A Near-Optimal Algorithm for Shortest Paths Among Curved Obstacles in the Plane

John Hershberger
Mentor Graphics Corp.
Wilsonville, OR, USA
john_hershberger@mentor.com

Subhash Suri
Dept. of Computer Science
UC Santa Barbara
Santa Barbara, CA, USA
suri@cs.ucsb.edu

Hakan Yıldız
Dept. of Computer Science
UC Santa Barbara
Santa Barbara, CA, USA
hakan@cs.ucsb.edu

ABSTRACT

We propose an algorithm for the problem of computing shortest paths among curved obstacles in the plane. If the obstacles have $O(n)$ description complexity, then the algorithm runs in $O(n \log n)$ time plus a term dependent on the properties of the boundary arcs. Specifically, if the arcs allow a certain kind of *bisector intersection* to be computed in constant time, or even in $O(\log n)$ time, then the running time of the overall algorithm is $O(n \log n)$. If the arcs support only constant-time tangent, intersection, and length queries, as is customarily assumed, then the algorithm computes an *approximate* shortest path, with relative error ε, in time $O(n \log n + n \log \frac{1}{\varepsilon})$. In fact, the algorithm computes an approximate shortest path map, a data structure with $O(n \log n)$ size, that allows it to report the (approximate) length of a shortest path from a fixed source point to any query point in the plane in $O(\log n)$ time.

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms: Algorithms, Theory

Keywords: Continuous Dijkstra, curved obstacles, polygonization, shortest paths

1 INTRODUCTION

The problem of computing shortest paths in the presence of obstacles has a long history in computational geometry, dating back to the late 1970s [11, 18, 27]. The problem draws inspiration from the fact that a Euclidean space cluttered with a finite number of geometrically-defined obstacles offers a clean and tractable abstraction for path-planning applications in robotics, industrial automation, model-based planning, and geographical information systems; see [21] for a comprehensive survey. The problem of computing a minimum length path avoiding a set of polygonal obstacles in the two-dimensional plane, in particular, has drawn great interest. After a sequence of steady improvements using a variety of techniques [10, 17, 19, 20, 28, 29], the worst-case complexity of the problem was settled by Hershberger and Suri [16] when they discovered an $O(n \log n)$-time algorithm based on the continuous Dijkstra framework. Many other variants of the problem, including shortest paths on a polyhedral surface [22], on convex polytopes [6, 12, 15], and in 3-space cluttered with polyhedral obstacles [23, 26], have been, and remain, active topics of research. In most cases, the algorithmic complexity of shortest paths, either exact or approximate, is now well-understood.

On the contrary, we do not seem to have a good grasp on the complexity of shortest paths when the obstacle boundaries are nonlinear *curves* [4,7]. Obstacles with curved boundaries present both *algebraic* and *combinatorial* challenges. In [2], for instance, Chang et al. show that even for n disjoint disc-shaped obstacles, computing a shortest path involves high bit-complexity algebra, and offer a singly-exponential algorithm in the exact geometric computation model. In this paper, we focus on the *combinatorial* complexity of the shortest path problem among curved obstacles, and adopt the usual custom that the algorithm has access to a small set of relevant *primitive operations* at constant time apiece. More specifically, to give our algorithmic framework broadest applicability, we assume a *black box* model of curved obstacles. Each obstacle boundary is given as a sequence of curved arcs, where each arc is monotone with respect to both the x- and y-axes and has no inflection points; that is, each arc is concave, convex or straight. The algorithm accesses the obstacles through a *primitive oracle* that can perform certain basic geometric computations such as finding a common tangent between two obstacles, finding the distance between two points along a curved boundary, and intersecting an obstacle boundary with a line. These primitives are easy to compute in constant time when the obstacle boundaries are defined by simple shapes such as circles and ellipses.

This black box model, which is implicitly used in most previous work on curved obstacle shortest paths, allows us to focus more cleanly on the following fundamental combinatorial bottleneck. *In the worst case, n curved obstacles have $\Omega(n^2)$ pairwise common tangents with $\Omega(n^2)$ distinct points of tangency.* Although a shortest path will use only a linear number of these tangencies as intermediate vertices,

SoCG'13, June 17–20, 2013, Rio de Janeiro, Brazil.

it is not clear how to extract that subset in less than $O(n^2)$ time. (By contrast, polygonal obstacles have only n such points of tangency, even if there are $\Theta(n^2)$ tangents, where n is the total number of vertices in all the obstacles.) This quadratic complexity has remained a major stumbling block for all shortest path algorithms that utilize a *visibility graph* approach, and indeed no subquadratic algorithm is known for shortest paths among curved obstacles, because all exact algorithms follow the visibility graph approach [4, 5, 13, 24].

Because the visibility graph approach seems inherently limited by the quadratic complexity of the graph, we are led to investigate the alternative *continuous Dijkstra* paradigm, based on simulating an expanding wavefront. That paradigm, however, trades one difficulty for another: it requires us to operate on *bisectors* defined by pairs of curved obstacle boundary segments. In particular, a basic operation needed by the continuous Dijkstra approach is to detect "event points" where two bisectors defined by pairs of obstacles intersect. Unfortunately, bisector computation is outside the primitive oracle's toolbox, and appears to be a complex problem in itself. We are unaware of any work directly addressing the complexity of computing these bisectors; however, the related problem of computing the medial axis of two curves has been investigated previously [8, 9]. In general, medial axes are *non-rational* curves even when the inputs are rational curves; they do not admit simple parametrizations or closed-form solutions, and therefore one must resort to numerical schemes for tracing them [8,9]. The bisectors we need can be described as medial axes of *involutes* of the obstacle arcs, i.e., curves traced by fixed-length taut strings wrapping around the arcs and, therefore, appear even harder to evaluate.

In this paper, we take a two-step approach to the problem of shortest paths among curved obstacles. First, we show how to compute shortest paths from a fixed source in $O(n \log n)$ time *assuming a perfect oracle that can localize the intersection of two bisectors*. We call this perfect oracle the *bisector oracle* for ease of reference. In particular, we can construct an $O(n \log n)$-space data structure encoding shortest paths from any point in free space to the fixed source under this oracle model. Although the algorithm exploits non-primitive operations involving bisectors of curved obstacles, it achieves an important intermediate goal: it shows that one can simulate the wavefront propagation even among curved obstacles in $O(n \log n)$ time and identify the linear number of tangent points on the curved obstacles that are needed for all the shortest paths from s.

In the second part of the paper, we show how to *implement* this bisector oracle, using only the primitive operations, to any desired relative accuracy ε. That is, we construct an $O(n \log n)$-space data structure in time $O(n \log(n/\varepsilon))$ that can report a $(1+\varepsilon)$-approximation of the shortest path distance from any query point q to the source s, using only the primitive operations allowable within the black box model of curved obstacles. Using standard techniques on this data structure, we can find the path length in $O(\log n)$ time and report a path in time $O(\log n + k)$, where k is the number of edges in the output path.

Our approach and the result should be contrasted with another approximation scheme proposed by Agarwal et al. [1]. Their scheme uses core-set ideas and constructs a Dudley-like approximation of convex polygons to reduce the complexity of the input, at the expense of relative error ε in path length. In particular, given n disjoint convex polygons of total complexity N, they construct an $O(\frac{n}{\sqrt{\varepsilon}})$ size sketch, and then invoke the Hershberger–Suri [16] shortest path algorithm directly on this simplified problem, with a running time of $O(N + \frac{n}{\sqrt{\varepsilon}} \log \frac{n}{\varepsilon})$. By contrast, our scheme directly, and adaptively, integrates the "obstacle polygonization" into the Hershberger–Suri framework of wavefront propagation, which yields a significant improvement in the overall space and time complexity.

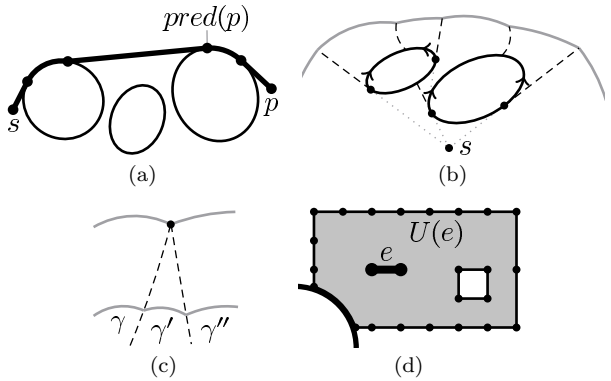


Figure 1: (a) An example shortest path. (b) The expanding wavefront, generators and bisectors. (c) A bisector event. (d) An edge e surrounded by the edges of $input(e)$.

The rest of the paper is organized as follows. In Section 2, we formally define the problem and describe the continuous Dijkstra framework. In Section 3, we describe our algorithm assuming access to the bisector oracle, and in Section 4 we describe how to implement the bisector oracle to arbitrary accuracy in the standard model of computation.

2 CONTINUOUS DIJKSTRA FRAMEWORK

We begin with a high-level description of the continuous Dijkstra framework on curved obstacles, and introduce some technical preliminaries. Let $\mathcal{O} = \{O_1, \ldots, O_k\}$ be a set of k disjoint obstacles in the plane. Each obstacle is a simple region bounded by a sequence of *arcs*, where each individual arc is of constant complexity, is monotone with respect to both x- and y-axes, and has no inflection points. The total number of arcs in all the obstacles is n. The plane minus the interiors of the obstacles is called the *free space*. A *shortest path* between two free-space points s (source) and p (destination) is a path of minimum total length connecting s to p and contained entirely within the free space. Such a path is denoted $\pi(s,p)$, and its length is denoted $d(s,p)$.

By the triangle inequality, a shortest path $\pi(s,p)$ is a sequence of edges, where each edge is either a subarc of a convex obstacle arc or a straight common tangent between two obstacle arcs (the edges incident to s and p being the only exception). For a fixed source point s, we define the *predecessor* of p, denoted $pred(p)$, to be the initial endpoint of the last obstacle arc in the sequence $\pi(s,p)$.[1] (See Figure 1a.) The point p may have multiple predecessors if there

[1]Unlike the case of polygonal obstacles, for curved obstacles, each "hop" from a point to its predecessor in general requires passing over two edges in the shortest path sequence. This seems necessary to define the predecessors consistently and uniquely.

are multiple shortest paths $\pi(s,p)$. The shortest path between p and a particular $pred(p)$ is obtained by following the tangent from p to the obstacle arc containing $pred(p)$ and then following this arc toward $pred(p)$. By repeatedly following the predecessors, one can construct the entire shortest path $\pi(s,p)$.

The *shortest path map* of a source point s, denoted $SPM(s)$, is a subdivision of the free space into regions in which all points have the same unique predecessor. The boundary between two regions corresponding to the predecessors p and q is a piece of a *bisector* curve, formed by the points that are equidistant (in terms of weighted shortest path distance) from p and q. The following lemma, whose proof appears in the full paper, bounds the size of a shortest path map.

LEMMA 2.1. *The shortest path map $SPM(s)$ has $O(n)$ vertices, edges, and simply connected regions.*

A continuous Dijkstra algorithm tracks the progress of an *expanding wavefront* consisting of all points whose shortest path distance from s is equal to some parameter t, which can be thought of as *time*. The algorithm simulates the expansion of the wavefront as time varies from 0 to $+\infty$. At any time, the wavefront is a sequence of *wavelets*, each emanating from a generator. A *generator* is a triple (a,p,w), where a is a convex obstacle arc (with an assigned clockwise or counterclockwise direction), p is a point on the arc a, and w is a non-negative weight. We say that a point q is reachable by a generator $\gamma = (a,p,w)$ if one can draw an obstacle-free path from p to q by following a in the assigned direction, to a point v on a such that $\overline{vq}$ is tangent to a, and then following $\overline{vq}$. The (weighted) distance between the generator γ and the point q, denoted $d(\gamma,q)$, is the length of this path plus w, i.e., $d(\gamma,q) = w + |\widehat{pv}| + |\overline{vq}|$.

A wavelet generated by γ at time t is a contiguous set of points q such that $d(\gamma,q) = t$ and $d(\gamma',q) \geq t$ for all other generators γ' in the wavefront. This wavelet is a piece of the *involute* curve traced by the end of a taut string of length $(t-w)$, anchored at p and wrapping around the arc a. If a point q is claimed at time t by a wavelet generated by γ, then the shortest path distance from s to q is $d(\gamma,q) = t$ and $pred(q) = p$. If q is on an obstacle arc a' and the tangent from q to a is also tangent to a', then a new generator (a',q,t) is added to the expanding wavefront.

As t increases, the points bounding the adjacent wavelets trace out the bisectors that form the boundaries of $SPM(s)$. (See Figure 1b.) The bisector between the wavelets of two generators γ and γ' consists of points p satisfying the equation $d(\gamma,p) = d(\gamma',p)$. A wavelet gets eliminated when the two bisectors bounding it intersect. Such an event is called a *bisector event*. In particular, if γ, γ', and γ'' are three consecutive generators of wavelets along the wavefront, the wavelet generated by γ' is hidden by those of γ and γ'' when the bisector of γ and γ' meets that of γ' and γ''. (See Figure 1c.) As we will see, computing the positions of bisector events is central to our algorithm. Wavelets are also eliminated by head-on collisions with obstacles and other wavelets, but those events are less important for our algorithm.

In their polygonal shortest path algorithm, Hershberger and Suri [16] simulate the wavefront expansion on a "conforming" subdivision that conforms to the polygon vertices. We adapt this idea to the case of curved obstacles by utilizing a subdivision that conforms to the arc endpoints. In particular, we construct a subdivision of the free space into $O(n)$ cells, where each cell is one of the connected regions formed by intersecting the free space with an axis-parallel rectangle or square annulus (a square with a square hole inside). The boundary of each cell is comprised of $O(1)$ edges (some of which are obstacle arcs) and contains *at most one* original arc endpoint. Each internal edge e of this subdivision is contained in a union of $O(1)$ cells, denoted $U(e)$, whose boundary is comprised of a set of edges (omitting obstacle arcs) denoted $input(e)$ with the following property: For each edge $f \in input(e)$, the straight line distance between e and f, denoted $d(e,f)$, is at least $2 \times \max(|e|,|f|)$. (See Figure 1d.) Such a subdivision can be constructed in $O(n \log n)$ time. Due to space limitations, we defer to the full paper the details of this construction, which is an adaptation of the conforming subdivision of [16] to curved obstacles.

To simulate the wavefront expansion, we compute the wavefront passing through each edge of the subdivision. Since computing an exact wavefront seems difficult, we compute two "one-sided wavefronts" for each edge, each representing the wavefront arriving from the generators on one side of the edge. A one-sided wavefront may contain wavelets that are covered by the wavefront coming from the other side and therefore are not in the true wavefront. To compute the wavefronts at an edge e, we propagate the wavefronts passing through each edge in $input(e)$ to e, and then merge the wavefronts coming from the same side to construct the final one-sided wavefronts at e. When a wavefront propagates across an edge e, we create artificial generators at the endpoints of e, weighted by the endpoints' distance from s. The bisectors between these artificial generators and their neighbors are determined by the tangents from the first and last real generators to the endpoints of e; this restricts the wavelets from non-artificial generators to pass through e.

To propagate the wavefronts consistently, we simulate a time parameter t and process the edges of the subdivision in a rough time order. In particular, the wavefronts of each edge e are constructed after e has been completely engulfed by the true wavefront, but within $|e|$ units of time thereafter. This implies that by the time the true wavefront engulfs an edge e, all edges $f \in input(e)$ that contribute a wavelet to the wavefront of e are processed (since $d(e,f) \geq 2|f|$). A processing time for e can be computed from the propagated wavefronts, by computing the time that an endpoint of e is first engulfed and adding $|e|$ to it. Details appear in the full paper.

To propagate the wavefront between two edges f and e, we propagate the wavefront of f through the cells of $U(e)$. Since $U(e)$ may contain holes, there may be multiple (but at most $O(1)$) topologically different shortest paths between e and f inside $U(e)$. We propagate the wavefront in multiple components, each defined by the particular sequence of subdivision edges in $U(e)$ it crosses. These wavefronts are later combined in a merge step at e. During propagation, some generators are eliminated due to bisector events, which are detected via calls to the oracle. Also note that, in contrast to the conforming subdivision for polygonal obstacles, the cells of our subdivision are not necessarily convex, because they contain curved obstacle arcs on their boundary. The interaction of the wavefront with these arcs not only eliminates some generators but also may create new generators in the wavefront. *We emphasize that this "on-the-fly" creation of generators during propagation is an important difference*

from the algorithm for polygonal obstacles, in which the generators are pre-given as subdivision vertices, waiting to be triggered. We describe the details of the wavefront propagation in Section 3, and simply state two important lemmas useful for our analysis. (Their proofs are given in the full paper.)

LEMMA 2.2. *Given the wavefronts from input(e) propagated to e, one can merge these wavefronts to construct the one-sided wavefronts at e using $O(1+k)$ bisector oracle operations and list modifications, where k is the total number of generators among the propagated wavefronts that do not appear in the final wavefronts at e.*

LEMMA 2.3. *The total number of times that generators are eliminated from wavefronts (during propagation or merging) is $O(n)$.*

3 SHORTEST PATHS USING THE BISECTOR ORACLE

We now describe the details of advancing the (topologically constrained) wavefront from an edge f to all edges e for which $f \in \mathit{input}(e)$. We assume access to a bisector oracle for determining bisector intersections. We also need to track the interaction of the wavefront with obstacles, which eliminates or creates generators. In [16], Hershberger and Suri track these events in *temporal order*. However, extending that approach to curved obstacles is complicated, so instead we simulate the generator events in *spatial order*. As a bonus, it turns out that spatial-order propagation is also easier to implement and analyze (even for the case of polygonal obstacles) and makes *on-the-fly generator creation* much easier.

3.1 Simulating the Wavefront in Spatial Order

A wavefront is a list of at most n generators in which each generator is assigned a "next bisector event" point, which is the position at which its two neighboring bisectors meet. (It is set to null for the first and the last generators in the wavefront, and for generators whose neighboring bisectors diverge.) *Computing bisector events is the only operation for which the bisector oracle is needed.* We store the generators in a balanced binary tree to perform the following operations in logarithmic time: insert, delete, concatenate, split, find previous/next element, and search. The search operation is used to find which generator in the list claims a query point by comparing distances to the generators. By storing $O(1)$ extra fields at each node of the binary tree, we also support the following priority query operations in logarithmic time: Update the next bisector event of a generator, find the generator whose bisector event has minimum/maximum x- or y-coordinate. Our wavefront data structure is fully persistent: each operation creates and modifies a copy of the wavefront, while keeping the old one intact and accessible. This can be achieved by path-copying [25] so that the per-operation time bound of $O(\log n)$ remains the same.

We use these data structures to propagate the one-sided wavefront at an edge e to every edge $f \in \mathit{output}(e) \equiv \{f \mid e \in \mathit{input}(f)\}$. The propagation is done cell by cell. We start by advancing the wavefront from e to its adjacent cell c, computing the wavefront passing through all edges bounding c. We then recursively repeat this process on cells that are adjacent to c using the computed wavefronts. Note that we only propagate to cells that are contained in $U(f)$ for some $f \in \mathit{output}(e)$. This guarantees that we propagate across a constant number of cells.

The essential building block of wavefront propagation is propagating a single wavefront across a single cell. We describe this operation in two parts: first we show how to propagate the wavefront across an empty rectangle, then across a subdivision cell possibly containing obstacle boundaries.

LEMMA 3.1. *Let $W(e)$ be a wavefront defined on an edge e of an empty rectangular cell c. Then we can propagate $W(e)$ to all the other edges of c in time $O((1+k)\log n)$ using $O(k)$ bisector oracle queries, where k is the number of bisector events of $W(e)$ that occur inside c.*

PROOF. Suppose that the rectangular cell c has coordinates $(x_{\mathrm{lo}}, x_{\mathrm{hi}}) \times (y_{\mathrm{lo}}, y_{\mathrm{hi}})$, and that the wavefront $W(e)$ travels up into c from an edge e on the bottom side of c. Our propagation algorithm critically relies on the fact that the bisectors of $W(e)$ intersect any horizontal line at most once, which is established in the full paper. Likewise each bisector defined by a pair of generators in $W(e)$ intersects the left or right side of c at most once. Consequently, we can perform a sweep across c, processing bisector events in y-order. We can visualize the sweep by considering the evolution of the 3-edge path $(x_{\mathrm{lo}}, y_{\mathrm{lo}}) \to (x_{\mathrm{lo}}, y) \to (x_{\mathrm{hi}}, y) \to (x_{\mathrm{hi}}, y_{\mathrm{lo}})$, as y continuously increases from y_{lo} to y_{hi}. We denote this path by $P(y)$. If $y' > y$ is the y-coordinate of the next bisector event above y, we show how to advance W from $P(y)$ to $P(y')$ in $O(\log n)$ time and $O(1)$ bisector event computations. (For simplicity we assume that all bisector events have distinct y-coordinates, but this is not a material restriction.)

Using priority queue operations on W, we can identify in $O(\log n)$ time a maximal prefix W_{lo} (resp., suffix W_{hi}) of W such that all bisector events in it have x-coordinates less than x_{lo} (resp., greater than x_{hi}). The remaining elements of W form a middle sublist W_{mid}, whose first and last generators have their bisector events in the x-interval $[x_{\mathrm{lo}}, x_{\mathrm{hi}}]$. Because bisectors are monotonic in y-coordinate, the lowest bisector event in c above y must be the next event for some generator in W_{mid}. By construction, the next bisector event above y involving generators in either W_{lo} or W_{hi} is outside c. So, the next event in c must belong to W_{mid}. Let γ_l and γ_r be the leftmost and rightmost generators in W_{mid}, with corresponding bisector event locations p_l, p_r, which are necessarily in the x-interval $[x_{\mathrm{lo}}, x_{\mathrm{hi}}]$. The tangent segments from γ_l and γ_r to p_l and p_r cross e and are not intersected by any bisector. It follows that if the lowest bisector event in W_{mid} is not p_l or p_r, it must lie between the indicated tangent segments, and hence lie inside the x-interval $[x_{\mathrm{lo}}, x_{\mathrm{hi}}]$. The "next bisector event" p' in W_{mid} with lowest y-coordinate is the event we seek, and we can find it in $O(\log n)$ time. We advance W from $P(y)$ to $P(y')$ by deleting the generator γ' associated with event p' from W, and recomputing the next events for the two neighbors of γ' in W.

Starting with $W = W(e)$, we repeatedly advance W until it represents the wavefront crossing $P(y_{\mathrm{hi}})$, the sides of c not containing e. The bisectors defined by the final state of W have no intersections inside c. For each edge f along $P(y_{\mathrm{hi}})$, we search in W with the edge endpoints to determine the

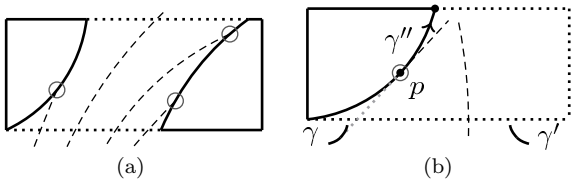


Figure 2: (a) Bisector-obstacle collisions in a wavefront propagation. (b) Creation of a new generator during propagation.

wavefront passing through f, then split W to produce the individual wavefront for f. This completes the proof. □

3.2 Creating Generators "On the Fly"

The setup for propagating a wavefront through a cell with obstacles on its boundary is similar to that in Section 3.1. Without loss of generality, assume that e is on the bottom side of c. Before we carry out the propagation, we temporarily partition c. If c is a cell that contains a square hole inside (recall square annulus cells), then we cut c into four subcells by cutting it with two lines parallel to e, each passing through an edge of the hole. We further partition c by drawing additional lines parallel to e, so that each resulting subcell contains at most two obstacle arcs, one on its left and one on its right side, and each side is bounded uniformly by a single arc or no arc (i.e., is transparent). It can easily be shown that $O(1)$ cuts are sufficient. We then propagate $W(e)$ from e through the resulting subcells, one subcell at a time.

Propagation among obstacles is more difficult than propagation through an empty cell, because there are two more kinds of bisector events to consider. The first new event type is collisions between bisectors and obstacle arcs (see Figure 2a). Because we prefer to impose as few requirements on the bisector oracle as possible, we do not compute intersections of bisectors with obstacle arcs explicitly. Instead, we detect these intersections after they have occurred, but before they can adversely affect the wavefront data structures.

The second new event type is creation of new generators by tangents from generators in the wavefront to obstacle arcs. In Figure 2b, γ and γ' are adjacent generators in W, and a is an arc bounding the left side of c. Let $p = \tau(a, \gamma)$ be the point on a that is one end of the common tangent between a and γ. If $d(\gamma, p) < d(\gamma', p)$, then the tangency at p creates a new generator $\gamma'' = (a, p, d(\gamma, p))$ that is inserted in W before γ. The bisector $b(\gamma'', \gamma)$ is the ray tangent to a at p extending away from γ. The region to the left of this bisector is not visible from γ and has γ'' as its predecessor. At the instant when γ'' is created, all the wavelets in W to the left of γ have already collided with a, so the first three generators in W are $\gamma'', \gamma, \gamma'$.

The following lemmas give the details of detecting and processing all three kinds of events inside c in y-coordinate order.

LEMMA 3.2. *Let γ and γ' be two adjacent generators in a wavefront propagating upward through a cell c, and suppose a is an obstacle arc on the left side of c. We can determine whether the bisector $b(\gamma, \gamma')$ hits arc a using $O(1)$ primitive arc operations.*

PROOF. Let $p' = \tau(a, \gamma')$; compute $d(\gamma, p')$ and $d(\gamma', p')$. If $d(\gamma, p') < d(\gamma', p')$, then every point on a is closer to γ than to γ'; $b(\gamma, \gamma')$ does not hit a. If $d(\gamma, p') \geq d(\gamma', p')$ then $b(\gamma, \gamma')$ hits a at or before p'. □

Within a single subcell the input to the propagation algorithm is a list of generators (a wavefront) in which each generator's next event is the intersection (if any) of the bisectors bounding its wavelet. We redefine the "next bisector event" of generators in a prefix and suffix of W. If $(\gamma, \gamma', \gamma'')$ are three consecutive generators in W, the natural next event for γ' is the intersection $z = b(\gamma, \gamma') \cap b(\gamma', \gamma'')$. If $b(\gamma, \gamma')$ and all bisectors to its left hit a, compute $p' = \tau(a, \gamma')$. We set the next event for γ' to the lower of p' and z. The next event for the first generator γ in W, which is normally null, is set to $\tau(a, \gamma)$. Symmetric definitions apply at the right of W. Every generator affected is sure to have an event inside c. One generator in the middle of W may have tangent events on both sides of c; we set the next event for this generator to be the lowest of the two tangents and the bisector event.

Propagation is similar to that in Lemma 3.1, with additional processing for tangent events. Consider a tangent event $p = \tau(a, \gamma)$. Without loss of generality assume a is the left boundary of c. When we process p, all bisectors to the left of γ have collided with a. If any are still present in W, we delete them. Let γ' be the right neighbor of γ in W. If $d(\gamma, p) < d(\gamma', p)$, then we create a new generator $\gamma'' = (a, p, d(\gamma, p))$ and prepend it to W. Otherwise (i.e., $d(\gamma, p) \geq d(\gamma', p)$), we delete γ from W, since $b(\gamma, \gamma')$ hits arc a at or before p.

When we process a normal bisector event that lies inside the bounding box of c, we have to check whether it is compatible with the obstacles. Suppose that generators γ, γ', and γ'' are consecutive in W, and event $p = b(\gamma, \gamma') \cap b(\gamma', \gamma'')$ is on the horizontal sweepline. If none of the three tangent segments from p to γ, γ', and γ'' intersects any obstacle, then p is a confirmed vertex of the local shortest path map for W. If any of the three tangents from p to one of the generators intersects the arc a on the left side of c, then all generators to the left of the rightmost such generator are deleted from W. (Symmetric operations apply for intersections with the right side of c.) These generators are to the left of bisectors that are known to hit a, and the fact that the bisector event p lies beyond a confirms that the sweepline is beyond the intersection of those bisectors with a.

In all cases, when we process an event, either a tangent or a true bisector event, we recompute bisector and tangent events for all affected generators. This includes computing additional tangent events whenever a bisector can be determined to intersect an obstacle arc.

LEMMA 3.3. *This sweepline algorithm processes every true bisector event and every tangent event inside c in y-coordinate order, and has the correct x-order of generators in W at all event y-coordinates and at the y-coordinates where bisectors intersect boundary arcs.*

PROOF. Bisectors are monotone in y-order even in the presence of obstacles, so all we need to do is be sure that each generator's events are processed in order. The initialization of next events ensures that true bisector events are processed before tangent events if and only if they occur below them in

y-order. This means that if a bisector hits an obstacle arc, that bisector exists in the wavefront before the collision y-coordinate. But tangent events are created for one generator of any bisector that hits an arc, and so all necessary tangent events are created before they need to be processed.

Bisector collisions with obstacle arcs may not be detected until after the y-coordinate where they occur, but no harm results: if a bisector $b(\gamma, \gamma')$ hits arc a at a point p, then for any point $q \in b(\gamma, \gamma')$ above p, at least one of the tangents from γ and γ' to q hits a, and so any generators that are no longer active will be detected and deleted from W. The sequence of generators in W is correct at the actual y-coordinate of the collision. □

As in Lemma 3.1, we split W at the endpoints of each edge on the boundary of c that it reaches. The total number of bisector events, bisector-obstacle collisions, and new generators processed is $O(k)$. Each costs $O(\log n)$ time to update the wavefront data structures.

The result of the operations described above is a sequence of persistent list structures that represent the generators in W at every y-coordinate in c. This establishes the following analogue of Lemma 3.1:

LEMMA 3.4. *Let $W(e)$ be a wavefront defined on an edge e of a subdivision cell c. Then we can propagate $W(e)$ to all the other edges of c in time $O((1+k)\log n)$ using $O(k)$ bisector oracle queries, where k is the number of generators eliminated from $W(e)$ during propagation.*

COROLLARY 3.5. *Given a subdivision edge e, one can propagate the one-sided wavefronts at e to all edges $f \in output(e)$ in $O((1+k)\log n)$ time and $O(k)$ bisector oracle calls, where k is the total number of generators that are eliminated during propagation.*

THEOREM 3.6. *The one-sided wavefronts on every edge e of the conforming subdivision can be constructed using $O(n)$ calls to the bisector oracle and $O(n \log n)$ time and space.*

PROOF. For each edge e, we propagate two one-sided wavefronts to edges of $output(e)$. By Corollary 3.5 and Lemma 2.3, this takes $O(n)$ oracle calls and $O(n \log n)$ time in total. By Lemmas 2.2 and 2.3, the merging of the propagated wavefronts to produce the final wavefronts also takes $O(n)$ oracle calls and $O(n \log n)$ time in total. The space complexity follows because we modify the persistent wavefront structures $O(n)$ times, and each modification requires $O(\log n)$ additional space. □

3.3 Shortest Path Query Data Structure

Once the wavefronts for all subdivision edges are computed, we can use the persistent search trees that we used to represent wavefronts as a shortest path query data structure. Given a query point q, one can report the shortest path from q to s as follows. We first locate the subdivision cell c that contains q. This can be done in $O(\log n)$ time using a precomputed point location data structure on the cells of the subdivision. During the wavefront expansion phase, a constant number of wavefronts $W_1, \ldots, W_m$ were propagated across c. We can find the shortest path distance and the predecessor of q by computing its shortest path distance restricted to each wavefront W_i, and then taking the minimum. We find the shortest path distance restricted to a single wavefront W as follows. Without loss of generality, assume that W was propagated from an edge on the bottom side of c. The propagation of W through c produced a sequence of wavefront data structures $W(y_1), \ldots, W(y_k)$, where each y_i is the y-coordinate of a bisector event, the y-coordinate of a tangent point where a generator is created, or the lowest y-coordinate of c. Let y_q be the y-coordinate of q and let y be the largest element of $\{y_1, \ldots, y_k\}$ smaller than y_q. By Lemma 3.3, $W(y)$ represents the correct wavefront at the query position y_q. We perform binary search on $W(y)$ to find which generator γ claims q. The shortest path to q, restricted to W, is $\pi(\gamma, q)$. We note that if W is propagated through subcells of c, we apply these operations on the subcell that contains q.

4 APPROXIMATING THE BISECTOR ORACLE

The shortest path algorithm described above relies on exact localization of bisector events, and the algorithm performs $O(n)$ such operations. However, these operations are not supported in the primitive oracle model, and so in this section we show how to localize the bisector events approximately using only the primitive oracle operations. Our approach is inspired by *backward error analysis* in numerical analysis: we compute exact shortest paths for a slightly modified (polygonized) set of obstacles. In particular, we select $O(n)$ short subarcs on the obstacle boundaries and replace each of them by a two-edge polygonal path. This *local polygonization* is performed on the fly, during wavefront propagation. The shortest paths that do not contain tangents to the polygonized subarcs are unaffected by this polygonization, while the lengths of all the remaining shortest paths are preserved to within a $(1+\varepsilon)$ factor. The polygonized subarcs are chosen so that the bisector events of the modified arcs can be computed using the primitive oracle operations in constant time. We answer shortest path queries approximately using the exact data structure for the modified obstacles. If an approximate shortest path that avoids the original (unmodified) obstacles is desired, one can be obtained by local modifications of the exact shortest path for the polygonized obstacles.

4.1 Local polygonization

Consider a generator $\gamma = (a, p, w)$. Let u and v be two points on a, with u closer to p. The portion of a between u and v lies inside the triangle defined by the segment $\overline{uv}$ and the two lines tangent to a at u and v. We refer to this triangle as the *error triangle* determined by u and v. The shape of a inside the triangle is unknown, but the primitive oracle tells us its length $\ell = |\widehat{uv}|$. We define two triangles $\Delta_{min}(a, u, v)$ and $\Delta_{max}(a, u, v)$ that both have base $\overline{uv}$, lie inside the error triangle, and have perimeter $|\overline{uv}| + \ell$ (see Figure 3a). One edge of $\Delta_{min}(a, u, v)$ is tangent to a at v, and one edge of $\Delta_{max}(a, u, v)$ is tangent to a at u. In each triangle, the position of the vertex opposite $\overline{uv}$ is chosen so that the two edges incident to it (one tangent to e, one not) have lengths that sum to ℓ. Replacing the subarc of a between u and v by one of these two triangles produces a simpler generator whose shortest path distances form either an upper or lower bound on original shortest path distances, as shown in the next lemma.

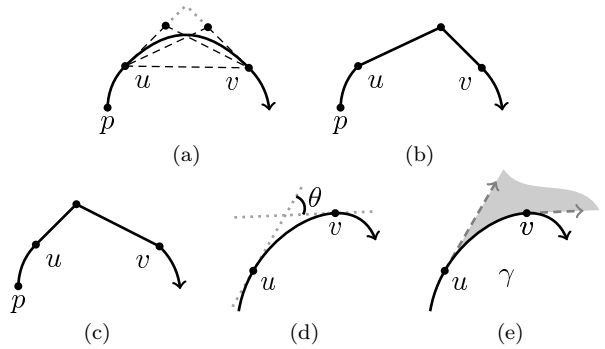


Figure 3: (a) The triangles $\Delta_{min}(a,u,v)$ and $\Delta_{max}(a,u,v)$. (b) Modified obstacle for γ_{min} and (c) for γ_{max}. (d) Angular size of $\widehat{uv}$. (e) Wedge of γ.

LEMMA 4.1. *Let γ_{min} be the generator obtained from γ by replacing the subarc of a between u and v by the two edges of $\Delta_{min}(a,u,v)$ opposite $\overline{uv}$. Define γ_{max} similarly for $\Delta_{max}(a,u,v)$ (see Figures 3b and 3c). Then for any point q outside of the triangles $\Delta_{min}(a,u,v)$ and $\Delta_{max}(a,u,v)$, the following holds:*

$$d(\gamma_{min},q) \;\le\; d(\gamma,q) \;\le\; d(\gamma_{max},q).$$

PROOF. We first show that $d(\gamma_{min},q) \le d(\gamma,q)$. If q's tangent to a is on the arc $\widehat{pu}$, the shortest paths $d(\gamma_{min},q)$ and $d(\gamma,q)$ are the same and the proposition holds. Otherwise, the arc $\widehat{pu}$ is included in both shortest paths. Let y be the third vertex of the triangle $\Delta_{min}(a,u,v)$. If q and v are on opposite sides of the line $\overline{uy}$, then γ_{min} reaches q directly from u with a straight line (in the shortest possible way); thus $d(\gamma_{min},q) \le d(\gamma,q)$. Otherwise, if q and u are on the same side of the line $\overline{yv}$, then both shortest paths follow their respective obstacles, pass v, and reach q from the same tangent point. By construction of $\Delta_{min}(a,u,v)$, both paths travel equal distances and the proposition holds.

The remaining case is when q is in the open region bounded by the lines $\overline{uy}$ and $\overline{yv}$. We consider two subcases. In the first case, the tangent line drawn from q to a does not intersect the segment $\overline{yv}$ (Figure 4a). Let t be the corresponding tangent point and let z be the intersection of the tangent with the line $\overline{uy}$. Then, $d(\gamma_{min},q) \le |\widehat{pu}| + |\overline{uz}| + |\overline{zq}|$ and $d(\gamma,q) \ge |\widehat{pu}| + |\overline{ut}| + |\overline{tz}| + |\overline{zq}|$. By the triangle inequality, $|\overline{uz}| \le |\overline{ut}| + |\overline{tz}|$, and this implies $d(\gamma_{min},q) \le d(\gamma,q)$. In the second case, the tangent from q to a intersects $\overline{yv}$ (Figure 4b). Let t be the tangent point and let z be the intersection. Then $d(\gamma_{min},q) \le |\widehat{pu}| + |\overline{uy}| + |\overline{yz}| + |\overline{zq}|$. By the definition of $\Delta_{min}(a,u,v)$, $|\overline{uy}| + |\overline{yz}| = |\widehat{uv}| - |\overline{zv}|$. Hence $d(\gamma_{min},q) \le |\widehat{pu}| + |\widehat{uv}| - |\overline{zv}| + |\overline{zq}|$. Similarly, since $|\widehat{ut}| = |\widehat{uv}| - |\widehat{tv}|$, we can write $d(\gamma,q) \ge |\widehat{pu}| + |\widehat{uv}| - |\widehat{tv}| + |\overline{tz}| + |\overline{zq}|$. It follows that $d(\gamma_{min},q) - d(\gamma,q) \le |\widehat{tv}| - (|\overline{tz}| + |\overline{zv}|) \le 0$, and thus the proposition holds.

We now show that $d(\gamma,q) \le d(\gamma_{max},q)$. If q's tangent to a is on the arc $\widehat{pu}$, the proposition clearly holds. Otherwise, let y be the third vertex of the triangle $\Delta_{max}(a,u,v)$. If q and u are on the same side of the line $\overline{yv}$, then the shortest path from γ_{max} to q follows the obstacle boundary and passes through v. The shortest path from γ to q also passes through v, and so because $|\overline{uy}| + |\overline{yv}| = |\widehat{uv}|$, $d(\gamma_{max},q) = d(\gamma,q)$. The remaining case is when q is in the region bounded by the lines $\overline{uy}$ and $\overline{yv}$. We consider two subcases. In the first case, the tangent line from q to a intersects the segment $\overline{uy}$ (Figure 4c). Let z be the point of intersection. Then, $d(\gamma,q) \le |\widehat{pu}| + |\overline{uz}| + |\overline{zq}|$ and $d(\gamma_{max},q) = |\widehat{pu}| + |\overline{uz}| + |\overline{zy}| + |\overline{yq}|$. By the triangle inequality, $|\overline{zq}| \le |\overline{zy}| + |\overline{yq}|$, and this implies $d(\gamma,q) \le d(\gamma_{max},q)$. In the second case, the tangent from q to a does not intersect $\overline{uy}$ (Figure 4d). This implies that the segment $\overline{yq}$ intersects the arc $\widehat{uv}$ twice; let z be the intersection point closer to q. We write $d(\gamma,q) \le |\widehat{pu}| + |\widehat{uz}| + |\overline{zq}| = |\widehat{pu}| + |\widehat{uv}| - |\widehat{zv}| + |\overline{zq}|$. Also, $d(\gamma_{max},q) = |\widehat{pu}| + |\overline{uy}| + |\overline{yz}| + |\overline{zq}| = |\widehat{pu}| + |\widehat{uv}| - |\overline{yv}| + |\overline{yz}| + |\overline{zq}|$. It follows that $d(\gamma,q) - d(\gamma_{max},q) \le |\overline{yv}| - (|\overline{yz}| + |\widehat{zv}|) \le 0$, and thus the proposition holds. □

Our algorithm chooses a set of obstacle subarcs whose error triangles are disjoint. Within each error triangle, we replace the subarc by the two non-base edges of Δ_{max} or Δ_{min}—this is the transformation, described above, that converts a generator γ to γ_{max} or γ_{min}. We refer to this process as *local polygonization*. The key to bounding approximation accuracy is bounding the size of the subarcs. We say that a subarc $\widehat{uv}$ of a generator arc is of *angular size* θ if the directions of the tangent lines through u and v differ by an angle of θ (see Figure 3d). The lemma below summarizes how we achieve $(1+\varepsilon)$ approximation accuracy by local polygonization.

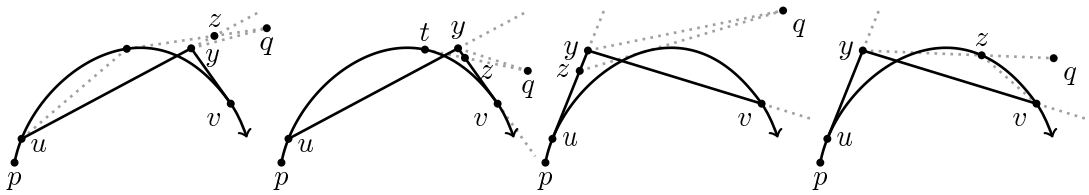
LEMMA 4.2. *Let $\mathcal{O}'$ be a modified set of non-intersecting obstacles obtained by polygonizing (as described above) a number of disjoint subarcs in $\mathcal{O}$, each of size at most ε, where $0 \le \varepsilon \le 1$. Then for any two points p and q in the free space of both $\mathcal{O}$ and $\mathcal{O}'$, the shortest path distance between p and q in the free space of $\mathcal{O}$, denoted $d_{\mathcal{O}}(p,q)$, and the corresponding distance $d_{\mathcal{O}'}(p,q)$ in the free space of $\mathcal{O}'$ satisfy the following inequalities:*

$$d_{\mathcal{O}'}(p,q) \le (1+\varepsilon) d_{\mathcal{O}}(p,q) \quad \textit{and} \quad d_{\mathcal{O}}(p,q) \le (1+\varepsilon) d_{\mathcal{O}'}(p,q)$$

PROOF. Let $\pi_{\mathcal{O}}$ and $\pi_{\mathcal{O}'}$ denote shortest paths in the free space of $\mathcal{O}$ and $\mathcal{O}'$, resp., corresponding to the distances $d_{\mathcal{O}}(p,q)$ and $d_{\mathcal{O}'}(p,q)$. The error triangles of the locally polygonized arcs are disjoint, so we can consider the effect of each error triangle on path length independently.

We first show that $d_{\mathcal{O}'}(p,q) \le (1+\varepsilon) d_{\mathcal{O}}(p,q)$. The shortest path $\pi_{\mathcal{O}}$ intersects the interiors of zero or more error triangles. Let p' and q' be the first and last intersections of $\pi_{\mathcal{O}}$ with one such triangle Δ. Points p' and q' belong to the two sides of Δ opposite the base—the base is not in the free space of either $\mathcal{O}$ or $\mathcal{O}'$, and if p' and q' were on the same side, $\pi_{\mathcal{O}}$ would not intersect the interior of Δ. The boundary of Δ between p' and q' is a path in the free space of both $\mathcal{O}'$ and $\mathcal{O}$. Denote this boundary path by $\widehat{p'q'}$. Since the external angle between the two edges of $\widehat{p'q'}$ is at most ε, the length of $\widehat{p'q'}$ is at most $|\overline{p'q'}|/\cos\varepsilon$, by simple trigonometry. We replace the section of $\pi_{\mathcal{O}}$ between p' and q' by $\widehat{p'q'}$, and perform a similar transformation for every error triangle that $\pi_{\mathcal{O}}$ intersects. This gives a path $\widehat{\pi_{\mathcal{O}}}$ whose length is an upper bound on $d_{\mathcal{O}'}(p,q)$. Likewise, because $|\overline{p'q'}| \le d_{\mathcal{O}}(p',q')$, the path $\overline{\pi_{\mathcal{O}}}$ obtained from $\pi_{\mathcal{O}}$ by replacing $\pi_{\mathcal{O}}(p',q')$ by the segment $\overline{p'q'}$ in each error triangle lower bounds the length of $\pi_{\mathcal{O}}$. Summing the results of local replacement over all error triangles, we have $|\overline{\pi_{\mathcal{O}}}| \le d_{\mathcal{O}}(p,q)$ and $d_{\mathcal{O}'}(p,q) \le |\widehat{\pi_{\mathcal{O}}}| \le |\overline{\pi_{\mathcal{O}}}|/\cos\varepsilon$. For $0 \le \varepsilon \le 1$, we have $1/\cos\varepsilon \le 1+\varepsilon$, and so

$$d_{\mathcal{O}'}(p,q) \le (1+\varepsilon) d_{\mathcal{O}}(p,q).$$



(a) (b) (c) (d)

Figure 4

To complete the proof, notice that the same argument applies unchanged if we reverse the rôles of $\pi_{\mathcal{O}}$ and $\pi_{\mathcal{O}'}$. Hence

$$d_{\mathcal{O}}(p,q) \leq (1+\varepsilon) d_{\mathcal{O}'}(p,q).$$

□

4.2 Localizing bisector events

Our goal is to polygonize obstacle arcs so that bisector curves and bisector events involving them can be computed precisely using primitive oracle operations. However, we do not want to polygonize indiscriminately. There is no need to know exact bisectors for regions of space that do not contain a bisector event. Our idea is to refine the region known to contain a bisector event *without polygonizing the contributing arcs* until the tangents from each arc to the bisector-containing region bound a subarc of angular size at most ε. We can then polygonize within each remaining subarc, so that the position of the bisector event is known exactly, and by Lemma 4.2 all distances to the polygonized generators are within the specified relative error ε of distances to the original generators.

Let us consider how to localize bisector events. We are given three generators, adjacent in the wavefront $W(e)$ at some edge e. Without loss of generality let us assume that e is horizontal and $W(e)$ propagates upward across e. Let us identify the generators as L, M, and R, and the bisectors they induce as $b(L,M)$ and $b(M,R)$. A bisector event $\mathit{event}(L,M,R)$ occurs where (if) $b(L,M)$ and $b(M,R)$ intersect and are replaced by $b(L,R)$. Note that $\mathit{event}(L,M,R)$ does not exist if $b(L,M)$ and $b(M,R)$ diverge.

Given a generator $\gamma = (a,p,w)$ and two points u and v on a, we define the *wedge* of γ induced by u and v to be the region of free space adjacent to a and bounded by the tangent rays at u and v whose directions are compatible with a. See Figure 3e. We denote the region by $\mathit{wedge}(\gamma,u,v)$. When the tangent points u and v are known from context, we shorten the notation to $\mathit{wedge}(\gamma)$. The measure of $\mathit{wedge}(\gamma,u,v)$, denoted by $|\mathit{wedge}(\gamma,u,v)|$ or $|\mathit{wedge}(\gamma)|$, is the angular size of $\widehat{uv}$.

Our algorithm for localizing bisector events has multiple steps. We summarize the result in the following theorem, whose proof is presented in the full paper.

Theorem 4.3.

Setup: *Let L, M, and R be adjacent generators in a wavefront $W(e)$, propagating upward across a horizontal edge e. A wedge is specified for each generator, and $\mathit{event}(L,M,R)$ is guaranteed to lie in the intersection $\mathit{wedge}(L) \cap \mathit{wedge}(M) \cap \mathit{wedge}(R)$, if the bisector event exists at all.*

Angular Condition: *Each of $|\mathit{wedge}(L)|$, $|\mathit{wedge}(M)|$, and $|\mathit{wedge}(R)|$ is at most $\pi/2$.*

Conclusion: *We can either (1) prove that $\mathit{event}(L,M,R)$ does not exist or (2) locally polygonize L, M, and R so that the bisector event of the polygonized generators is known exactly, and distances measured to the polygonized generators match distances to the original generators to within relative error ε.*

Bound: *The operation takes $O(\log \frac{1}{\varepsilon})$ time and primitive oracle queries.*

The idea of the proof is to bisect $\mathit{wedge}(L)$, $\mathit{wedge}(M)$, and $\mathit{wedge}(R)$ repeatedly until each has measure at most ε. This turns out to be nontrivial, because we must guarantee that $\mathit{event}(L,M,R)$ is contained in the intersection of the wedges, and the geometry of the bisectors themselves limits which wedges we can subdivide.

Once two of the three wedges have measure at most ε, refining the third wedge is easy: we polygonize the two small wedges to determine exactly the bisector for the corresponding generators. Then we do binary search along that bisector—a one-dimensional search—to find an ε-size wedge of the third generator that contains the bisector event.

We do not have space to describe the wedge refinement process in full detail, but to give a flavor of the proof, we describe the initial refinement step, which guarantees at its conclusion that either

$$|\mathit{wedge}(M)| \leq \varepsilon \quad \text{or} \quad \max(|\mathit{wedge}(L)|, |\mathit{wedge}(R)|) \leq \varepsilon.$$

The proof is based on an operation, described below, that takes a pair of generators, either (L,M) or (M,R), and bisects one of the wedges in the pair. (We cannot control which of the two wedges is bisected.) We apply this operation repeatedly to the pair with the larger non-M wedge. That is, if $|\mathit{wedge}(L)| > |\mathit{wedge}(R)|$ we bisect one of $\mathit{wedge}(L)$ and $\mathit{wedge}(M)$; otherwise we bisect one of $\mathit{wedge}(M)$ and $\mathit{wedge}(R)$. We apply the operation until the improved angle bound holds. Note that if either $|\mathit{wedge}(L)|$ or $|\mathit{wedge}(R)|$ reaches ε, the wedge is never bisected again. Bisection continues until one of the other wedges reaches ε, at which point the target condition holds. The total number of bisections is at most $3(\lg \frac{1}{\varepsilon} + \lg \frac{\pi}{2}) = O(\log \frac{1}{\varepsilon})$.

We now describe the bisection operation in detail. Without loss of generality suppose that the operation is applied to (L,M). For each of L and M, we construct the tangent that bisects the angle of the wedge. Let p be the intersection point of the two tangent rays. (See Figure 5.) Compute the distances $d(L,p)$ and $d(M,p)$ and consider what they imply about the position of $b(L,M)$. The bisector travels monotonically across the wedges of both L and M. In particular, if $d(L,p) < d(M,p)$, then $b(L,M)$ does not enter

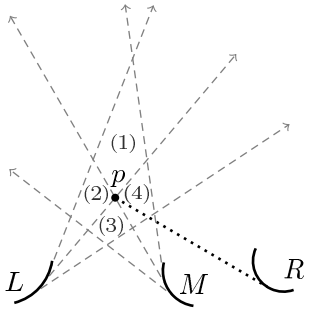


Figure 5

the upper left quadrant, labeled (2) in Figure 5. Similarly, if $d(L,p) > d(M,p)$, then $b(L,M)$ does not enter the lower right quadrant, labeled (4).

Now compute the distance $d(R,p)$. Comparing $d(R,p)$ with $d(M,p)$ lets us determine whether $b(M,R)$ passes above or below p. Because $event(L,M,R)$ is the intersection of $b(L,M)$ and $b(M,R)$, we have enough information to restrict it to two of the four quadrants:

$$\begin{aligned}
d(L,p) < d(M,p) \text{ and } d(R,p) < d(M,p) &\Rightarrow \text{quads (3,4)}\\
d(L,p) < d(M,p) \text{ and } d(R,p) > d(M,p) &\Rightarrow \text{quads (1,4)}\\
d(L,p) > d(M,p) \text{ and } d(R,p) < d(M,p) &\Rightarrow \text{quads (2,3)}\\
d(L,p) > d(M,p) \text{ and } d(R,p) > d(M,p) &\Rightarrow \text{quads (1,2)}
\end{aligned}$$

Note that each of these quadrant pairs lies in one of the half-wedges produced by bisection. We replace the wedge of L or M by the appropriate half-wedge and continue until the target condition is met. Each of the $O(\log \frac{1}{\varepsilon})$ operations requires a constant number of primitive oracle queries.

The final step of the proof starts with $|wedge(M)| \le \varepsilon$ and refines one of the two remaining wedges. This step focuses on the geometry of a bisector in an intersection of wedges. We use the following lemma:

LEMMA 4.4. *If p is a point on the bisector of two generators γ and γ', then the line tangent to the bisector at p is the angular bisector of the angle formed by the two tangents from p to γ and γ'.*

This means that, for example, the direction of $b(L,M)$ inside $wedge(L) \cap wedge(M)$ lies in a specific angular range determined by the directions of the rays bounding the wedges. If $|wedge(M)| \le \varepsilon$, the direction of $b(L,M)$ is even more tightly bounded. If the physical width of $wedge(M)$ is small relative to that of either $wedge(L)$ or $wedge(R)$ where they intersect (which can be ensured by $O(\log \frac{1}{\varepsilon})$ further refinements of $wedge(M)$), then the angle of $wedge(L)$ (or the angle of $wedge(R)$) spanned by $b(L,M)$ (or $b(M,R)$) inside the wedge intersection must be small, in fact at most ε. We can reduce $|wedge(L)|$ (or $|wedge(R)|$) to ε by $O(\log \frac{1}{\varepsilon})$ pruning steps.

The algorithm of Section 3 needs two types of bisector operations. The first, discussed above, is localizing the position of a bisector event in the plane. The second operation, used when one-sided wavefronts are merged at a destination edge, determines the relative order of bisector intersections with a line segment. This can be done by applying event localization—two bisectors determined by three generators intersect a segment in different orders depending on the position of their bisector event relative to the line supporting the segment—or more simply by binary search for each bisector's intersection with the segment.

Polygonizations accumulate as the wavefront propagation algorithm runs. That is, we maintain a sequence of the $\Delta_{max}/\Delta_{min}$ polygonizations that have been applied to each arc. A tangent query whose answer lies within the error triangle of a previous polygonization returns the appropriate vertex of the polygonization. No polygonization, once performed, is ever undone: future polygonizations are forced to be disjoint from earlier ones. In the worst case, this adds an $O(\log n)$ overhead to each of $O(n)$ oracle queries, which is of the same order of magnitude as the other data structure costs.

It is important that polygonizations on different arcs are disjoint, to avoid changing the topology of free space. We achieve this by an initial partitioning of the obstacle arcs. We break each arc at cell boundaries, so each arc is contained in a single cell. Within each cell, we compute the common tangents of all $O(1)$ arcs, and split the arcs at the tangent endpoints. This guarantees that no error triangle crosses a common tangent, and hence that all polygonizations in a cell are disjoint. Because each arc is x- and y-monotone, any error triangle lies in the arc bounding box, and hence inside its cell, if the cell is simple. If the cell derives from a square annulus, we break its arcs at the endpoints of the tangents from the inner square to the arcs. This ensures that no error triangle crosses into the inner square. This partitioning adds only $O(1)$ arc endpoints per cell, so the total number of arcs remains $O(n)$.

4.3 Answering shortest path queries

Given a query point q, we can use the algorithm of Section 3.3 to compute the length of a shortest path from q to s among the polygonized obstacles. The length of this path is within a $(1+\varepsilon)$ factor of the true shortest path length.

To compute an approximate shortest path that avoids the unmodified obstacles is a bit more work. The problem is that the polygonizing edges of Δ_{min} and Δ_{max} are not entirely in free space. A path that crosses an error triangle of a polygonized arc might intersect the arc itself. To find potential path intersections with error triangles, we build a ray shooting structure on the obstacles [3, 14]. We link the obstacles into a shortest path tree by connecting each obstacle to s or to another obstacle by a predecessor edge. Because shortest paths from a single source do not cross or cycle, this creates an embedded tree on the obstacles. We then augment the obstacles by adding in all the polygonizing error triangles. We build a ray shooting data structure for this tree of augmented obstacles in $O(n \log n)$ time.

Let $\overline{qx}$ be the free-space edge from a query point q leading toward $pred(q)$. We use ray shooting to find the first shortest path tree point hit by $\overrightarrow{qx}$. The segment $\overline{qx}$ does not cross any of the predecessor links in the shortest path tree, so the first point the ray hits is either an obstacle arc or an error triangle. If it hits an error triangle, we perform a line intersection query on the arc inside the error triangle, and detour the path along the arc if necessary to bypass any portion of $\overline{qx}$ that is not in free space. This modification takes $O(\log n)$ time and $O(1)$ primitive oracle operations to repair one predecessor segment. We can process a k-edge shortest path in $O(k \log n)$ time. Alternatively, if we precompute the path from each generator to its predecessor in $O(n \log n)$

total time, the query time to produce a free-space path reduces to $O(k + \log n)$. The length of the modified path, with detours, is still within a $(1 + 2\varepsilon + o(\varepsilon))$ factor of the true shortest path length, because every error triangle has measure at most ε.

We can now state our main result.

THEOREM 4.5. *Given a set of curved obstacles $\mathcal{O}$ with total complexity n and a source point s in the plane, one can construct (using only primitive operations) a data structure in $O(n \log n + n \log \frac{1}{\varepsilon})$ time and $O(n \log n)$ space, such that the length of an obstacle-free approximate shortest path from a query point q to s can be computed in $O(\log n)$ time. The path itself can be reported in $O(\log n + k)$ time, where k is the complexity of the path. The reported path's length is within a $(1 + \varepsilon)$ factor of the true shortest path length from q to s.*

References

[1] P. K. Agarwal, R. Sharathkumar, and H. Yu. Approximate Euclidean shortest paths amid convex obstacles. In *Proc. of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 283–292, 2009.

[2] E. Chang, S. Choi, D. Kwon, H. Park, and C. Yap. Shortest path amidst disc obstacles is computable. *International Journal of Computational Geometry and Applications*, 16(05n06):567–590, 2006.

[3] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12:54–68, 1994.

[4] D. Z. Chen and H. Wang. Computing shortest paths amid pseudodisks. In *Proc. of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 309–326, 2011.

[5] D. Z. Chen and H. Wang. Computing shortest paths among curved obstacles in the plane. `http://arxiv.org/abs/1103.3911v1`, 2011. A condensed version of the paper appears in these proceedings (*SoCG 2013*).

[6] J. Chen and Y. Han. Shortest paths on a polyhedron, Part I: Computing shortest paths. *International Journal of Computational Geometry and Applications*, 6(2):127–144, 1996.

[7] L. P. Chew. Planning the shortest path for a disc in $O(n^2 \log n)$ time. In *Proc. of the 1st Annual Symposium on Computational Geometry*, pages 214–220, 1985.

[8] G. Elber and M.-S. Kim. Bisector curves of planar rational curves. *Computer-Aided Design*, 30(14):1089 – 1096, 1998.

[9] R. T. Farouki and J. K. Johnstone. Computing point/curve and curve/curve bisectors. In *Proc. of the 5th IMA Conference on the Mathematics of Surfaces*, pages 327–354, 1994.

[10] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991.

[11] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.

[12] S. Har-Peled. Approximate shortest-path and geodesic diameter on convex polytopes in three dimensions. *Discrete & Computational Geometry*, 21:216–231, 1999.

[13] J. Hershberger and L. J. Guibas. An $O(n^2)$ shortest path algorithm for a non-rotating convex body. *Journal of Algorithms*, 9(1):18–46, 1988.

[14] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.

[15] J. Hershberger and S. Suri. Practical methods for approximating shortest paths on a convex polytope in R^3. *Computational Geometry*, 10(1):31–46, 1998.

[16] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal of Computing*, 28(6):2215–2256, 1999.

[17] S. Kapoor and S. N. Maheshwari. Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles. In *Proc. of the 4th Annual Symposium on Computational Geometry*, pages 172–182, 1988.

[18] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.

[19] J. S. Mitchell. Shortest paths among obstacles in the plane. *International Journal of Computational Geometry and Applications*, 06(03):309–332, 1996.

[20] J. S. B. Mitchell. Shortest paths among obstacles in the plane. In *Proc. of the 9th Annual Symposium on Computational Geometry*, pages 308–317, 1993.

[21] J. S. B. Mitchell. Shortest paths and networks. In *Handbook of Discrete and Computational geometry*, pages 445–466, 1997.

[22] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.

[23] C. H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Information Processing Letters*, 20(5):259–263, 1985.

[24] M. Pocchiola and G. Vegter. Topologically sweeping visibility complexes via pseudotriangulations. *Discrete & Computational Geometry*, 16(4):419–453, 1996.

[25] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986.

[26] M. Sharir. On shortest paths amidst convex polyhedra. *SIAM Journal on Computing*, 16(3):561–572, 1987.

[27] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM Journal on Computing*, 15(1):193–215, 1986.

[28] J. A. Storer and J. H. Reif. Shortest paths in the plane with polygonal obstacles. *Journal of ACM*, 41(5):982–1012, 1994.

[29] E. Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Information Processing Letters*, 20(4):167–171, 1985.