertices. The robot moves outside all the polygons. (The polygons must be disjoint to satisfy the preconditions of algorithms we use as subroutines.) The set of all polygon points, both vertices and points on segments, is S.

The robot A has a *center*, which is just the coordinate origin in its frame of reference. In that frame, B is the point-wise reflection of A through its center. (Note that the center of A need not lie within its boundary.) We denote the set of points covered by B when its center is placed at a point q by B^q (pronounced "B at q").

To plan the motion of the robot among polygons, we solve the equivalent problem of moving the robot's center among fattened versions of the polygons. To distinguish between the obstacles of the original problem and their fattened versions, we refer to the polygons that the robot avoids as *obstacles* and to the fattened polygons that the robot's center avoids as *barriers*. Points on the boundaries of barriers are especially important to our algorithm; we will refer to these as *boundary points*.

We can draw the barriers by using B as a paintbrush and tracing each polygon's border; the painted areas are the barriers that the robot's center must avoid. The painted region is the Minkowski sum (or vector sum) of B with the polygons and their interiors.† The boundary

† The Minkowski sum of two regions X and Y consists of all points expressible as a vector sum $x + y$, where $x \in X$ and $y \in Y$. The sum is symmetric in its arguments—it is the set $\{x + y \mid x \in X, y \in Y\}$—but it can also be viewed asymmetrically as $\bigcup_{x \in X} Y^x = \bigcup_{y \in Y} X^y$.

of the painted area is closely related to the *convolution* of the boundary of B with the polygons in S, as defined by Guibas, Ramshaw, and Stolfi [GRS].

Any point on the boundary of a barrier must be on the boundary of B^q for at least one polygon point q, and it may not lie inside $B^{q'}$ for any other q' in S. Given a barrier boundary point p, we refer to a polygon point q such that p is on the boundary of B^q as a *generator* of p and denote the set of generators of p by $g(p)$. Unless the boundary of B contains a segment parallel to one of the polygon segments, the number of generators $|g(p)|$ is finite for any point p on the boundary of a barrier. In fact, of those boundary points p with finitely many generators, only a finite number have $|g(p)| > 1$. (We sometimes give the term *generator* a slightly broader meaning; we say that q *generates* the barrier B^q.)

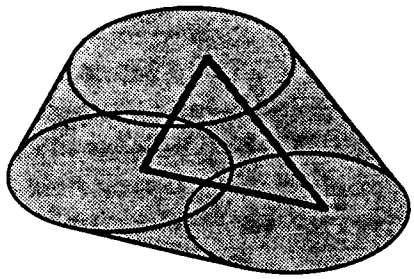


Figure 1. When a convex obstacle polygon is fattened by the inverted robot B, the outer boundary of the resulting barrier has arcs and segments. The tangent at each of an arc's endpoints matches the slope of the adjacent segment. The tangent's direction varies monotonically between these two extremes along the arc.

We can classify each barrier boundary point p by its generator set $g(p)$. We use the term arc to denote a maximal connected set of boundary points generated by a single polygon vertex. The barrier boundaries are composed of arcs, of straight segments whose generators all lie on a single polygon segment, and of intersection points of these elements.† Each arc copies a portion of the border of B. If an obstacle polygon is convex, then, in the fattened polygon, each arc bridges the difference in slopes between the segments that precede and follow it. (See Figure 1.) When two arcs or segments intersect other than by abutment, the point of intersection has multiple generators. (An arc

† These definitions can break down if the boundary of B contains a segment parallel to one of the polygon segments. In this case, boundary arcs and segments may not be disjoint. To remedy the problem, we

and its adjacent segment intersect at their point of abutment, but that point has only a single polygon vertex as its generator.)

3. The Path Graph

Since combinatorial problems are often easier to solve than geometric ones, we reduce the shortest path problem to a combinatorial problem by introducing the *path graph*. Our path graph is a combinatorial graph structure obtained from the positions of the barriers and is an extension of the one used by Baker and Chew. Each path graph edge corresponds to a path among the barriers. We show that except for its initial and final segments, any barrier-avoiding shortest path is a subset of the path graph edges.

The path graph of a set of barriers is very similar to the *visibility graph* of a set of polygons. The visibility graph of a set of polygons records vertex pairs that can be connected by segments whose interiors are free of polygon points. It is closely related to the *visibility polygon* of a query point, which is the polar sequence of polygon points visible from the query point. (See Figure 2.) Because each non-terminal edge of a shortest path for a point moving among polygons is a polygon edge or a visibility graph edge, the visibility graph is important in shortest path algorithms.

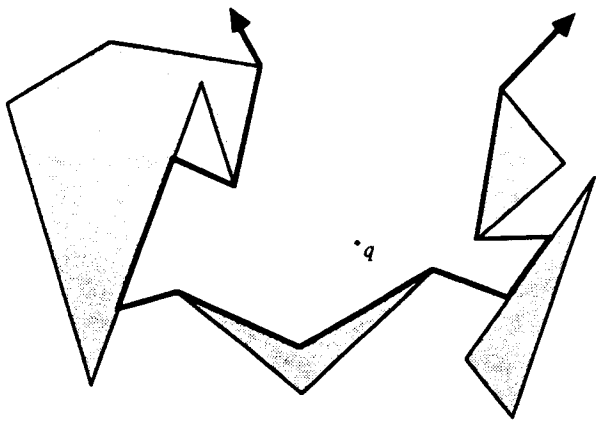


Figure 2. The visibility polygon of query point q is the boundary of the region visible from q in the presence of polygonal obstacles.

---

use the technique of ϵ-perturbation: we define the boundary arcs and segments as if the polygon segment were not parallel to the flat spot on B, but rotated clockwise by an infinitesimal amount ϵ. This results in a consistent definition of disjoint arcs and boundary segments.

The visibility graph records pairs of mutually visible vertices; similarly, the path graph records pairs of barriers connected by common tangents that avoid all other barriers. Because shortest paths for the robot's center follow barrier boundaries as well as common tangents, the path graph has two kinds of edges: portions of the barrier boundaries and tangent segments.

Definition 3.1. *The nodes and edges of the path graph are defined as follows:*

(1) *Every maximal straight boundary segment generated by a single polygon segment is an edge of the path graph, and its endpoints are path graph nodes. This implies that when a segment generated by a polygon segment intersects arcs or other boundary segments, each intersection is a path graph node. Intersections of boundary arcs are also path graph nodes.*

(2) *If a line tangent to two arcs does not intersect any barrier between its points of tangency, the two tangent points are path graph nodes, and the segment connecting them is a path graph edge. If the tangent touches an arc at more than one point (the arc contains a straight segment), we use the tangent point that minimizes the length of the tangent segment. These edges are called* tangent edges.

(3) *Minimal arc sections connecting nodes defined in* (1) *and* (2) *are path graph edges (no two such edges overlap). These edges are convex curves; some may be straight line segments, but only if the boundary of B has flat spots.*

Path graph edges defined in (1) and (3) are called *boundary edges*; their union includes all the boundary arcs and straight segments.

Recall that the path graph of a set of barriers is the analogue of the visibility graph of a set of polygons. Since the visibility graph has $O(n^2)$ edges, it is reasonable to assume that the path graph also has $O(n^2)$ nodes and edges. The following lemma shows that this assumption is true.

Lemma 3.1. *The path graph defined above has $O(n^2)$ nodes and edges.*

Proof: We begin by bounding the number of path graph tangent edges. Consider centering a copy of B at each polygon vertex. These n (possibly overlapping) convex regions have at most $4\binom{n}{2}$ common tangents, since two translated copies of B have at most four common tangents. Because tangent edges are common tangents of boundary arcs, and arcs are generated by polygon vertices, the path graph tangent edges are a subset

of the $O(n^2)$ tangents. Each tangent edge contributes at most two endpoint nodes to the path graph.

Kedem et al. [KLPS] show that there are only $O(n)$ nodes formed by intersections of segments and arcs on the boundary of the barriers. (As in Figure 3, there can be $O(n^2)$ intersections, but only $O(n)$ of them are on the boundary.) These are the nodes of type (1).

Every path graph node has exactly two incident boundary edges. Since there are $O(n^2)$ nodes, there are $O(n^2)$ boundary edges. This fact, in combination with the bounds on nodes and edges already given, proves the lemma. ∎

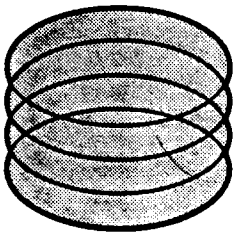


Figure 3. Although n barriers can intersect in $O(n^2)$ points, only a linear number of the intersections are on the boundary of the barriers.

An instance of the shortest path problem specifies the initial and final positions α and ω of A's center, chosen so that A^α and A^ω avoid the polygons. Shortest paths among the fattened polygons begin and end with edges from α and ω, but intermediate edges come from the path graph.

In order to use Dijkstra's algorithm to find the shortest path, we need a graph whose edges contain all shortest paths from α to ω. We produce such a graph, which we call the *augmented graph* G_a, by adding nodes and edges to the path graph. If the segment $\overline{\alpha\omega}$ does not intersect any barrier, it is the shortest path and belongs in G_a. If $\overline{\alpha\omega}$ is blocked, we add to the path graph all unobstructed segments passing through α or ω and tangent to some barrier. The endpoints of the added segments augment the set of path graph nodes. Some boundary edges are split in two by these additional nodes. Because there are at most $4n$ tangents from α and ω to barriers, the added nodes and edges leave the augmented path graph still of size $O(n^2)$. The following lemma shows that G_a is useful for motion planning.

Lemma 3.2. *Every shortest path from α to ω that avoids the barriers follows edges of the augmented path graph, G_a.*

Proof: Any shortest path is composed of subpaths that alternately follow barrier boundaries and move along straight lines between barriers. If the endpoints of the segments that connect barriers are augmented path graph nodes, then the boundary subpaths are made up of boundary edges. Thus it suffices to show that every non-boundary segment on the shortest path is an edge of the augmented path graph.

Suppose that a segment $\overline{ab}$ lies on a shortest path from α to ω, touches barriers only at its endpoints, and is not an edge of the augmented path graph. At least one of $\overline{ab}$'s endpoints, say b, lies on a barrier boundary and is not α, ω, or a point of tangency of $\overline{ab}$ with the barrier. Because b is not α or ω, the path continues beyond b. Some point d on the continuation must be visible from a point c on $\overline{ab}$, since $\overline{ab}$ is not tangent at b. But this means that the supposed shortest path could be shortened by replacing the subpath from c to d via b by the segment $\overline{cd}$, which contradicts the assumption that $\overline{ab}$ lies on a shortest path. (See Figure 4.) ▮

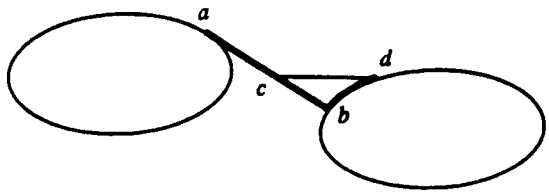


Figure 4. Barrier-connecting segments on a shortest path must be tangents. The dark path through a, b, and d cannot be a subpath of any shortest path. Because b is not a tangent point of $\overline{ab}$, the point d on the continuation of the path is visible from c. Taking the shortcut $\overline{cd}$ gives a path shorter than the dark one.

4. Tangent-Visibility

This section deepens the connection between the path graph of a set of barriers and the visibility graph of the polygons that generate the barriers. It shows that the path graph tangent edges from an arc are closely related to the visibility graph edges from the vertex that generates the arc. The next section exploits this connection to construct the path graph quickly.

If we are given a set of simple polygons with n vertices and a query point q outside the polygons, the visibility polygon of q lists in polar order the polygon points visible from q. (See Figure 2.) Let us denote this (infinite) cyclic sequence of points by $VP(q)$. We can define a similar concept for the fattened polygons by sweeping them with a ray tangent to B^q whose endpoint moves along the boundary of B^q. If the tangent point of the ray is not inside a barrier that overlaps B^q, we say that the first boundary point encountered by the ray is *tangent-visible* from B^q. See Figure 5 for an example. Each query point q has two sequences of tangent-visible points, since each boundary point of B has both a clockwise and a counterclockwise tangent. Both sequences may contain points of tangency that are path graph nodes; since the two sequences are similar, we discuss only the sequence derived from the clockwise tangent. Let us denote the sequence of barrier boundary points swept by the clockwise tangent to B^q by $VS(q)$. The generators of the points in $VS(q)$ form a sequence we call $g(VS(q))$. (If a point p in $VS(q)$ has multiple generators, any one of them may appear in the generator sequence.)

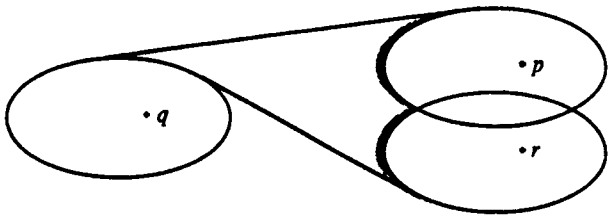


Figure 5. Tangent-visibility. The darkened section of the boundary of B^p and B^r is tangent-visible from B^q.

To help characterize the sequence of generators $g(VS(q))$, we introduce some special notation. As the tangent to B^q sweeps around, it points in each direction between 0 and 2π exactly once. Let p be a point and T a set of points. The point p generates the barrier B^p, and T generates barriers that are the Minkowski sum of T and B. We use the notation $v_T(q,p)$ to refer to the set of directions in $[0,2\pi)$ for which the tangent from B^q hits B^p strictly before hitting any other barrier

B^r for $r \neq p$ and r in T. (It doesn't matter whether p is in T or not.) In this notation $v_\emptyset(q,p)$ is an interval (allowing wraparound across 2π) of measure less than π. The set $v_T(q,p)$ shrinks as T grows; that is, $v_{T\cup\{r\}}(q,p) \subseteq v_T(q,p)$ for any r. The set $v_S(q,p)$ can be expressed as $\bigcap_{r\in S} v_{\{r\}}(q,p)$. For convenience, we let $v_r(q,p)$ stand for $v_{\{r\}}(q,p)$. (See Figure 6 for an example of the notation.) The following lemmas apply this notation to show that the generators of the tangent-visible sequence $VS(q)$ are a subsequence of the visibility polygon $VP(q)$.

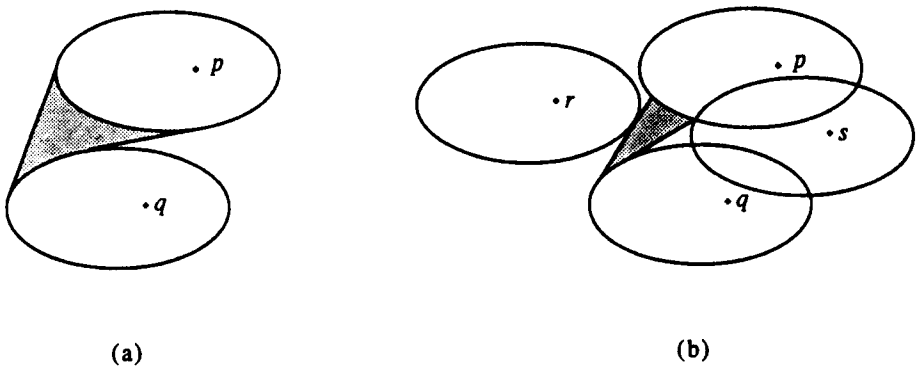


Figure 6. Tangent-visibility notation. When no other barriers are present, B^p is tangent-visible from B^q in the shaded sector shown in (a). The directions of the rays that bound the sector define the interval $v_\emptyset(q,p)$. When barriers B^r and B^s are added, the sector of tangent-visibility is reduced to the shaded region shown in (b). This sector's boundary rays define $v_{\{r,s\}}(q,p)$.

Lemma 4.1. *If p' is a point of the tangent-visible sequence $VS(q)$, all of its generators are in the visibility polygon $VP(q)$.*

Proof: Let $p \in g(p')$. If $p \notin VP(q)$, a polygon point r lies on the segment connecting p and q. The object B^r hides every point of B^p from the sweeping tangent, as shown in Figure 7. Because $v_r(q,p) = \emptyset$, its subset $v_S(q,p)$ is also empty, and p' cannot be in $VS(q)$. ∎

Lemma 4.2. *For any set of barrier generators T, $v_T(q,p)$ is an interval.*

Proof: Since the intersection of intervals of length less than π results in an interval, we need only show that $v_r(q,p)$ is an interval for all r. If $v_r(q,p)$ is not an interval for some r, then let $a < b < c$ be directions in the interval $v_\emptyset(q,p)$ such that a and c are in $v_r(q,p)$ and b is not. (We have assumed

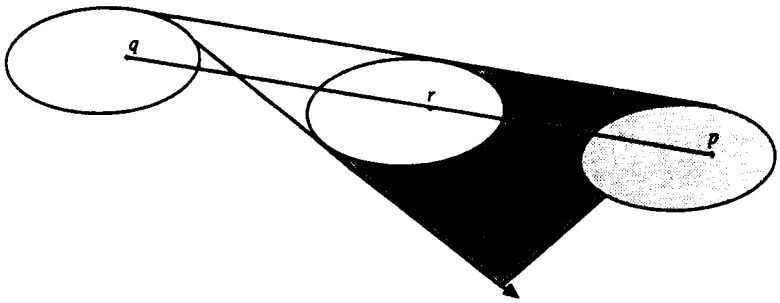


Figure 7. B^r hides B^p from B^q.

without loss of generality that $v_\emptyset(q,p)$ does not include 2π.) In Figure 8, the segments $\overline{u_a v_a}$, $\overline{u_b v_b}$, and $\overline{u_c v_c}$ have directions a, b, and c. Because directions a and c are part of $v_r(q,p)$, the barrier B^r cannot intersect the segments $\overline{u_a v_a}$ and $\overline{u_c v_c}$, even at their endpoints.

We show that B^r cannot intersect $\overline{u_b v_b}$, and hence b is in $v_r(q,p)$. Consider all possible placements of B^r that intersect the shaded region of Figure 8. Note that every placement that abuts the region without overlapping its interior touches $\overline{u_a v_a}$ or $\overline{u_c v_c}$. Since the shaded region is narrower than the parallelogram defined by the outer common tangents to B^q and B^p, any placement of B^r that overlaps the shaded region's interior must also intersect $\overline{u_a v_a}$ or $\overline{u_c v_c}$. Because $\overline{u_b v_b}$ is contained in the shaded region, B^r cannot intersect it. ∎

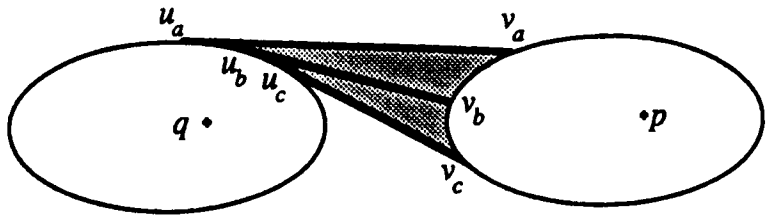


Figure 8. Proof that $v_r(q,p)$ is an interval. B^r cannot intersect $\overline{u_b v_b}$ (block tangent-visibility in direction b) without also intersecting either $\overline{u_a v_a}$ or $\overline{u_c v_c}$.

Lemma 4.3. *If points p, r, and s are in the sequence $g(VS(q))$ and appear in clockwise order around q, then the intervals $v_S(q,p)$, $v_S(q,r)$, and $v_S(q,s)$ are disjoint and also appear in clockwise order.*

Proof: The intervals $v_S(q,p)$, $v_S(q,r)$, and $v_S(q,s)$ are disjoint because p, r, and s all belong to S.

The ray from q to p is parallel to the outer common tangents from B^q to B^p. Let p_t (t for *tangent*) be the direction of this ray, and let p_v (v for *visible*) be any direction in $v_S(q,p)$. We define r_t, r_v, s_t, and s_v similarly. These six directions occur in some clockwise order, though there may be degeneracies. If $p_t = p_v$, we order p_t and p_v such that p_v immediately follows p_t in clockwise order. If $p_v = r_t$, we order p_v and r_t such that r_t immediately follows p_v in clockwise order. (The first condition applies to r and s as well, and the second applies to all pairs taken from $\{p,r,s\}$.) See Figure 9 for an example of this notation. We want to show that if p_t, r_t, and s_t occur in clockwise order, so do p_v, r_v, and s_v.

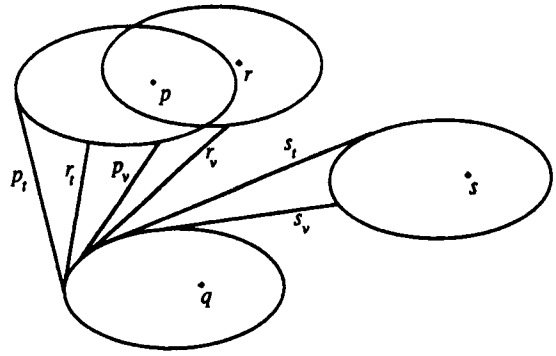
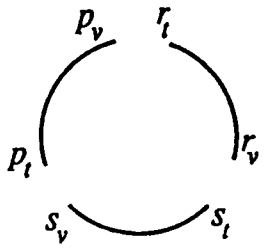


Figure 9. The tangent directions p_t, r_t, and s_t and the visibility directions p_v, r_v, and s_v. The cyclic permutation associated with these directions is $(p_t\, r_t\, p_v\, r_v\, s_t\, s_v)$.

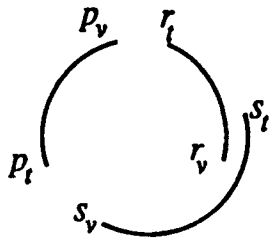
The following simple condition on the clockwise sequence of directions is stated using p and r, but it applies to all pairs drawn from $\{p,r,s\}$: At most one of r_t and r_v occurs after p_t and before p_v. If $(p_t\, r_t\, r_v\, p_v)$ is a subsequence of the cyclic sequence of angles, then $v_r(q,p)$ is not an interval, since it includes both p_t and p_v, but not r_v, which lies between them. This contradicts Lemma 4.2. If $(p_t\, r_v\, r_t\, p_v)$ is a subsequence of the cyclic sequence of angles, then at least one of $v_\emptyset(p)$ and $v_\emptyset(r)$ is larger than π, also a contradiction.

Of the possible angle sequences, the only ones that satisfy the condition also have p_v, r_v and s_v in order. There are 120 cyclic permutations on $\{p_t, p_v, r_t, r_v, s_t, s_v\}$. Of these, 60 have p_t, r_t, and s_t in clockwise order. Of the 60, only eleven meet the condition given above; in all of those, p_v, r_v, and s_v appear in clockwise order. If we class together permutations that are

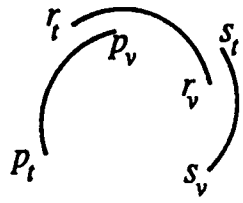
equivalent under renaming of p, r, and s, there are really only five different permutations that satisfy the condition. They are given in Figure 10. This proves that the tangent-visible points generated by p, r, and s occur in the same clockwise order as p, r, and s themselves. ∎



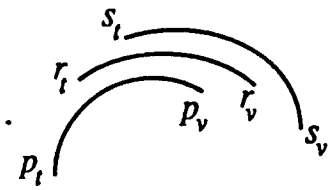
Class 1



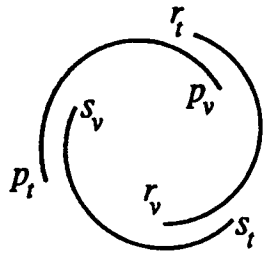
Class 2



Class 3



Class 4



Class 5

	Example	Equivalent permutations	
Class 1:	$(p_t\, p_v\, r_t\, r_v\, s_t\, s_v)$		
Class 2:	$(p_t\, p_v\, r_t\, s_t\, r_v\, s_v)$	$(p_t\, s_v\, p_v\, r_t\, r_v\, s_t)$	$(p_t\, r_t\, p_v\, r_v\, s_t\, s_v)$
Class 3:	$(p_t\, r_t\, p_v\, s_t\, r_v\, s_v)$	$(p_t\, s_v\, p_v\, r_t\, s_t\, r_v)$	$(p_t\, s_v\, r_t\, p_v\, r_v\, s_t)$
Class 4:	$(p_t\, r_t\, s_t\, p_v\, r_v\, s_v)$	$(p_t\, r_v\, s_v\, p_v\, r_t\, s_t)$	$(p_t\, r_t\, s_v\, p_v\, r_v\, s_t)$
Class 5:	$(p_t\, s_v\, r_t\, p_v\, s_t\, r_v)$		

Figure 10. Five essentially different cyclic permutations satisfy the condition of Lemma 4.3. The figure shows one example from each class, and the table lists its equivalent permutations, normalized to start with p_t.

Lemmas 4.1 and 4.3, taken together, have the following consequence:

Theorem 4.1. *The sequence $g(VS(q))$ is a subsequence of $VP(q)$.*

5. Construction of the Augmented Path Graph

Since any shortest path from α to ω follows edges of the augmented path graph, our first step in finding a shortest path is to construct that graph. The graph has two kinds of edges, boundary edges and tangent edges, which we find separately. (We need to find only the edges, since the nodes of the graph are given by edge intersections.)

To find the boundary edges, we construct the boundary of the barriers, that is, the border of the region where the robot's center may be placed. We compute this boundary using the divide-and-conquer method of Kedem et al. [KLPS], which runs in time $O(\tau n \log^2 n)$; here τ is a constant that depends on B and which will be defined later.

To find the tangent edges, we construct the two tangent-visible sequences from each fattened polygon vertex B^q. The sequence $VS(q)$ and its counterclockwise counterpart give all path graph edges tangent to B^q.

The results of the preceding section provide a way to produce the visibility sequence $VS(q)$ from the visibility polygon $VP(q)$, which we can find using the algorithms of Asano et al. [AAGHI] and Welzl [W]. These algorithms compute the visibility polygon in $O(n)$ time per vertex. (The algorithms require that the obstacle segments be disjoint, as does the boundary construction of Kedem et al. mentioned above.)

To bound the time it takes to construct $VS(q)$, we introduce some conditions on B. These conditions are not restrictions on the shape of B, but rather characterizations of it. Each of the five operations listed below is used in some phase of our shortest path algorithm; to bound the algorithm's performance, we need to bound the complexities of the basic operations. We therefore assume that each of the following operations can be performed in time τ, for some τ dependent on B:

To compute $VS(q)$ from $VP(q)$, we must be able to

(1) *Find intersection points of two translated copies of B,*

(2) *Find the intersection of a line with B, and*

(3) *Compute the inner and outer common tangents of two translated copies of B.*

To compute the boundary points visible from the source and destination points α and ω, we must be able to

(4) *Find the tangents to B through a point.*

To compute the lengths of path graph edges quickly, we need to

(5) *Compute the perimeter distance between two arbitrary points on an arc of B.*

(Note that preprocessing of B may be used to speed up distance computations.)

Because of Theorem 4.1, we are able to compute $VS(q)$ using an algorithm similar in spirit to the Graham scan convex hull algorithm [Gr]. Roughly speaking, we replace each vertex and segment in $VP(q)$ by the barrier it generates, then find the tangent-visible points on the union of these barriers. Each visibility polygon vertex p contributes B^p to the union, and each segment $\overline{pr}$ contributes a tube $\bigcup_{s \in \overline{pr}} B^s$, which we represent compactly by $B^{\overline{pr}}$. (See Figure 11.)

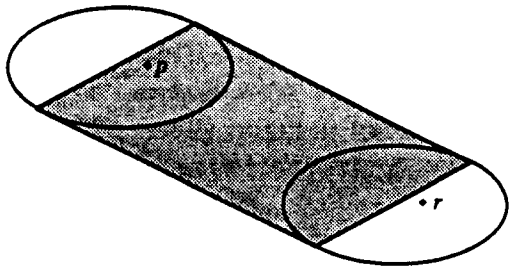


Figure 11. The tube $B^{\overline{pr}}$ and its internal parallelogram.

It is convenient to replace the tube $B^{\overline{pr}}$ by the parallelogram defined by the endpoints of the outer common tangents of B^p and B^r. This choice removes a possible degeneracy from the construction. Each arc is generated just once, by a vertex, rather than three times— once by the vertex and twice by the vertex's incident segments. Computing the parallelogram requires operation (3) above.

Each barrier generated by an element i in the visibility polygon $VP(q)$, either vertex or segment, has a range of angles in which it is tangent-visible from B^q if no other barriers are present. This range is $v_\emptyset(q,i)$ if i is a vertex; for convenience, we use the notation $v_\emptyset(q,i)$ to represent the range when i is a segment, too.

If B^i intersects B^q, the intersection blocks all tangent-visibilities in the angular range it covers. The range in which the barrier generated by i blocks all visibilities is called its *blocking interval* $b(i)$. If B^i and B^q do not intersect, $b(i)$ is empty. For clockwise-going tangents to B^q, the intersection of B^i and B^q is clockwise of the tangent-visible part of B^i. The blocking interval $b(i)$ immediately follows $v_\emptyset(q,i)$ in clockwise order.

When more than one barrier is present, each barrier B^i is visible in some subinterval of $v_\emptyset(q,p)$. An interval in which a barrier B^i is

tangent-visible may be followed in clockwise order by a blocking interval. If two blocking intervals $b(i)$ and $b(j)$ overlap, they can be merged into a single joint blocking interval. (See Figure 12.) We associate this joint interval with the tangent-visible barrier B^i that precedes it in clockwise order; note that the blocking interval $b(i)$ contributes to the joint blocking interval.

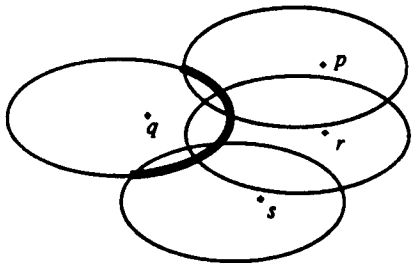


Figure 12. The blocking intervals of B^p, B^r and B^s are merged into a single joint blocking interval and associated with B^p.

Theorem 4.1 has an important algorithmic consequence. Let $L = (i_1, \ldots, i_k)$ be a clockwise subsequence of the visibility polygon $VP(q)$. Each element i in L has associated with it a blocking interval $b(i)$ that may be a joint interval (larger than the intersection of B^i with B^q). Suppose that when the barriers and blocking intervals given by the elements of L are present, every such barrier is tangent-visible from B^q. Let j be a successor of i_k in $VP(q)$; that is, $(i_1, \ldots, i_k, j)$ is a subsequence of $VP(q)$. Theorem 4.1 implies that if B^j is hidden from B^q by barriers and blocking intervals given by elements in L, it is hidden by the union of B^{i_1}, B^{i_k}, $b(i_1)$, and $b(i_k)$. On the other hand, if B^j hides some of the barriers generated by elements in L, it is visible itself. Furthermore, B^j and $b(j)$ hide a contiguous initial and final subsequence of L. That is, if they hide a part of B^x for some x in L, they also hide all the predecessors of x in L or all the successors.

The following algorithm exploits these observations to compute $VS(q)$. It maintains a double-ended linked list L of visibility polygon elements. Each element has associated with it an interval of current visibility $v(i)$ and a blocking interval $b(i)$. (As the algorithm runs, $v(i)$ shrinks and $b(i)$ grows.) The list corresponds to the subsequence L described above. At the top of each **for** loop, the following invariant holds: L contains a clockwise subsequence of $VP(q)$ such that when only the barriers and blocking intervals given by L are present, every such barrier is tangent-visible from B^q. (At the end of execution, L contains exactly the elements that generate $VS(q)$.) During each loop, a new element of the visibility polygon $VP(q)$ is inserted at the end

of L. The operations necessary to restore the invariant affect only the ends of the list.

```
Set L to be an empty double-ended queue. The functions head(L)
  and tail(L) return the elements at the ends of the queue.
for each i in VP(q) in clockwise order do
  begin
    Let b(i) be the interval given by the intersection of B^i and B^q
    while B^i and b(i) hide B^head(L) in the interval v(head(L)) do
      begin
        Merge b(head(L)) into b(i);
        Delete head(L) from L;
      end
    while B^i and b(i) hide B^tail(L) in the interval v(tail(L)) do
      begin
        Merge b(tail(L)) into b(i);
        Delete tail(L) from L;
      end
    Let U be the set of barriers and blocking intervals B^head(L),
      B^tail(L), b(head(L)), b(tail(L)), and b(i);
    (The following test uses the primitive operation twice:)
    if B^i is tangent-visible from B^q in the presence of U then
      begin
        Determine the endpoints of v(i) by comparisons with U;
        Adjust the endpoints of v(head(L)) and v(tail(L)) if B^i and
          b(i) affect them;
        Insert i at the tail of L;
        (The barriers generated by i, head(L), and tail(L) are all
          visible and separate their associated blocking intervals, so
          no blocking intervals need to be merged.)
      end
    else (B^i is not tangent-visible)
      Merge b(i) into b(tail(L));
  end
```

The algorithm uses just one geometric primitive: Given two elements of the visibility polygon i and j and their associated blocking intervals $b(i)$ and $b(j)$, the primitive determines all tangent visibilities that exist when only the barriers B^i and B^j and the blocking intervals $b(i)$ and $b(j)$ are present. If the barrier B^i is not tangent-visible under these circumstances, we say that $b(j)$ and B^j *hide* B^i. The primitive requires a constant number of the tangent- and intersection-finding operations (1)–(4), and therefore takes $O(\tau)$ time. (Operation (4) is needed

because we replaced B^i by a parallelogram for each segment i in the visibility polygon.)

This algorithm runs in $O(\tau n)$ time. Each execution of the outer **for** loop takes $O(\tau)$ time, exclusive of the time spent in the two inner **while** loops. Each time one of those loops is executed, an element of L is deleted. No more than n elements are ever added to L, so the inner loops are executed at most n times altogether; each execution takes $O(\tau)$ time.

The preceding discussion and algorithm constitute a proof of the following lemma.

Lemma 5.1. *If each of the operations (1), (2), (3), and (4) given above can be performed in time τ, then the tangent-visible sequence $VS(q)$ can be produced from the visibility polygon $VP(q)$ in $O(\tau n)$ time.*

When the algorithm terminates, L contains $g(VS(q))$, the generators of the barriers clockwise tangent-visible from B^q. It is easy to find the path graph edges clockwise-tangent to B^q given the list L. For each element $i \in L$, if either end of the visible range $v(i)$ is delimited by a common tangent of B^q and B^i, then we add the tangent edge to the path graph.

We can construct the tangent edges of the path graph, but we still need to find the edges from α and ω that augment the path graph. However, the ideas used in Section 4 and the preceding lemma apply to this problem as well.

Lemma 5.2. *The edges from α and ω that augment the path graph can be found in $O(\tau n)$ time.*

Proof: Lemmas 4.1 through 4.3 discuss tangent-visibility of barriers. Similar lemmas are true for visibility of barriers from a point. For example, if r lies on the segment connecting α and p, no point of B^p is visible from α, which proves a lemma analogous to Lemma 4.1. Let $\tilde{v}_T(q,p)$ have the same meaning for visibility from q as $v_T(q,p)$ has for tangent-visibility from B^q. Then $\tilde{v}_T(q,p)$ is an interval for any T: as in the proof of Lemma 4.2, we consider $\tilde{v}_r(q,p)$ for any r and show that B^r cannot intersect $\overline{qv_b}$ in Figure 13 without touching $\overline{qv_a}$ or $\overline{qv_c}$.

The analogues of Lemma 4.3 and Theorem 4.1 follow directly from the modified Lemmas 4.1 and 4.2, and the Graham scan of Lemma 5.1 is easily modified to handle visibility from a point, given that B satisfies condition (4) given above. ∎

This section has described three separate aspects of constructing the path graph. The method of Kedem et al. finds the boundary of the barriers in $O(n\tau \log^2 n)$ time. The algorithm given above produces the path graph tangent edges in $O(n^2\tau)$ time. A variant on the algorithm,

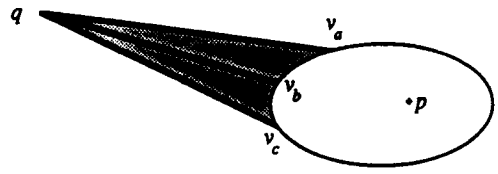


Figure 13. Proof that $\tilde{v}_T(q,p)$ is an interval; see Figure 8. B^r cannot intersect $\overline{qv_b}$ (block visibility in direction b) without also intersecting either $\overline{qv_a}$ or $\overline{qv_c}$.

sketched in the previous lemma, finds the edges from α and ω that augment the path graph in $O(n\tau)$ time. The endpoints of these edges, taken together with the barrier boundaries, give all the boundary edges of the augmented path graph. The following theorem summarizes these results:

Theorem 5.1. *The augmented path graph G_a can be constructed in $O(\tau n^2)$ time.*

6. Pruning the Path Graph

It is possible to find a shortest path by running Dijkstra's algorithm on the augmented path graph, G_a. However, because the tangent edges in G_a can have $\Omega(n^2)$ endpoints altogether, Dijkstra's algorithm may take $\Omega(n^2 \log n)$ time. To reduce the time to $O(n^2)$, we need a graph with fewer nodes. This section and the following section show how to produce a graph equivalent to G_a that has fewer nodes. This section deletes unusable tangent edges and their endpoints; the next section groups nodes together and modifies edges to link the groups.

This section shows how to eliminate some edges of G_a that lie on no shortest path between α and ω; we identify these edges using only local tests. After deleting the edges from G_a, we delete any resulting nodes of degree two and merge their incident edges. The result is a *pruned path graph*, G_p.

We have already noted that in the path graph there are both clockwise and counterclockwise tangent edges from a barrier arc. There are inner and outer common tangent edges of both types. Altogether, the four possible combinations of clockwise/counterclockwise with inner/outer give four classes of path graph edges from each barrier arc. Within each class all tangents to an arc are cyclically ordered. The following lemma shows how to identify unusable edges of each class.

Lemma 6.1. *Suppose e_1 and e_2 in G_a are consecutive clockwise outer tangents going from B^q to barriers B^{p_1} and B^{p_2} with p_2 to the right of the directed line from q to p_1. If B^{p_1} and B^{p_2} intersect, the continuation of e_2 intersects B^{p_1}, thus defining a bay. (See Figure 14.) Then edge e_2 cannot appear on a shortest path unless α or ω lies in the bay.*

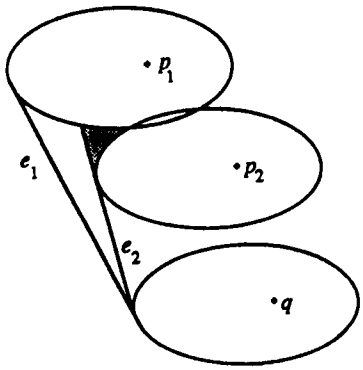


Figure 14. Because B^{p_1} and B^{p_2} intersect, edge e_2 cannot appear on a shortest path unless α or ω lies in the shaded bay.

Proof: The continuation of e_2 must intersect B^{p_1}; otherwise it would separate B^{p_1} and B^{p_2}. If neither α nor ω lies in the bay, any path that enters the bay can be shortened by re-routing it to avoid the bay. Any path that uses e_2 has no forward continuation (wrapping around the corner), because all such continuations enter the bay. ∎

Similar conclusions hold for pairs of consecutive edges from the other three classes: clockwise inner tangents, counterclockwise outer tangents, and counterclockwise inner tangents.

The preceding lemma gives a way of pruning the augmented path graph to remove unneeded edges. The following lemma shows that the pruning can be performed in $O(\tau n^2)$ time overall.

Lemma 6.2. *All prunable edges from arcs generated by a polygon vertex q can be found and discarded in $O(\tau n)$ time.*

Proof: Each pruning test takes only $O(\tau)$ time. Suppose that e_1 and e_2 are consecutive edges of the same class tangent to B^{p_1} and B^{p_2}. If B^{p_1} and B^{p_2} intersect, we can find the points a, b, and c shown in Figure 15 using $O(\tau)$ time. Since α and ω lie outside the barriers, α or ω lies in the shaded bay if and only if it lies in the triangle $\triangle abc$, which can be checked in constant time. If neither α nor ω lies in $\triangle abc$, then edge e_2 may be removed.

To check whether an edge needs pruning, we need only test it against edges of the same class whose endpoints precede or follow it on the same arc. When an edge is pruned, two edges that were not neighbors become neighbors and must be tested. However, because there are $O(n)$ path graph edges of each class from arcs generated by q, only $O(n)$ tests are needed for each polygon vertex q. ∎

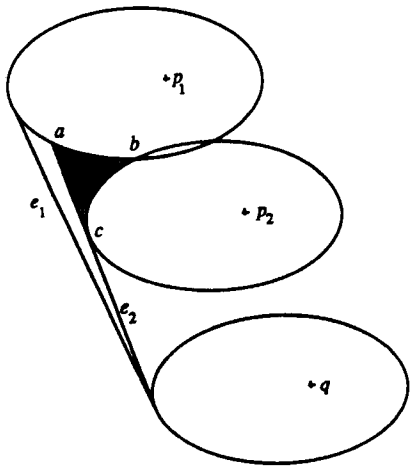


Figure 15. Point α or ω is in the bay if and only if it is in $\triangle abc$. If neither α nor ω lies in $\triangle abc$, then e_2 can be pruned.

If two successive edges of the same class go to overlapping translated copies of B and neither α nor ω lies in the bay reached by the inner edge (e_2 in Figure 15), the inner edge is pruned. In cases like the one shown in Figure 16, many unnecessary edges can be pruned. After pruning, at most two pairs of successive edges go to overlapping barriers. (Each unpruned inner edge defines a bay containing α or ω.) The following lemma, which is used in the next section, shows that if barriers reached by successive edges of the same class do not overlap, neither do barriers reached by non-successive edges.

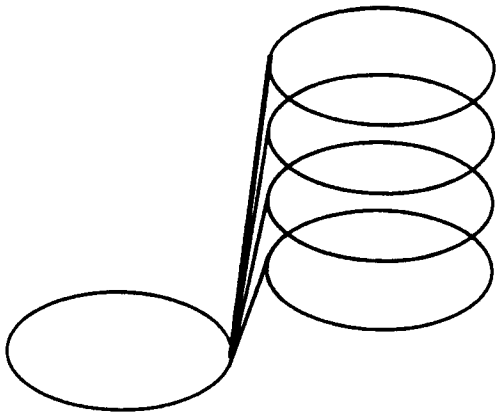


Figure 16. In this example, pruning removes many useless edges. Of the edges shown, only the leftmost can appear on a shortest path, since neither α nor ω lies in any of the bays formed by the other edges.

Lemma 6.3. *Let E be a subset of the edges in G_p that connect B^q to arcs of other barriers, chosen so that all edges in E have the same class. Let U be the set of polygon vertices that generate the arcs reached by edges of E. The angles of the edges in E give a cyclic ordering on the elements of U. If $B^p \cap B^s$ is empty for all consecutive generators p and s, then $B^p \cap B^s$ is empty for all distinct p and s in U.*

Proof: Suppose to the contrary that $B^p \cap B^s$ is non-empty for some p and s in U. Choose p and s such that p precedes s and the size of the subset of U between p and s is minimized. By hypothesis this subset contains some element r. Since the common tangent from B^q to B^r reaches a visible point, the point of tangency is in the shaded region (see Figure 17). This implies that B^r extends beyond (in Figure 17, to the right of) any line passing through the intersection of B^p and B^s and tangent to B^q. To get out of the shaded region without overlapping B^p or B^s, B^r must cross the tangent from B^q to B^s, and hence that edge cannot be in the path graph, a contradiction. ∎

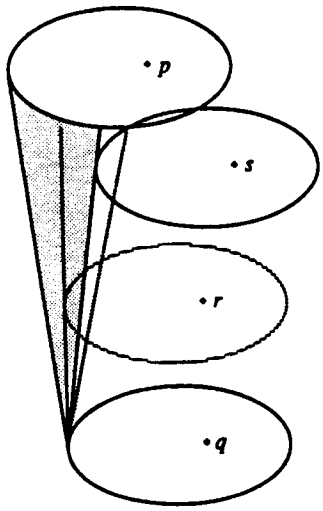


Figure 17. Neighbor non-intersection implies global non-intersection. The tangents from B^q to B^p, B^r, and B^s occur in clockwise order. If B^p and B^s intersect, but B^r intersects neither of them, then B^r blocks the tangent to B^s.

7. Node Coalescing in the Pruned Path Graph

We have shown that any shortest path from α to ω follows edges of the pruned path graph, G_p. Because G_p may still have $\Omega(n^2)$ nodes, Dijkstra's algorithm may require $\Omega(n^2 \log n)$ time when applied to it. In this section, we show how to modify the graph so that Dijkstra's algorithm takes only $O(n^2)$ time when applied to the result. Because nodes of G_p are coalesced during the modification, we call the resulting graph the *coalesced graph* G_c.

The modifications of this section group consecutive nodes on the boundaries of barriers. To describe the modifications better, we first characterize barrier boundaries. Every boundary between a maximal connected barrier region and the barrier-free region is a ring of path graph nodes and edges. (A barrier may have several such rings if it has holes.) Any subpath of a shortest path that uses edges from one of these rings follows them in a consistent direction, either clockwise or counterclockwise. A ring is entered and left via tangents to it, and these connections are consistent with the orientation of the ring. See Figure 18. Any path in G_a that traverses rings in a single direction and connects to them consistently is called *forward-going*.

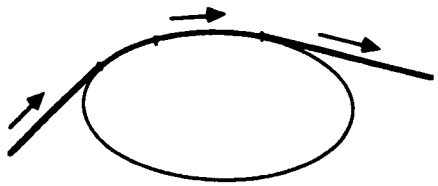


Figure 18. A forward-going subpath.

An edge of the pruned path graph can be traversed in either direction, since the path graph is undirected. As the first step in forming the coalesced graph, we transform G_p into a directed graph, replacing each undirected barrier boundary ring by two oppositely directed copies of itself. Each tangent edge is replaced by two oppositely directed copies of itself, and each copy is connected to the ring consistent with its direction. The procedure of duplication and connection is depicted in Figure 19. Path graph edges from α and ω are replaced by directed edges (outgoing from α, incoming to ω) whose other endpoints are connected to the appropriate directed rings. In this directed graph, which we call G_d, only forward motion is possible. Every shortest path from α to ω is present in G_d.

To help define the nodes of the coalesced graph, we introduce *distinguished nodes*, which form a subset of the nodes of G_d. (We show how to select this subset later.) The distinguished nodes have three

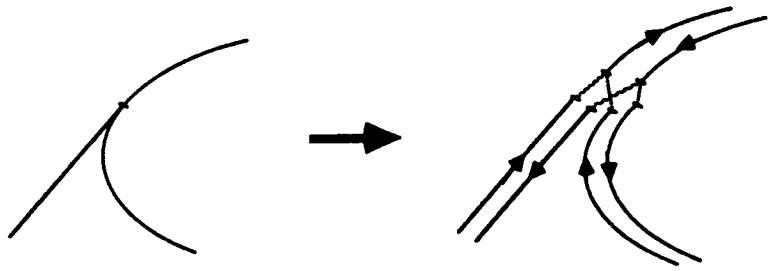


Figure 19. Construction of G_d. Each path graph edge is replaced by two oppositely directed edges; the edges are linked up to allow only forward-going paths.

properties that we use presently. First, all arc endpoints are distinguished nodes. Second, the tangents to consecutive distinguished nodes on a single arc differ in direction by at most $\pi/2$. Third, every tangent edge that ends between two consecutive distinguished nodes on a single arc is at least as long as the interval between the distinguished nodes, measured along the barrier boundary.

The distinguished nodes break the barrier boundary rings into intervals. (Note that each point on the boundary of a barrier belongs to two intervals: one for each direction of traversal.) Somewhat unconventionally, we view the distinguished nodes as single-point closed intervals distinct from the open intervals they separate. The set of all such intervals (both open and closed) forms the node set of the coalesced graph G_c.

The edges of the coalesced graph connect the intervals defined by the distinguished nodes. An edge linking two intervals is assigned a weight (roughly equivalent to length) as if it went from the forward point of one interval (as defined by the ring orientation) to the forward point of the other.

We first assign weights to tangent edges. Each tangent edge of G_d links intervals (either open or closed) delimited by distinguished nodes. In Figure 20, the edge from a to b links the intervals (a', a'') and (b', b''). We assign the corresponding edge in G_c the weight $|ab| - |a''a| + |bb''|$, where distances $|a''a|$ and $|bb''|$ are measured along the barrier boundary. This is the distance registered on the odometer of a hypothetical car that backs up from a'' to a along the arc, goes forward along $\overline{ab}$, and then advances from b to b'' along that arc. (The fifth condition on B means that the length computation takes $O(\tau)$ time.) Since there is at most one edge of G_d from one arc to any other, there is at most one tangent edge connecting any pair of intervals. Because $|a'a''|$ and $|b'b''|$ are no greater than $|ab|$, the edge linking the intervals of a and b in the coalesced graph has non-negative weight.

We now assign weights to boundary edges. The distinguished nodes

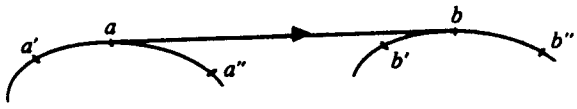


Figure 20. The edge of G_c corresponding to the directed edge from a to b is assigned weight $|ab| - |a''a| + |bb''|$.

break each barrier boundary into a ring of intervals. These rings are a coarsening of the rings of boundary edges in G_d. From a single-point interval $[a', a']$ to its open successor (a', a'') we create an edge with weight $|a'a''|$. An edge of weight 0 goes from the open interval (a', a'') to its closed successor $[a'', a'']$. Note that each edge joining adjacent intervals has weight equal to the distance between the forward points of the intervals.

If we are to use the coalesced graph to find shortest paths, we must show that shortest paths in G_c correspond to shortest paths in the augmented path graph.

Lemma 7.1. *Any minimum weight path from α to ω through the coalesced graph G_c corresponds to a shortest path in the augmented path graph of the same length.*

Proof: The proof has two parts. We first show that every path in G_d maps to a path in the coalesced graph G_c with the same length. Next we show that every path in G_c that does not correspond to a forward-going path has weight greater than the length of the shortest forward-going path.

A shortest path through G_d is also a shortest path in the augmented path graph, since a shortest path is a forward-going path. When the nodes of G_d are coalesced into the intervals of G_c, the lengths of forward-going paths are preserved. The coalesced edge weights are calculated so that when a path travels forward along a barrier boundary, the perimeter length is accounted correctly. For example, in Figure 21, the forward-going path from s to f has length $|sa| + |ab| + |bf|$. In G_c the path has two edges, one from s to the interval (a', a'') and one from the interval to f. These edges have weights $|sa| + |aa''|$ and $|bf| - |a''b|$, which sum to the correct value.

Grouping edge endpoints into intervals allows some paths in G_c that are not derived from forward-going paths. These are subpaths that enter an interval, back up, and then leave, as in Figure 22. The edge weights of such a subpath in G_c add up to less than the Euclidean length of the path. However, we show that the total weight of a path in G_c from α to ω that includes such a subpath is still greater than the length of the

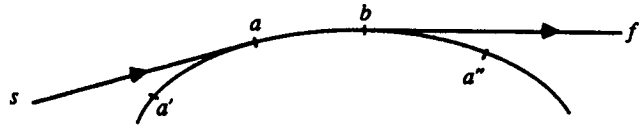


Figure 21. Forward-going paths are charged accurately in the coalesced graph. The path from s to f has weight $(|sa| + |aa''|) + (|bf| - |a''b|)$, which is equal to $|sa| + |ab| + |bf|$, the Euclidean length of the path.

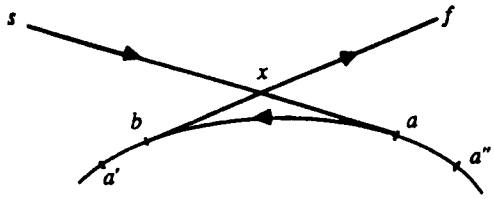


Figure 22. Loops are undercharged, but not too much. The path $s \rightarrow a \rightarrow b \rightarrow f$ has weight $|sa| + |bf| - |ab|$, which is less than its length, but more than the length of the path $s \rightarrow x \rightarrow f$. This latter path is longer than the shortest path from s to f, so the loop cannot be mistaken for a shortest path.

shortest path in G_d.

The subpath in Figure 22 has weight $(|sa|+|aa''|)+(|bf|-|a''b|) = |sa| + |bf| - |ab|$, which is less than $|sa| + |ab| + |bf|$, the actual length of the subpath. However, because $|sa|$ and $|bf|$ are both greater than $|a'a''|$ and the difference between the tangent directions at b and a is at most $\pi/2$ ($\angle sxf \geq \pi/2$), the edges $\overline{sa}$ and $\overline{bf}$ intersect at x. Since $|ab|$ is measured along a convex barrier boundary, $|ax| + |xb| > |ab|$. Though the weight of the subpath is less than its actual length, it is still more than $|sx| + |xf|$, the length of the subpath that follows $\overline{sx}$ to x and then follows $\overline{xf}$. Because of the open corner at x, this second subpath cannot be part of any shortest path.

The preceding analysis assumes that the loop $x \rightarrow a \rightarrow b \rightarrow x$ is isolated from any other such loop. We now show that the assumption is valid. If an edge $\overline{ba}$ connects two loops as in Figure 23(a), the two loops are independent: the subpath containing them has weight at least $|sc| + |cd| + |df|$ in G_c, the length of an easily identified subpath. If intersections c

and d appeared in opposite order, the two loops would not be independent: the lower bound would be $|sc| + |df| - |cd|$, which corresponds to no recognizable subpath. However, because $\angle sca$ and $\angle fdb$ are at least $\pi/2$, the interlocking loops of Figure 23(b) are impossible.

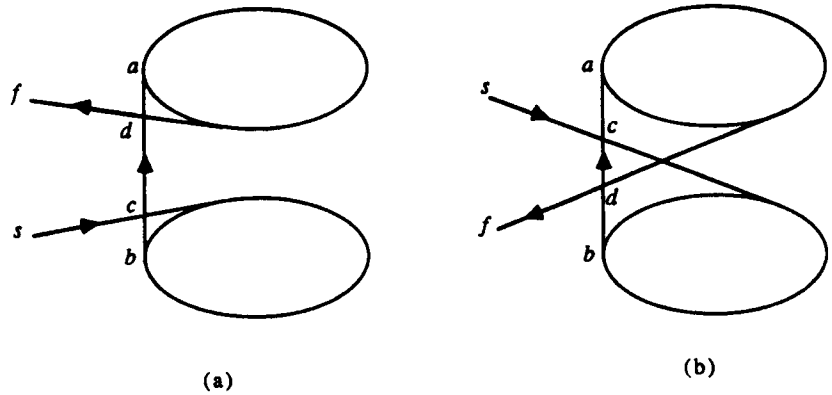


Figure 23. Two loops connected by a single edge; (a) possible and (b) impossible configurations. Because angles $\angle sca$ and $\angle fdb$ are at least $\pi/2$, the points a, d, c, and b occur in that order along $\overline{ab}$.

Any path from α to ω in G_c that includes instances of doubling back, as in Figure 22, has weight greater than the length of the shortest path in G_d. Consider replacing each instance of doubling back by the shortcut corresponding to the subpath in Figure 22 that follows $\overline{sx}$ to x and then follows $\overline{xf}$. The length of the resulting barrier-avoiding path is less than the weight of the path through G_c from which it is derived. The resulting path is not a shortest path from α to ω, since short-cutting the corner corresponding to $\angle sxf$ gives a shorter path. This means that no path through G_c that is not forward-going can be mistaken for a shortest path. ▮

The proof of the preceding lemma uses the three properties of distinguished nodes mentioned earlier. We now show how to select the distinguished nodes so that these properties hold. (The properties, once again, are the following: all arc endpoints are distinguished nodes, the tangent directions at consecutive distinguished nodes differ by at most $\pi/2$, and every tangent edge is at least as long as the intervals containing its endpoints.) There are a number of ways of selecting distinguished nodes; the one that we adopt here is perhaps the simplest nontrivial method.

Our approach characterizes the robot with two parameters. We take l to be the *diameter* of B, that is, the maximum distance between

two points of B. This is also equal to the maximum separation of two parallel supporting lines of B. The minimum separation w of two such lines is called the *width* of B.

The first distinguished nodes we choose are those path graph nodes that are intersections of arcs or boundary segments with each other, including the junctions between arcs and segments at their endpoints. Kedem et al. [KLPS] show that there are only $O(n)$ such intersections. We add distinguished nodes to each arc so that no interval on an arc is longer than l. We then add distinguished nodes to guarantee that tangent directions at successive distinguished nodes differ by at most $\pi/2$. (Note that these two conditions on intervals need only be met if the interval contains an endpoint of a tangent edge. If an interval is too long or turns by more than $\pi/2$, but has no tangent endpoints in it, there is no need to break it in two with a distinguished node. Therefore all distinguished nodes can be taken to be path graph nodes.) So far, we have chosen $O(n)$ distinguished nodes. Now we add to the set of distinguished nodes the endpoints of all tangent edges shorter than l. This guarantees the non-negativity of all edge weights in G_c. The following lemma bounds the number of distinguished nodes added in the final step.

Lemma 7.2. *Let w be the width and l the diameter of B. The pruned path graph G_p has $O(nl/w)$ edges shorter than l.*

Proof:

The proof concentrates on the barriers connected to B^q by tangent edges. It uses Lemma 6.3 to divide the barriers into four sets, each of them free of overlaps. For each set, the subset connected to B^q by edges shorter than l lies in a relatively small region close to B^q. Area arguments show that there are only $O(l/w)$ barriers in each such subset.

As in Lemma 6.3, let the set U contain the generators of the arcs reached from B^q by tangent edges of a single class in the pruned graph G_p. The elements of U are cyclically ordered by the angles of the edges. At most two pairs of consecutive vertices in U generate overlapping barriers. If we remove from U any vertex (there are at most two) whose corresponding edge leads to a bay containing α or ω, Lemma 6.3 implies that the barriers generated by the remaining vertices are pairwise disjoint.

Let $\mathcal{R}$ be the region consisting of all points closer than $2l$ to B^q. The area of $\mathcal{R}$ is $O(l^2)$. Every vertex in U corresponding to a tangent edge shorter than l generates a barrier that is entirely contained in $\mathcal{R}$. (See Figure 24.) These barriers are non-overlapping, and each has area $\Omega(lw)$; hence there are at most $O(l/w)$ of them. This argument applies to all four

tangent edge classes, and so for each polygon vertex q, there are $O(l/w)$ edges in G_p tangent to B^q and shorter than l. Summing over all vertices proves the lemma. ▌

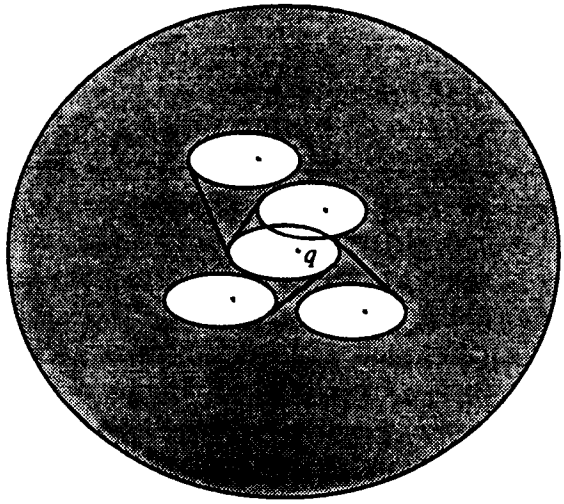


Figure 24. The barriers reached from B^q by tangent edges of a single class in the pruned path graph are non-overlapping (with at most two exceptions). Those connected to B^q by edges shorter than l lie completely inside the region closer than $2l$ to B^q (the shaded region). This region has area $O(l^2)$, and each barrier has area $\Omega(lw)$, so the number of short edges is $O(l/w)$.

Our algorithm finds shortest paths by producing a graph on which Dijkstra's single-source shortest path algorithm will run quickly, then invoking that algorithm. Sections 5 through 7 show how to construct the coalesced graph G_c in $O(rn^2)$ time. Dijkstra's algorithm takes $O(n^2 + \min(n, l/w)n \log n)$ to find the shortest path in G_c, which corresponds directly to the shortest path in the original setting. (The min function is present because there are at most $O(n^2)$ tangent edges shorter than l in the path graph.) Overall, our algorithm uses time $O(rn^2 + \min(n, l/w)n \log n)$ and space $O(n^2)$. If l/w is $O(n/\log n)$, then the algorithm takes $O(rn^2)$ time. If B is independent of n, as in the important special case when B is a disk, the algorithm finds shortest paths in $O(n^2)$ time. This concludes the proof of the following theorem:

Theorem 7.1. *Given a non-rotating convex body A, a set S of simple polygons with a total of n vertices, and two points α and ω, it is possible*

to find a shortest polygon-avoiding path for A from α to ω in $O(n^2)$ time, where the implied constant depends on the complexity of A.

8. Conclusions and Open Questions

In this paper we have presented an algorithm to find shortest paths for a non-rotating convex body moving among polygonal obstacles. We first reduced the problem to the equivalent problem of moving a point among "fattened" obstacles [LPW], [GRS]. We then introduced the concept of tangent-visibility to describe the sequence of barrier boundary points swept by a tangent ray continuously rotating around a placement of B, the inverted robot. This concept allowed us to compute efficiently the path graph of the barriers, which contains the edges of all shortest paths. Unfortunately the path graph may contain $\Omega(n^2)$ nodes, so that Dijkstra's shortest path algorithm runs in superlinear time when applied to it. By pruning certain path graph edges and then coalescing the remaining nodes into distinguished nodes, we were able to reduce the total number of nodes to $O(n)$ and thus get a shortest path algorithm that runs in $O(n^2)$ time.

Perhaps the least satisfying feature of this algorithm is its dependence on the aspect ratio l/w of the object to be moved. According to our bounds, the algorithm could require $\Omega(n^2 \log n)$ time to find shortest paths for an ellipse whose aspect ratio is a linear function of n. This is disturbing, since in the limit such an ellipse becomes a segment, for which shortest paths are easy to find.

Our algorithm uses $O(nl/w)$ distinguished nodes in the construction of G_c. However, this upper bound on the number of necessary distinguished nodes is very crude. If we omit inner common tangents from the path graph, we can show that only $O(n \log(l/w))$ distinguished nodes are needed to construct G_c. We have not yet been able to extend this bound to allow inner common tangents, though we believe such an extension is possible.

Both upper bounds mentioned are local bounds: we prove that no fattened polygon vertex needs to have more than $O(l/w)$ or $O(\log(l/w))$ distinguished vertices on its boundary. Such bounds fail to consider the interactions of multiple barriers. It seems quite possible that a global argument could bound the number of necessary distinguished nodes by $O(n^2/\log n)$ or something even smaller.

The gap between upper and lower bounds is large; we can construct an example using only outer common tangents in which $\Omega(\sqrt{\log(l/w)})$ distinguished nodes are needed on a particular fattened vertex, but we have no super-linear global lower bounds. Improvements to either our lower or upper bounds would be welcome.

The algorithm we have presented does not process S before α and ω are given. However, it might be possible to construct approximations

to the whole progression of graphs from G to G_c before α and ω are given, then modify them in $O(\tau n)$ time once the source and destination are known. The most difficult preprocessing task would be pruning edges and simultaneously building a structure to identify the pruned edges that must be added back once α and ω are known. This is an interesting problem, but it is probably not worth working out the details so long as the final step of the algorithm has to run Dijkstra's algorithm, which may take $\Omega(n^2)$ time.

An alternative approach to shortest paths in the presence of polygons is taken by Reif and Storer [RS]. They avoid the Dijkstra step in their $O(n(k+\log n))$ shortest path algorithm for a point moving among polygons (k is the number of obstacle polygons). Their method factors the shortest path problem into two parts. First they use $O(n(k+\log n))$ time to compute a triangulation based on the source point α. Then the length of the shortest path to any point ω can be found in $O(\log n)$ time using an optimal point location method. The path itself can be found in additional time proportional to the number of turns along it. The Reif and Storer algorithm can be extended to plan the motion of a disk in the same time bound.

Acknowledgements

The authors would like to thank Jim Saxe, whose perceptive comments helped clarify the technical content of this report, and Cynthia Hibbard, whose advice greatly improved the presentation.

References

[AAGHI] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai: Visibility-polygon search and Euclidean shortest paths. *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, Portland, 1985, pp. 155–164.

[AHU] A. Aho, J. Hopcroft, and J. Ullman: *The design and analysis of computer algorithms.* Addison-Wesley, 1974, pp. 207–209.

[Ba] B. Baker: Shortest paths with unit clearance among polygonal obstacles. *SIAM Conference on Geometric Modeling and Robotics*, Albany, NY, July 15–19, 1985.

[Ch] L. P. Chew: Planning the shortest path for a disc in $O(n^2 \log n)$ time. *Proceedings of the ACM Symposium on Computational Geometry*, Baltimore, 1985, pp. 214–220.

[FT] M. Fredman and R. Tarjan: Fibonacci heaps and their uses in improved network optimization algorithms. *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, Singer Island, Florida, 1984, pp. 338–346.

[Gr] R. Graham: An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, Vol. 1 (1972), pp. 132–133.

[GRS] L. Guibas, L. Ramshaw, and J. Stolfi: A kinetic framework for computational geometry. *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, Tucson, 1983, pp. 100–111.

[KLPS] K. Kedem, R. Livne, J. Pach, and M. Sharir: On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete and Computational Geometry*, Vol. 1, No. 1 (1986), pp. 59–71.

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

[RS] J. Reif and J. Storer: Shortest paths in Euclidean space with polyhedral obstacles. Technical Report CS-85-121, Computer Science Department, Brandeis University, 1985. Submitted to *Journal of the ACM.*

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

Index

SRC Reports

"A Kernel Language for Modules and Abstract Data Types."
R. Burstall and B. Lampson.
Report #1, September 1, 1984.

"Optimal Point Location in a Monotone Subdivision."
Herbert Edelsbrunner, Leo J. Guibas, and Jorge Stolfi.
Report #2, October 25, 1984.

"On Extending Modula-2 for Building Large, Integrated Systems."
Paul Rovner, Roy Levin, John Wick.
Report #3, January 11, 1985.

"Eliminating go to's while Preserving Program Structure."
Lyle Ramshaw.
Report #4, July 15, 1985.

"Larch in Five Easy Pieces."
J. V. Guttag, J. J. Horning, and J. M. Wing.
Report #5, July 24, 1985.

"A Caching File System for a Programmer's Workstation."
Michael D. Schroeder, David K. Gifford, and Roger M. Needham.
Report #6, October 19, 1985.

"A Fast Mutual Exclusion Algorithm."
Leslie Lamport.
Report #7, November 14, 1985.

"On Interprocess Communication."
Leslie Lamport.
Report #8, December 25, 1985.

"Topologically Sweeping an Arrangement."
Herbert Edelsbrunner and Leonidas J. Guibas.
Report #9, April 1, 1986.

"A Polymorphic λ-calculus with Type:Type."
Luca Cardelli.
Report #10, May 1, 1986.

"Control Predicates Are Better Than Dummy Variables For Reasoning About Program Control."
Leslie Lamport.
Report #11, May 5, 1986.

"Fractional Cascading."
Bernard Chazelle and Leonidas J. Guibas.
Report #12, June 23, 1986.

"Retiming Synchronous Circuitry."
Charles E. Leiserson and James B. Saxe.
Report #13, August 20, 1986.

by John Hershberger and Leonidas J. Guibas

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301