Computing Approximate Shortest Paths on Convex Polytopes*

Pankaj K. Agarwal† Sariel Har-Peled‡ Meetesh Karia§

Abstract

The algorithms for computing a shortest path on a polyhedral surface are slow, complicated, and numerically unstable. We have developed and implemented a robust and efficient algorithm for computing approximate shortest paths on a convex polyhedral surface. Given a convex polyhedral surface P in $\mathbb{R}^3$, two points $s, t \in P$, and a parameter $\varepsilon > 0$, it computes a path between s and t on P whose length is at most $(1+\varepsilon)$ times the length of the shortest path between those points. It first constructs in time $O(n/\sqrt{\varepsilon})$ a graph of size $O(1/\varepsilon^4)$, computes a shortest path on this graph, and projects the path onto the surface in $O(n/\varepsilon)$ time, where n is the number of vertices of P. In the post-processing we have added a heuristic that considerably improves the quality of the resulting path.

*Work by P.A. was supported by Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by NSF grants EIA–9870724 and CCR–9732787 and by a grant from the U.S.-Israeli Binational Science Foundation.Work by S.H.-P. was supported by Army Research Office MURI grant DAAH04-96-1-0013.

†Center for Geometric Computing, Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA. E-mail: pankaj@cs.duke.edu

‡Center for Geometric Computing, Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA. E-mail: sariel@cs.duke.edu

§Trilogy Software, Inc., 6034 W. Courtyard Drive, Austin, TX 78730, USA. E-mail: Meetesh_Karia@trilogy.com

Computational Geometry 2000 Hong Kong China

1 Introduction

Let P be a polyhedral surface in $\mathbb{R}^3$ with a total of n vertices. Without loss of generality, we can assume that the faces of P are triangules. Given two points $s, t \in P$, we want to compute a path $\pi_P(s,t)$ from s to t of minimum length that lies on P; $\pi_P(s,t)$ is usually, but not always, unique. Let $d_P(s,t)$ denote the length of $\pi_P(s,t)$. Computing a shortest path on a polyhedral surface is a widely studied problem in computational geometry, robotics, and geographic information systems, as it arises in a wide range of applications including route planning in geospatial data [8, 31], military mission planning [5, 17, 18, 20, 21, 28], injection molding [15], computer-assisted surgery. See [25] for a survey of such applications.

Sharir and Schorr [29] were the first to provide an efficient algorithm for computing a shortest path on convex polyhedral surfaces.[1] Their algorithm runs in $O(n^3 \log n)$ time and relies on the fact that a shortest path on the surface of a polytope *unfolds* into a straight line. The running time of this algorithm was improved to $O(n^2 \log n)$ by Mitchell et al. [24]; they also showed that their algorithm works for non-convex polyhedra as well. Chen and Han [6] further improved the running time to $O(n^2)$. Kapoor [16] has recently announced a near-linear time algorithm for this problem. These exact algorithms are too complicated and slow to be of any practical use, and they all suffer from numerical problems because the shortest path may require an exponential number of bits (as a function of the maximum number of bits used to specify the coordinates of a vertex of P). All of this has sparked interest in developing algorithms to find an approximate shortest path in near-linear

[1]The earliest reference of this problem that we are aware of is the following puzzle by Henrey Ernest Dudeney, a famous English puzzlist, which he published in an English newspaper in 1903: If there are a spider and a fly on the walls of a rectangular room, what is the shortest path the spider can take to catch the fly?

time. For a given $\varepsilon > 0$, a path Π from s to t on P is called an $(1+\varepsilon)$-approximate shortest path if $|\Pi| \leq (1+\varepsilon)d_P(s,t)$.

Hershberger and Suri [14] presented a simple algorithm that computes a 2-approximate shortest path on a convex surface in $O(n)$ time. Varadarajan and Agarwal [30] described a subquadratic algorithm for a constant-factor approximate shortest path on a polyhedral terrain. Recently, there has been more practical work on developing and implementing simple approximation algorithms for shortest paths on polyhedral surfaces. Aleksandrov *et al.* [4], Mata and Mitchell [22], Lanthier *et al.* [19], proposed a number of approximation algorithms for computing a shortest path on a polyhedral surface, especially for unweighted and weighted terrains. Using the idea by Papadimitriou [27], they place Steiner points along the edges of P, construct a graph by choosing appropriate pairs of Steiner points (e.g., choosing every pair of Steiner points that lies on the boundary of the same face) as edges, and compute a shortest path in this graph using Dijkstra's algorithm. Their empirical results show that on most data sets tested their algorithms perform well even if only $O(n)$ Steiner points are placed on P. However, in order to ensure the length of the paths is at most $(1+\varepsilon)$ times that of the shortest path in the worst case, these algorithms have to place $\Omega(n^2/\varepsilon)$ Steiner points even for convex surfaces.

Another disadvantage of these algorithms is that they place $\Omega(1)$ points for each edge of P, so it constructs a graph with $\Omega(n)$ nodes. A natural question is whether the size of the graph has to be $\Omega(n)$, or one can construct a smaller graph by approximating P with another surface Q that is close to P and constructing a path-graph on Q. Although for any given $\varepsilon > 0$, one can easily construct a terrain P so that any ε-approximation of Q also has $\Omega(n)$ vertices, the vast literature on terrain approximation [13] indicates that in practice an ε-approximation of a terrain is much smaller. In fact, if P is a convex surface, then a result by Dudley [9] shows that we can compute another convex polyhedral set Q with $O(1/\varepsilon^{3/2})$ vertices so that P lies in the interior of Q and so that the Hausdorff distance between P and Q is $\varepsilon \cdot \operatorname{diam} P$. Using this and other ideas, Agarwal *et al.* [3] developed an algorithm that constructs in $O(n\log(1/\varepsilon))$ time a convex polytope with $O(1/\varepsilon^{3/2})$ vertices such that $s,t \in Q$, $P \subseteq Q$ and $d_Q(s,t) \leq (1+\varepsilon)d_P(p,q)$. They compute $\pi_Q(s,t)$ and project it onto P without increasing its length, thereby computing an $(1+\varepsilon)$-approximate shortest path in $O(n\log(1/\varepsilon) + 1/\varepsilon^3)$ time. Har-Peled [11] extended their algorithm to answer two-point shortest-path queries, where each query takes $O((\log n)/\varepsilon^{1.5} + 1/\varepsilon^3)$ time. Since these algorithms uses the Chan-Han algorithm [6] as a subroutine, they are not practical.

In this paper, we present a simple algorithm for computing a $(1+\varepsilon)$-approximate shortest path on a convex polytope and we report a fast, robust implementation of the algorithm. It constructs in $O(n/\sqrt{\varepsilon}+1/\varepsilon^4)$ time a graph with $O(1/\varepsilon^{5/2})$ vertices and $O(1/\varepsilon^4)$ edges and computes a shortest path in this graph, and then projects the path on the polytope in time $O(n/\varepsilon)$. Since we are focusing on developing a simple algorithm and its implementation and, as we will show below, small values of $1/\varepsilon$ and a coarser grid work on most examples, we have not made any attempt to improve the running time as a function of ε. Although the analysis and the current implementation of the algorithm work only for convex polytopes, many of the ideas extend to arbitrary polyhedral surfaces. Moreover, convex polytopes are good testbeds because, on one hand, they are somewhat easier to handle and, on the other hand, the numerical problems are in fact harder since the shortest path always passes through the interior of edges and can have a large folding angle, see Table 1.

Our algorithm also provides the first robust implementation of the Hershberger-Suri algorithm. Actually, it integrates the algorithms by Hershberger and Suri [14] and Agarwal *et al.* [3] with the graph based approaches discussed above. Roughly speaking, the graph-based approaches work well at the global level, for guiding toward a good path, while the geometric approaches (e.g., Hershberger-Suri approach) work better locally. By combining the two approaches we retain the advantages of the both approaches.

We add Steiner points only on some of the edges of P and construct a sparse graph G_P, compute a shortest path π in G_P, and project π onto P by "unfolding" each edge of π on P (using the same approach as in [14]). Unlike the previous approaches, we do not place Steiner points directly on P. Instead we first place Steiner points in the vicinity of P and then snap them to P. This allows us to keep the size of G_P small — actually, independent of the size of P. Unfolding a path along k edges on P, corresponding to a graph edge, involves performing a sequence of k three-dimensional linear transforms. In order to handle the numerical problems, a robust implementation of this step requires several clever ideas because the floating-point arithmetic would generate too much error, and the exact arithmetic would explode the bit complexity. We therefore use a hybrid approach.

After projecting π onto P, we apply a heuristic that improves the quality of the shortest path. Lanthier *et al.* [19] also apply a heuristic to improve the quality

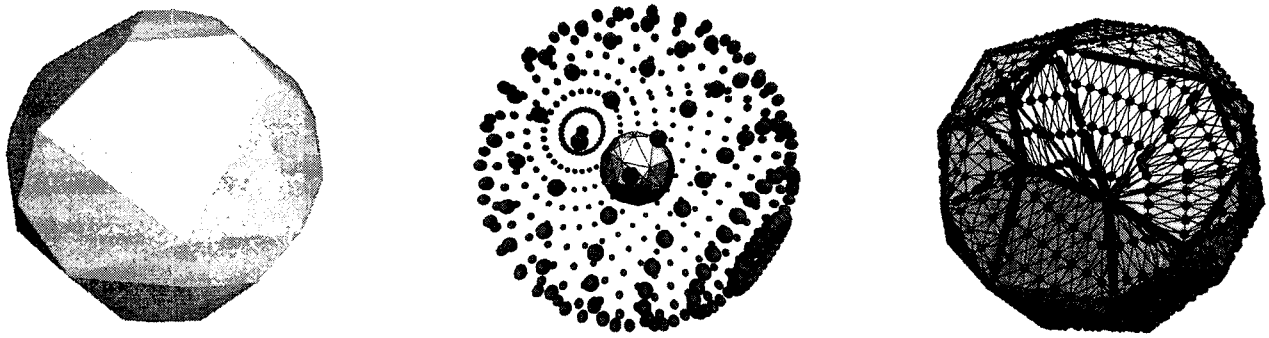


Figure 1: Construct of G_P: (i) The polytope P, (ii) The point set Z_1 on the sphere, and (iii) the projected point-set with the corresponding graph G_P.

of the computed path, but their approach modifies the path only locally and is therefore not as effective as ours. Our implementation results suggest that the heuristics used in the post-processing step affects the quality of the path much more than the number of Steiner points placed on the polytope. In fact, on some polytopes, simply using the Hershberger-Suri algorithm followed by our heuristic works almost as well as our overall algorithm. Nevertheless, there are polytopes on which only the combination of our new algorithm in conjunction with the heuristic is able to yield a good approximation.

This paper is organized as follows. In Section 2 we describe the overall algorithm, provide details of some of the steps, and sketch the proof of the correctness. Section 3 describes the heuristic for improving the quality of the path. Section 4 discusses implementation details, additional techniques that we used to expedite the algorithm. We discuss the experimental results in Section 5, and then conclude in Section 6 by discussing some of the future work.

2 The Algorithm

Let P be a triangulated convex polyhedral surface in $\mathbb{R}^3$ with a total of n vertices. Abusing the notation, we will use P to denote the convex polyhedron bounded by P as well. We will refer to the vertices, edges, and faces of P as the *features* of P. Let s and t be two points on P, and let $\varepsilon > 0$ be a parameter. We first present a brief outline of the algorithm and then discuss various steps in detail.

ALGORITHM: APPROXIMATE-SHORTEST-PATH

1. Compute a value Δ such that $d_P(s,t) \leq \Delta \leq 2d_P(s,t)$, using the Hershberger-Suri algorithm.
2. Let $\mathcal{S}$ be a sphere of radius 4Δ centered at s, and let B be the cube of side-length 2Δ centered at s. Compute $Q = B \cap P$.[2]
3. Let $r = \sqrt{\varepsilon}/c_1$, where $c_1 > 1$ is a constant to be chosen later. Draw a grid $\mathcal{G}$ of longitudes and latitudes on $\mathcal{S}$ that are spaced by r radians each. Let Z_1 be the set of $O(1/r^2) = O(1/\varepsilon)$ grid points; see Figure 1 (ii).
4. Place $O(1/\varepsilon^{3/2})$ points on each edge of $\mathcal{G}$ so that the distance between any two points is at most ε^2/c_2 for a constant $c_2 > 1$ whose value will be decided later. Let Z_2 be the resulting set of points. $|Z_2| = O(1/(r^2\varepsilon^{3/2})) = O(1/\varepsilon^{5/2})$.
5. For each point $p \in Z = Z_1 \cup Z_2$, find its closest point $\Phi(p) \in Q$.
6. Construct a weighted graph $G_P = (V_P, E_P)$, where $V_P = \{\Phi(p) \mid p \in Z\} \cup \{s,t\}$. $(\Phi(p), \Phi(q)) \in E_P$ if p and q lie in the same grid cell. The weight of an edge is the Euclidean distance between its endpoints. By construction, $|V_P| = O(1/\varepsilon^{5/2})$ and $|E_P| = O(1/\varepsilon^4)$. See Figure 1 (iii). (The graph shown in the figure is considerably denser than what would be used in practice.)
7. Use Dijkstra's algorithm to compute a shortest path Π_G between s and t in G_P.
8. Embed the path Π_G onto P and shortcut it. Let Π_P denote the resulting path.

Steps 5, 6, and 8 are the only nontrivial steps. After having computed G_P, Π_G can be computed in $O(1/\varepsilon^4)$ time.

Remarks. (i) In Step 6, instead of connecting all pairs of points lying in the same cell of the grid $\mathcal{G}$, we can construct an ε-spanner of these points. This will

[2] Actually, we do not have to compute Q explicitly, but it simplifies the description and the analysis of the algorithm.

reduce the number of edges by a factor of $1/\sqrt{\varepsilon}$, but our implementation results (and also in [19]) indicate that it is not worth the effort.

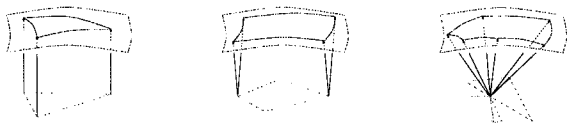
(ii) In practice, we can construct the graph G_P in advance. For a pair of points s and t, we add them in the vertex set, add appropriate edges in G, and construct a shortest path in the graph.

2.1 Projecting grid points

We first describe how to compute $\Phi(p)$ for each point $p \in Z$. Theoretically, we can preprocess Q in $O(n)$ time into a linear-size data structure, using the Dobkin-Kirkpatrick hierarchy [7], so that the closest point on Q of a query point can be computed in $O(\log n)$ time. Using this method, V_P can be computed in $O(n + (\log n)/\varepsilon^{5/2})$ time. But, in practice, the Dobkin-Kirkpatrick hierarchy is rather inefficient and complex. We therefore use a different approach. Let $\mathcal{V}(\mathcal{S})$ be the subdivision of the sphere $\mathcal{S}$ into maximal connected regions so that the nearest point on P for all points within the same region lies in the relative interior of the same vertex, edge, or face of P; $\mathcal{V}(\mathcal{S})$ can be regarded as the restriction of the Voronoi diagram of P on $\mathcal{S}$. Since P is convex, each feature φ of P induces a single connected region $V(\varphi)$ in $\mathcal{V}(\mathcal{S})$. We call a face f of $\mathcal{V}(\mathcal{S})$ a vertex (resp. edge, face) region if the nearest neighbor of the points of f lies on a vertex (resp. edge, face) of P. The vertices of each region of $\mathcal{V}(\mathcal{S})$ can be computed as follows.

- For each face f of P, the vertices of $\mathcal{V}(f)$ are the projections of the three vertices of f onto $\mathcal{S}$ in the direction normal to f; see Figure 2(a).
- For each edge e of P, the vertices of $\mathcal{V}(e)$ are the projections of the endpoints of e onto $\mathcal{S}$ in the direction normals to the two faces adjacent to e; see Figure 2(b). There are two vertices in $\mathcal{V}(e)$ for each endpoint of e.
- For each vertex v of P, the vertices of $\mathcal{V}(v)$ are the projections of v onto $\mathcal{S}$ in the direction normals to all the faces adjacent to v; see Figure 2(c). The number of vertices in $\mathcal{V}(v)$ is equal to the degree of v in P.

$\mathcal{V}(\mathcal{S})$ can be constructed in $O(n)$ time by traversing P in a systematic manner, and we can preprocess it for point-location in $O(n)$ time [10]. But the bit complexity of computing the vertices and edges of $\mathcal{V}(\mathcal{S})$ and of preprocessing it for point location is high. We therefore compute V_P directly without constructing $\mathcal{V}(P)$ explicitly.



(a) (b) (c)

Figure 2: (a) Face Region on S; (b) Edge Region on S; (c) Vertex Region on S.

Let N be the north pole of $\mathcal{S}$. We first compute $\Phi(N)$ in $O(n)$ time. We then traverse each longitude circle C of the grid and compute the nearest neighbors of all points in $C \cap Z$ as follows. Let $N = p_1, p_2, \ldots$ be the sequence of points of $C \cap Z$ sorted in counter-clockwise direction. Suppose we have computed $\Phi(p_{i-1})$ (initially, this is true because we have $\Phi(N)$ at our disposal), and we want to compute $\Phi(p_i)$. Set ξ to the feature of P containing $\Phi(p_{i-1})$. We check whether an edge of $\mathcal{V}(\xi)$ intersects $C[p_{i-1}, p_i]$, portion of C between p_{i-1} and p_i.[3] If the answer is "no," then $\Phi(p_i)$ also lies in ξ, and we compute the point on ξ closest to p_i. If an edge e of $\mathcal{V}(\xi)$ intersects $C[p_{i-1}, p_i]$, then we set ξ to be the other feature of P whose Voronoi region is adjacent to e and repeat the above step.

The time spent in computing the nearest neighbors of points in $C \cap Z$ is $O(1/\varepsilon^2)$ plus the number of edges in the regions of $\mathcal{V}(\mathcal{S})$ that intersect C. In the worst case, the running time is $O(n + 1/\varepsilon^2)$, but in practice much fewer (e.g., $O(\sqrt{n})$) faces will intersect C. We repeat this procedure $O(1/\sqrt{\varepsilon})$ times for all longitudes. We thus obtain the following.

Lemma 2.1 *V_P can be computed in $O(n/\sqrt{\varepsilon} + 1/\varepsilon^{5/2})$ time.*

2.2 Embedding the path

Next we describe how to embed Π_G on P. Before projecting Π_G onto P, we first shortcut it as follows: If two vertices p and q of Π_G lie on the same face of P, we shortcut $\Pi_G[p, q]$ by replacing it with the edge pq. This step does not increase the length of the path, and it can be accomplished in time proportional to the length of the path. Abusing the notation slightly, let Π_G denote the new path as well. If two consecutive vertices p, q of Π_G do not lie on the same face of P, then the corresponding edge (p, q) intersects the

[3] Detecting whether $C[p_{i-1}, p_i]$ intersects $\partial\mathcal{V}(\xi)$ can be done without computing the intersection points of $C \cap \partial\mathcal{V}(\xi)$ explicitly, by checking whether the projection planes induced by $\partial\mathcal{V}(\xi)$ intersects the *segment* $p_{i-1}p_i$.

interior of P. We embed this edge on P, using an approach described by Hershberger and Suri [14], as follows:

Let f_p and f_q be the faces of P containing p and q, respectively, and let H_p and H_q be the planes supporting f_p, f_q, respectively. We compute the *shortest* path $\pi_{pq} = puq$ on the wedge formed by the planes H_p and H_q, where u is the point on $H_p \cap H_q$ defined as follows: We unfold H_q with respect to $H_p \cap H_q$ until it lies on the same plane as H_p. Let q' be the image of q on the unfolded plane. Then u is the intersection point of the segment pq' with the line $H_p \cap H_q$; see Figure 3.

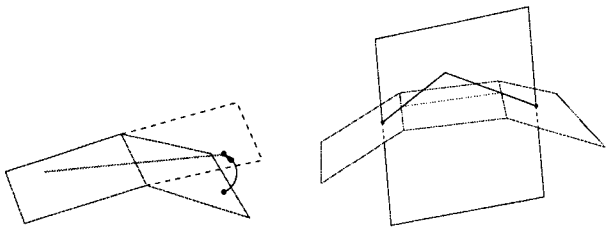


Figure 3: (i) Unfolding a point on a plane onto another plane. (ii) Embedding π_{pq} into P.

After having computed π_{pq}, we embed this path on P. Let H be the plane determined by p, u, q. $H \cap P$ is the boundary of a convex polygon, and let Π_{pq} be the smaller portion of this polygonal boundary; see Figure 3 (ii). We replace π_{pq} with Π_{pq}. Π_{pq} can be computed in time proportional to the number of edges on Π_{pq}, by traversing P from p to q; see [14] for details. We repeat this procedure for all edges of Π_G. The resulting path Π_P lies on P.

Π_G consists of $O(1/\varepsilon)$ edges and the projection of each edge of Π_G crosses each edge of P at most once, therefore it takes $O(n/\varepsilon)$ time in the worst case to compute Π_P.

Theorem 2.2 *Given a convex polyhedral surface P in $\mathbb{R}^3$ with a total of n vertices, two points s, t on P, and a parameter $\varepsilon > 0$, we can compute, in $O(n/\sqrt{\varepsilon} + 1/\varepsilon^4)$ time, a number D, so that $d_P(s,t) \le D \le (1+\varepsilon)d_P(s,t)$. The path realizing D can be computed in $O(n/\varepsilon)$ time.*

Remark. The running time of the embedding step can be improved to $O(n \log(1/\varepsilon))$ by using the algorithm described in [3]. But the existing algorithm works quite well in practice.

2.3 Correctness of the algorithm

In this subsection we prove that the shortest path from s to t in G_P approximates $\pi_P(s,t)$. Intuitively, the grid $\mathcal{G}$ on the sphere $\mathcal{S}$ induces a partition on P into connected regions so that, for any two points p, q within a region, $|pq| \le d_P(p,q) \le (1+\varepsilon/2)|pq|$ and so that the points in Z are located on the boundaries of these regions. We take the shortest path $\pi_P(s,t)$ and shortcut it so that it passes through each region only once and in a connected set. We then snap the resulting path to the projected grid points. We show that the length of the resulting path Π' is at most $(1+\varepsilon/3)d_P(s,t)$ and that the graph G_P contains a path whose length is at most $|\Pi'|$. Finally, we show that the length of the projection of Π' onto P is at most $(1+\varepsilon/2)|\Pi'|$.

We give a more formal proof.

Lemma 2.3 *The length of Π_G, the shortest path in G_P from s to t, is at most $(1+\varepsilon/3)d_P(s,t)$.*

Proof: Let Π be a shortest path from s to t on P. Define $\Gamma = \{\Phi^{-1}(q) \subseteq \mathcal{S} \mid q \in \Pi\}$ to be the pre-image of Π on $\mathcal{S}$. It can be shown using the properties of $\mathcal{V}(\mathcal{S})$ and of a shortest path that $\Gamma \subseteq \mathcal{S}$ is a simple curve. We will first deform the curve Γ so that it visits each grid cell on $\mathcal{S}$ only once and so that it crosses the boundary of a cell only at a point in Z. We will then project this path on P and show that it is a path in G_P and that its length is at most $(1+\varepsilon/3)d_P(s,t)$. We now describe the proof in detail.

If Γ visits a grid cell g in $\mathcal{G}$ more than once, i.e., $\Gamma \cap g$ consists of more than one connected component, let a and b be the first and the last point of $\Gamma \cap g$. We replace $\Gamma[a,b]$ with an arc inside g connecting a to b. We repeat this procedure until Γ visits each grid cell at most once. Let Γ' be the resulting path. Let $\Theta = \langle q_1, q_2, \ldots q_k \rangle$ be the sequence of the intersection points of the grid edges with the relative interior of Γ'. By construction, q_i lies on the original curve Γ as well. For each $1 \le i \le k$, let q_i' be the point in Z closest to q_i and lying on the same grid edge as q_i. We obtain a new curve Γ'' by connecting the initial endpoint of Γ' with q_1', for $1 \le i \le k-1$ connecting q_i' to q_{i+1}' by the great arc that lies inside the grid cell containing q_i', q_{i+1}', and finally connecting q_k' to the final endpoint of Γ'. Set $\psi_i = \Phi(q_i')$ and $\zeta_i = \Phi(q_i)$. Since $q_i \in \Gamma$, $\zeta_i \in \Phi(\Gamma) = \Pi$. Since q_i', q_{i+1}' lie on the boundary of the same grid cell, (ψ_i, ψ_{i+1}) is an edge in G_P. Let Ψ be the path $s = \psi_0, \psi_1, \psi_2, \ldots, \psi_k, \psi_{k+1} = t$ in G_P. The distance between q_i' and q_i is ε^2/c_2, where c_2 is the constant defined in Step 4 of the algorithm. Using the same argument as in [9, 3], it can be shown that the Euclidean distance $|\psi_i \zeta_i| \le 2\varepsilon^2\Delta/c_2$. We can easily prove that $|\Psi| \le (1+\varepsilon/3)d_P(s,t)$ provided that c_2 is chosen sufficiently large, as $k = O(1/\varepsilon)$. Since Π_G is the shortest path in G, the lemma follows. □

Lemma 2.4 $|\Pi_P| \leq (1+\varepsilon)d_P(s,t)$.

Proof: Let (p,q) be an edge of Π_G. Using the same argument as in [3], we can show that the length of Π_{pq}, the projection of the edge (p,q) onto P, is at most $(1+\varepsilon/2)|pq|$. Hence,

$$|\Pi_P| \leq (1+\varepsilon/2)|\Pi_G| \leq (1+\varepsilon)d_P(s,t).$$

□

3 The Post-Processing Step

As the preceding analysis shows, the graph constructed by the algorithm has to be quite large in order to ensure that $|\Pi_P| \leq (1+\varepsilon)d_P(s,t)$. In practice, this considerably slows down the algorithm, and our empirical results show that it is better to construct a sparse graph, compute a path on the polytope using this graph, as described above, and then apply a heuristic to shortcut the path. In this section we describe the heuristic that we use to improve the quality of the computed path. Intuitively, as mentioned before, while the wedge-shortcuting idea of Hershberger and Suri is good locally (i.e., the source and target points are close together), it is too rough to be used globally. On the other end of the spectrum, the graph approach performs well globally, but it performs poorly locally (it forces the path to pass through the nodes of the graph, which are not adapted to the local features of the polytope).

Lanthier et al. [19] suggest to unfold the polytope along the approximated shortest-path and shortcut the computed path within the "sleeve" defined by this sequence of faces (i.e., find the shortest path that passes through the same sequence of edges). However, after this shortcutting is done, the shortest path might pass through a vertex, so one would like to perform this sleeve shortcuting again — on the new sequence of edges (i.e., we slightly deform the path so that it passes slightly on the other side of this vertex). However, since this is a local change in the path, this improvement process would have to be repeated several times to get any notable improvement to its length. Figure 5 shows two examples on which the above local heuristic would have to be applied several times.

We first tried several local heuristics such as shortcutting along an edge or a vertex, but none of them improved the quality of the path as well as the repeated application of the wedge shortcuting method described earlier to embed an edge of Π_G onto P. More precisely, we do the following: Pick two points p,q on the path π, and let H_p, H_q be two supporting planes of P passing through those two points. In our implementation, they are the planes supporting the faces that contain p and q, respectively. We compute the shortest path from p to q on the wedge formed by H_p and H_q and project it onto P as described earlier. If the new path from p to q is shorter than $\Pi[p,q]$, we replace it with the current subpath $\Pi[p,q]$. For efficiency reasons we apply this heuristic in $O(\log n)$ phases. If the path after the $(i-1)$th phase is $v_0, v_1, \ldots$, then in the ith phase, we apply the above step on the pairs $(v_{(j-1)2^i}, v_{j2^i})$, for $j \geq 1$.

Remark. It turns out that, in practice, the projection step is expensive and it does not always improve the quality of the path. Therefore, the running time of this hueristic can be improved by performing the projection step more selectively, i.e., first estimating whether modifying the path between p and q would help and performing the projection only if it is advantageous. However, we focussed on implementing the algorithm and on acheiving the best possible results, so we did not put any effort into optimizing this hueristic.

4 Implementation Details

Our algorithm is implemented in C++ on a Sun Ultra SPARC-5 with 128MB memory, and uses the LEDA (*Library of Efficient Data structures and Algorithms*) library [23] for the exact rational number class and for the representation of 3D geometric primitives. We use OpenGL for the graphical interface. Our implementation source is available at [2].

We modified the algorithm slightly to make it more efficient in handling multiple shortest-path queries on the same polytope with the same value of ε. We construct the entire graph in a preprocessing step. Although this prevented us from using the estimate produced by the Hershberger-Suri algorithm to clip the polytope and the path Π_G is no longer guaranteed to be an $(1+\varepsilon)$-approximate shortest path, we used this approach because, as mentioned earlier, a denser graph does not really improve the quality of the path in practice as long as we perform our post-processing step. For the same reason, we projected only the points in Z_1 on the polytope, i.e., we did not place additional $O(1/\varepsilon^2)$ points on each edge of the grid.

As mentioned in the introduction, we cannot completely rely on the floating point arithmetic. Since the depth of computations is not fixed (i.e., it is a function of the input size) in our case, the roundoff errors accumulated during the computation make the

results useless. The exact arithmetic is also problematic since the number of bits needed to represent the coordinates of path vertices becomes very large. In an earlier implementation, the program died gracefully even on moderate size polytopes, as the exact-arithmetic computations slowed down the program to a standstill.

We use a hybrid method, that is, we use exact arithmetic, but apply various techniques, e.g., rational approximation, whenever possible to keep the representation compact. For example, we observe that all interior vertices of the path lie on the edges of the polytope. Let p be such a point, lying on an edge $e = uv$. Then p can be represented as $\lambda u + (1-\lambda)v$, $\lambda \in [0,1]$. The exact representation of λ may require many bits. We therefore apply a rational-approximation techniques, which, given a parameter M, compute two integers a, b, so that $b \leq M$ and a/b is the best possible approximation to λ. Although efficient theoretical algorithms for this approximation are well known [12, 26], we use the following very simple scheme: We convert λ from an exact representation to a floating-point number (i.e., `double`), and then convert this number back to an exact rational representation. Let λ' be the resulting number, and $p' = \lambda' u + (1-\lambda')v$. Clearly, p' still lies on e very close to p and has a small representation. In fact, we perform this *edge-snapping* for any computed point that lies on an edge.

Another major source of numbers with huge representations is the computation of the cross-product of vectors. However, in almost all computations requiring this operator, any vector with the same orientation as the required vector is good enough. We therefore compute the shortest integer lattice vector that points in the same direction as our desired vector. LEDA represents a vector in homogeneous coordinates (namely, uses 4 coordinates where each coordinate is an exact integer number), therefore the required grid point for a point (a, b, c, d) is $(a/g, b/g, c/g, sign(d))$, where $g = gcd(a, b, c)$ and $sign(d)$ is ± 1, depending on the sign of d.

Even after incorporating the above changes, some minor problems remained with the underlying arithmetic computations, especially in the code for unfolding a path on the polytope. To overcome those problems, we used floating point-arithmetic, with snapping of the resulting point to the plane it is supposed to lie on. In certain cases, computations are carried out using exact arithmetic if the floating-point computations failed (i.e., this is a primitive implementation of arithmetic filtering). The following observation proved to be useful in several cases: If the algorithm computes a point p that need not lie on P, then p can usually be approximated by a point with floating-point coordinates (i.e., with a more compact representation).

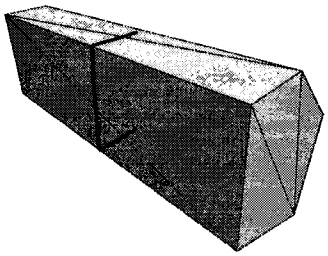


Figure 4: Hershberger-Suri algorithm goes around the polytope in the wrong way. See the first figure in Table 1 for a considerably better shortest-path

Finally, we note that for the projection of points into the polytope (i.e., the computation of Φ), we used a variant of the algorithm described in Section 2.1. Informally, we move from the current projected point to the next, by locally inspecting the polytope. Details are deferred to the full version.

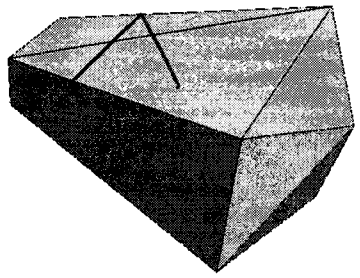
5 Experimental Results

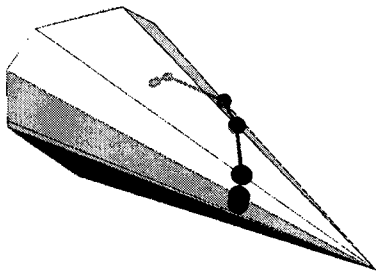
Our implementation results are presented in Table 1. Since we used rational-arithmetic and our main effort had gone into the implementation itself, the running time information, is at best, an indicator of the algorithm performance. The running time can be considerably improved by deploying filtering techniques in the program and by developing a more sophistictaed implementation of the heuristic.

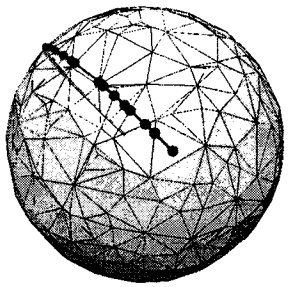
Our program enables one to pick two points on the polytope interactively and then computes the shortest-path between those two points. In Table 1, we demonstrate the results for four different polytopes. The first column specifies the algorithm/parameter that was used. For example, Graph 0.5 means that our algorithm was executed using a spherical grid, with angle of 0.5 radian between grid points. The second column gives the length of the path returned by the algorithm. The next three columns contain the length of the path after our heuristic was applied to the path repeatedly. The last column, contains a picture of the polytope used, with the path computed by the algorithm (with Graph 0.5) after the hueristic was applied to it several times.

The results for the first polytope, in Table 1 indicate that on some examples Hershberger-Suri algorithm returns a considerably longer path even if our heuristic is applied to the resulting path. The reason is that Hershberger-Suri algorithm decides to go

Algorithm	length	Path length after Post-Processing Iteration first	second	third	Shape
Hershberger-Suri	376.686	376.684	376.684	376.684	
seconds	3.090	3.330	4.140	4.140	
Graph 0.05	318.283	312.809	312.573	308.844	
seconds	127.440	2.970	2.600	3.720	
Graph 0.1	386.985	312.850	308.986	308.859	
seconds	38.790	2.130	2.640	3.240	
Graph 0.25	361.870	313.816	308.863	308.849	
seconds	8.540	2.750	3.490	2.850	
Graph 0.5	361.862	308.888	308.884	308.884	
seconds	3.270	3.330	2.990	3.010	
Graph 1	765.758	376.890	376.686	376.684	
seconds	1.490	3.540	5.290	4.700	
Hershberger-Suri	89.369	88.517	88.383	88.377	
seconds	9.600	14.450	20.680	20.160	
Graph 0.05	110.496	101.189	94.277	88.983	
seconds	280.990	6.830	8.140	12.150	
Graph 0.1	111.979	89.138	88.370	88.368	
seconds	93.050	5.410	18.770	19.920	
Graph 0.25	113.907	89.931	88.659	88.487	
seconds	21.590	5.460	20.860	20.270	
Graph 0.5	106.224	88.667	88.391	88.368	
seconds	9.290	12.450	16.430	15.950	
Graph 1	94.843	88.679	88.604	88.531	
seconds	5.250	15.880	20.300	20.950	
Hershberger-Suri	291.671	291.667	291.665	291.663	
seconds	16.020	214.580	884.470	1089.180	
Graph 0.05	301.452	292.366	292.188	292.078	
seconds	168.260	1099.080	1044.440	991.900	
Graph 0.1	301.746	291.827	291.733	291.721	
seconds	58.000	1033.170	1162.050	1048.500	
Graph 0.25	294.564	292.105	291.628	291.414	
seconds	13.930	671.090	1055.520	1078.430	
Graph 0.5	291.788	291.573	291.301	291.200	
seconds	6.190	93.930	837.360	834.790	
Graph 1	291.525	291.175	291.112	291.079	
seconds	3.480	536.760	925.390	884.270	
Hershberger-Suri	116.855	116.471	116.149	115.881	
seconds	26.350	459.680	533.550	541.560	
Graph 0.05	152.918	116.683	115.752	115.344	
seconds	1098.420	41.690	594.840	741.200	
Graph 0.1	150.077	121.559	117.538	116.602	
seconds	416.550	40.930	473.360	593.400	
Graph 0.25	159.124	119.111	116.775	115.790	
seconds	82.860	581.110	713.660	748.130	
Graph 0.5	165.264	119.699	118.779	118.168	
seconds	22.650	478.150	726.920	714.720	
Graph 1	162.843	162.762	162.761	162.761	
seconds	7.240	97.320	132.140	131.480	







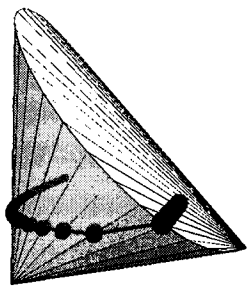
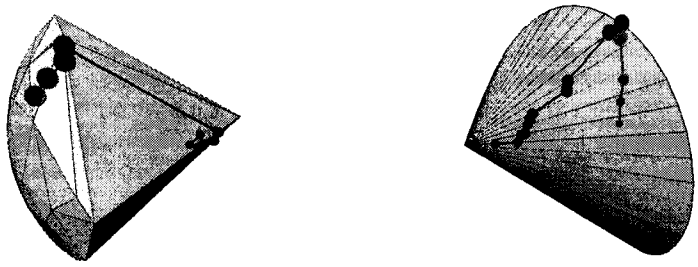


Table 1: A summary of experimental results



(i) (ii)

Figure 5: Path computed by the algorithm before executing the post-processing step: (i) Second example with Graph 0.5. (ii) Fourth example with Graph 0.1.

around the polytope from the wrong side. See Figure 4.

For all the other inputs, Hershberger and Suri algorithm performs surprisingly well. Except in a few cases, all algorithms essentially yield the same results after the shortcutting heuristic has been applied. In particular, it seems that using our algorithm with a very coarse grid (i.e., angle of 0.5 radians) together with our shortcutting heuristic yields very good results, and in practice this seems to be the best combination. Moreover, as the results indicate, it suffices to apply two or three iterations of our heuristic.

The polytopes used in our four examples have 16, 198, 1574, and 96 faces, respectively. The program was also tested and worked correctly on considerably larger convex polytopes with tens of thousands of triangles.

Figure 5 shows the paths computed by our algorithm on two examples (second and fourth in Table 1) before applying the heuristic. These examples amply indicate the advantages of the post-processing stage. Intuitively, the graph based approach is least effective on inputs in which the shortest path crosses many long and skinny faces. Figure 5 (ii) shows why our shortcutting approach is superior to Mata and Mitchell [22] and Lanthier *et al.* [19] approach. In this example, the path computed by the algorithm is "far" from the shortest path, in the sense that if we were to apply the "sleeve" shortcutting described in [19], we would have to apply it several times to get reasonably close to the shortest path. However, since our shortcutting approach is more global, it converges considerably faster.

6 Conclusions

In this paper we had presented a practical algorithm for approximating the shortest path on a convex polytope in three-dimensions. Coupled with a cleanup hueristic the new algorithm performs extremely well in practice.

We are currently working on approximating shortest paths on non-convex polytopes in $\mathbb{R}^3$. This has direct applications to the navigation of tanks, robotics, geographic information systems, medical imaging, low-altitude flight simulation, and water flow analysis. As mentioned in the introduction, algorithms for computing an approximate shortest path on weighted terrains have already been implemented [22, 19]. These algorithms concentrate on placing additional points on the edges of the polytope and using these points in the path graph. Our approach will differ in the graph construction stage: We aim for constructing a graph of *constant size*, and use this for the approximate shortest path computations. We also look for a simple cleanup hueristic to be used in the non-convex case. A preliminary work using those ideas is currently underway [1].

Acknowledgments

The authors thank Kasturi Varadarajan for helpful discussions concerning the problems studied in this paper and related problems, and Alex Karwait for implementing parts of the GUI. The authors also thank the anonymous referees for a number of useful comments.

References

[1] P. K. Agarwal and S. Har-Peled. Approximating shortest-path of terrains. In preparation.

[2] P. K. Agarwal, S. Har-Peled, and M. Karia. Shortest path — demo program. `http://www.cs.duke.edu/~sariel/papers/99/nav/nav.html`, 1999.

[3] P. K. Agarwal, S. Har-Peled, M. Sharir, and K. R. Varadarajan. Approximate shortest paths on a con-

vex polytope in three dimensions. *J. Assoc. Comput. Mach.*, 44:567–584, 1997.

[4] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack. An ϵ-approximation algorithm for weighted shortest paths on polyhedral surfaces. In *Proc. 6th Scand. Workshop Algorithm Theory*, volume 1432 of *Lecture Notes Comput. Sci.*, pages 11–22. Springer-Verlag, 1998.

[5] R. Alexander and N. Rowe. Path planning by optimal-path-map construction for homogeneous-cost two-dimensional regions. In *Proc. IEEE Internat. Conf. Robot. Autom.*, 1990.

[6] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 360–369, 1990.

[7] D. P. Dobkin and D. G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *J. Algorithms*, 6:381–392, 1985.

[8] D. Douglas. Least cost path in geographic information systems. Research note no. 61, Department of Geography, University of Ottawa, Ottawa, Ontario, Aug. 1993.

[9] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approx. Theory*, 10(3):227–236, 1974.

[10] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.

[11] S. Har-Peled. Approximate shortest paths and geodesic diameters on convex polytopes in three dimensions. *Discrete Comput. Geom.*, 21:216–231, 1999.

[12] G. Hardy and E. Wright. *The Theory of Numbers.* Oxford University Press, London, England, 4th edition, 1965.

[13] P. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. Technical report, CMU-CS, 1997. http://www.cs.cmu.edu/~garland/Papers/simp.pdf.

[14] J. Hershberger and S. Suri. Practical methods for approximating shortest paths on a convex polytope in $\Re^3$. *Comput. Geom. Theory Appl.*, 10(1):31–46, 1998.

[15] P. Johansson. On a weighted distance model for injection moulding. Linköping Studies in Science and Technology, Thesis No. 604 LiU-TEK-LIC-1997:05, Division of Applied Mathematics, Linköping University, Linköping, Sweden, Feb. 1997.

[16] S. Kapoor. Efficient computation of geodesic shortest paths. In *Proc. 31rd Annu. ACM Sympos. Theory Comput.*, pages 770–779, 1999.

[17] M. Kindl, M. Shing, and N. Rowe. A stochastic approach to the weighted-region problem, I: The design of the path annealing algorithm. Technical report, Computer Science, U.S. Naval Postgraduate School, Monterey, CA, 1991.

[18] M. Kindl, M. Shing, and N. Rowe. A stochastic approach to the weighted-region problem, II: Performance enhancement techniques and experimental results. Technical report, Computer Science, U.S. Naval Postgraduate School, Monterey, CA, 1991.

[19] M. Lanthier, A. Maheshwari, and J.-R. Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 274–283, 1997.

[20] M. J. Longtin. Cover and concealment in ModSAF. In *Proc. 4th Conf. on Computer Generated Forces and Behavioral Representation*, pages 239–247, 1994.

[21] M. J. Longtin and D. Megherbi. Concealed routes in ModSAF. In *Proc. 5th Conf. on Computer Generated Forces and Behavioral Representation*, pages 305–313, 1995.

[22] C. Mata and J. S. B. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 264–273, 1997.

[23] K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *Cambridge University Press*, 1999.

[24] J. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16:647–668, 1987.

[25] J. S. B. Mitchell. An algorithmic approach to some problems in terrain navigation. In S. S. Iyengar and A. Elfes, editors, *Autonomous Mobile Robots: Perception, Mapping, and Navigation*, pages 408–427. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[26] C. Papadimitriou. Efficient search for rationals. *Info. Process. Lett.*, 8(1):1–4, 1979.

[27] C. H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Inform. Process. Lett.*, 20:259–263, 1985.

[28] R. F. Richbourg, N. C. Rowe, M. J. Zyda, and R. McGhee. Solving global two-dimensional routing problems using Snell's law. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1631–1636, 1987.

[29] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, 15:193–215, 1986.

[30] K. R. Varadarajan and P. K. Agarwal. Approximating shortest paths on an nonconvex polyhedron. *SIAM J. Computing*, to appear.

[31] W. Warntz. Transportation, social physics, and the law of refraction. *The Professional Geographer*, 9(4):2–7, 1957.