Approximating Shortest Paths on a Nonconvex Polyhedron*

Kasturi R. Varadarajan† Pankaj K. Agarwal‡

July 18, 1997

Abstract

We present an approximation algorithm that, given a simple, possibly nonconvex polyhedron P with n vertices in $\mathcal{R}^3$, and two points s and t on its surface ∂P, constructs a path on ∂P between s and t whose length is at most $7(1+\varepsilon)\rho$, where ρ is the length of the shortest path between s and t on ∂P, and $\varepsilon > 0$ is an arbitararily small positive constant. The algorithm runs in $O(n^{5/3}\log^{5/3} n)$ time. We also present a slightly faster algorithm that runs in $O(n^{8/5}\log^{8/5} n)$ time and returns a path whose length is at most $15(1+\varepsilon)\rho$.

* Work on this paper has been supported by National Science Foundation Grant CCR-93–01259, by an Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by an NYI award, by matching funds from Xerox Corporation, and by a grant from the U.S.–Israeli Binational Science Foundation.

† Department of Computer Science, Box 90129, Duke University, krv@cs.duke.edu

‡ Department of Computer Science, Box 90129, Duke University, pankaj@cs.duke.edu

1 Introduction

Problem statement. Let P be a simple nonconvex polyhedron with n vertices, and let s and t be two points on the boundary of P, denoted by ∂P. We consider the problem of computing an *approximate shortest path* on ∂P from s to t.

Computing a shortest path on a polyhedral surface is a central problem in numerous areas, including robotics, geographic information systems, medical imaging, low-altitude flight simulation, and water flow analysis. In most of these applications, a simple, efficient algorithm for computing an approximate shortest path is preferable to an expensive algorithm that computes an exact shortest path, since the input surfaces are rather large, efficiency is critical, and the polyhedral surface is just an approximation of the real surface.

Previous results. Motived by these applications, many researchers have studied the problem of computing a shortest path on a polyhedral surface [3, 8, 12, 13, 14, 15, 20, 28, 26, 32]. Sharir and Schorr [32] gave an $O(n^3 \log n)$ algorithm for the case of a convex polyhedron, exploiting the property that a shortest path on a polyhedron "unfolds" into a straight line. Mitchell et al. [26] improved the running time to $O(n^2 \log n)$; their algorithm works for non-convex polyhedra as well. Chen and Han [8] gave another algorithm with an improved running time of $O(n^2)$. Although several heuristics have been proposed (see [3, 12, 13, 15, 20, 28], for example), it is a long-standing and intriguing open problem whether a subquadratic algorithm can be developed even for computing an approximate shortest path. Recently, Hershberger and Suri [19] presented a simple algorithm that computes a 2-approximate path on a *convex* polyhedron in $O(n)$ time.[1] Agarwal et al. [1] presented an algorithm that computes a $(1+\varepsilon)$-approximate path on a convex polyhedron in $O(n \log 1/\varepsilon + 1/\varepsilon^3)$ time, for any given $\varepsilon > 0$. However, both these algorithms heavily exploit the convexity of the polyhedron and do not extend to nonconvex polyhedra. It was posed as an open problem by Agarwal et al. [1] to find a subquadratic-time algorithm for computing an approximate path between two points on a nonconvex polyhedron.

For the three-dimensional Euclidean shortest-path problem, where we are given a set of polyhedral obstacles and we want to compute the shortest collision-free path between two given points, there are some approximation algorithms due to Papadimitrou [27], Clarkson [10], and Choi et al. [9]. All these algorithms compute a $(1+\varepsilon)$-approximate path, but their running times are superquadratic in n. Recently, Lanthier et al. [21] have considered the problem of computing the *weighted* shortest path problem on polyhedral surfaces. In this problem, each face of the given polyhedron has a weight associated with it, and the cost of traversing a face is the distance travelled on the face times the weight of the face. In this scenario, they give an approximation algorithm for computing a minimum-cost path between two given points on the polyhedral surface. Their algorithm is an adpatation of Papadimitrou's algorithm which we mention above, and computes a $(1+\varepsilon)$-approximate path for the unweighted case in superquadratic time.

Our results. In this paper, we present an algorithm for computing a path on ∂P from s to t whose length is at most $7(1+\varepsilon)\rho$, where ρ is the length of a shortest path on ∂P from s to t. (Throughout the rest of this paper, we will assume that $\varepsilon > 0$ is an arbitararily small, positive, number.) The algorithm runs in time $O(n^{5/3} \log^{5/3} n)$. We also present a second algorithm that runs in $O(n^{8/5} \log^{8/5} n)$ time, and computes a path whose length is at most $15(1+\varepsilon)\rho$. We believe that the running time of the first algorithm can be improved to $O(n^{3/2} \log n)$, but we are faced with some technical difficulties at present. Although our approach falls short of giving a near-linear

[1] We call a path λ-approximate, for $\lambda \geq 1$, if its length is at most λ times that of a shortest path.

time algorithm for this problem, it is an important step because the problem of computing an approximate shortest path in subquadratic time has been open for quite some time now.

The basic idea of our algorithm is rather simple: we choose a parameter r and partition ∂P into $O(n/r)$ "patches," each consisting of at most r faces. For each patch P_i', we carefully choose a set of points on the "boundary" edges of P_i' and construct a graph G_i that approximates a shortest path between any two points lying on the boundary edges of P_i'. We merge these graphs G_i into a single graph G; s and t are guaranteed to be vertices of G. We then compute a shortest path from s to t in G, using Dijkstra's algorithm, and prove that the length of this path is at most $7(1+\varepsilon)d_P(s,t)$. There are two nontrivial steps in the algorithm — how to choose points on the boundary edges and how to construct the graphs G_i's. By using a scheme based on the planar separator theorem [22] to partition ∂P into patches, we ensure that there are only $O(\sqrt{r})$ boundary edges per patch, which allows us to put only a small number of points on the boundary of each patch P_i'. To compute the graph G_i efficiently, we use some novel ideas such as computing a shortest-path map of P_i' from an *edge source*, and generalizing the Voronoi diagram (on the polyhedron) to allow edge sites. These concepts are interesting in their own right. An efficient algorithm for computing the Voronoi diagram from an edge source is not completely trivial, but due to lack of space, we omit the details from this abstract.

The paper is organized as follows. In Section 2, we give basic definitions and properties related to shortest paths, and in Section 3, we describe how to partition ∂P into patches. Section 4 describes the general algorithm. Sections 5 and 6 describe how to choose points on the edges of each patch and how to construct the graphs G_i's, respectively. Due to lack of space, we have included the proofs of many lemmas in an appendix and have omitted many details.

2 Geometric Preliminaries

We assume that the polyhedron P is specified by a set of faces, edges, and vertices, with each edge occurring in exactly two faces, and two faces intersecting either at a common edge, vertex, or not at all. We consider faces to be closed polygons (they include their boundaries) and edges to be closed line segments (they include their endpoints, which are vertices). Without loss of generality, we assume that all faces are triangles (otherwise, we can use any polygon triangulation algorithm to triangulate the faces, thus introducing at most $O(n)$ additional edges), and that s and t are vertices of the polyhedron. If π is a path on ∂P, we let $|\pi|$ denote its length. For any two points $a, b \in \pi$, let $\pi[a,b]$ denote the portion of π between a and b. We refer to $d_P(u,v)$ as the *shortest-path distance* on ∂P between u and v. We denote by ρ the shortest-path distance $d_P(s,t)$. We sometimes refer to a path on the surface of P as simply a path on P. We denote by $d(p,q)$ the Euclidean distance between points p and q in $\mathcal{R}^3$.

We review some relevant properties of shortest paths on polyhedra. A detailed analysis can be found in [26, 32].

Unfolding. Two faces f and f' of P are said to be *edge adjacent* if they share a common edge. A *sequence of edge-adjacent faces* is a list of one or more faces $\mathcal{F} = (f_1, f_2, \ldots, f_{k+1})$ such that the face f_i is edge-adjacent to the face f_{i+1}. Let e_i be the edge shared by the faces f_i and f_{i+1}. We then refer to the (possibly empty) list of edges $\mathcal{E} = (e_1, e_2, \ldots, e_k)$ as an *edge-sequence*. If no face (resp. edge) appears more than once in $\mathcal{F}$ (resp. $\mathcal{E}$), then we call the sequence *simple*.

Each face has a two-dimensional coordinate system associated with it. If faces f and f' are edge-adjacent sharing an edge e, we define the *planar unfolding of f' onto f* as the image of points

of f' when f' is rotated about the line through e until f and f' become coplanar and lie on *opposite* sides of e; see Figure 1. Points in the planar unfolding of f' onto f are written in the coordinate system of f. We *unfold an edge sequence* $\mathcal{E}$ as follows: Rotate f_1 around e_1 until its plane coincides with f_2, rotate f_1 (already unfolded around e_1) and f_2 around e_2 until their plane coincides with that of f_3, and continue in this manner until all faces $f_1, f_2, \ldots, f_k$ lie in the plane of f_{k+1}. If x is a point on face f_i, then the *unfolded image of x along $\mathcal{E}$* is the image of x when we unfold the edge sequence $\mathcal{E}$ (written in the coordinate system of face f_{k+1}).

Let v_i be a point that lies in the interior of edge e_i, for $1 \leq i \leq k$. Let v (resp. v') be a point on face f_1 (resp. f_{k+1}) such that (1) either v (resp. v') is the vertex of f_1 (resp. f_{k+1}) that is opposite edge e_1 (resp. e_k), or (2) v (resp. v') lies in the interior of f_1 (resp. f_{k+1}), or (3) v (resp. v') coincides with v_1 (resp. v_k). (There are nine possibilities allowed here, three each for v and v'.) Then, we say that the path π given by the concatenation of the segments $vv_1, v_1v_2, \ldots, v_{k-1}v_k, v_kv'$ *connects the edge-sequence* $\mathcal{E}$. If π connects edge-sequence $\mathcal{E}$, then the *planar unfolding of π along $\mathcal{E}$* is simply the unfolded image of π along $\mathcal{E}$.

Shortest Paths. A *geodesic* is a path that is locally optimal and cannot, therefore, be shortened by slight perturbations. Note that a shortest path is necessarily a geodesic. We now state a few relevant properties of geodesics and shortest paths.

Lemma 2.1 *The intersection of a shortest path with a face is a (possibly empty) line segment. If π is a geodesic path that connects (two points along) the edge sequence $\mathcal{E}$, then the planar unfolding of π along $\mathcal{E}$ is a straight line segment.*

Unlike in the case of convex polytopes, where a geodesic cannot pass through a vertex of the polytope, a geodesic on a nonconvex polytope can pass through a vertex. The subpath of a geodesic π between any two vertices v and v' of P that are consecutive on π connects some edge-sequence $\mathcal{E}$ (if $\mathcal{E} = \phi$, then v and v' are the endpoints of some edge e which is contained in π). By Lemma 2.1, the subpath from v to v' is completely determined by giving the edge-sequence $\mathcal{E}$. Thus, we have the following characterization of geodesics and shortest paths.

Lemma 2.2 *We can describe a geodesic path π from s to t by writing it as a list*

$$(v_1 = s, \mathcal{E}_1, v_2, \mathcal{E}_2, v_3, \ldots, v_k, \mathcal{E}_k, v_{k+1} = t),$$

where $v_1, \ldots, v_k$ are the vertices (in order) through which π passes, and each $\mathcal{E}_i$ is the (possibly empty) edge sequence which $\pi[v_i, v_{i+1}]$ connects. The planar unfolding of $\pi[v_i, v_{i+1}]$ along $\mathcal{E}_i$ is a straight line segment. If π is a shortest path, no edge of P appears in more than one edge sequence $\mathcal{E}_i$, and each edge sequence $\mathcal{E}_i$ is simple.

3 Partitioning the Polyhedral Surface

We can associate with the polyhedron P a graph G_P, the dual graph of the 1-skeleton of P, defined as follows. (We use 'nodes' and 'arcs' in the context of graphs and use 'vertices' and 'edges' when speaking of terrains.) For each face f of P, there is a node n_f in G_P. There is an arc between nodes n_f and $n_{f'}$ in G_P if the corresponding faces f and f' of the polyhedron are edge-adjacent. Since P is simple, its genus is zero, and G_P is planar. Moreover, since each face of P is a triangle, each node of G_P has degree at most three.

Our algorithm exploits a scheme for partitioning any planar graph $G = (V, E)$ that was developed by Frederickson [16]. Let $(V_1, V_2, \ldots, V_k)$ be a covering of the node set of V, that is, $V_i \subseteq V$ and $\bigcup_i V_i = V$. We refer to each V_i as a *region*. A node is *interior* to a region if it is adjacent only to nodes in that region, while a *shared* node is present in at least two regions. For a given parameter r, an *r-partition* of G is a covering of the node set V by $\Theta(|V|/r)$ regions, such that (1) any node is either a shared node or it is an interior node of some region, and (2) each region contains at most r nodes and $O(\sqrt{r})$ shared nodes. Based on the separator theorem of Lipton and Tarjan [22], Frederickson showed that r-partitions exist and can be constructed in $O(|V| \log |V|)$ time.

An r-partition of G_P induces a subdivision of the polyhedral surface ∂P, which we describe below. A *polyhedral patch* of ∂P (or simply, a patch of ∂P) is just the portion of ∂P comprised of a subset of the faces of P. Given an r-partition of G_P, we designate any face that corresponds to a shared node of G_P as a *buffer face*, and associate a polyhedral patch with the faces corresponding to the interior nodes of each region of the r-partition. (That is, each shared node defines one buffer face, and each region defines one polyhedral patch.) Thus, an r-partition of G_P induces a partition ∂P into $O(n/r)$ polyhedral patches, with each patch containing at most r faces, and $O(n/\sqrt{r})$ buffer faces; abusing terminology slightly, we refer to this partition of ∂P as an *r-partition* of ∂P. An r-partition of ∂P can be computed in $O(n \log n)$ time.

An edge incident to a buffer face is called a *frontier* edge. It is easily seen that a r-partition satisfies the following properties:

1. If face f in a patch is edge-adjacent to a face f' along an edge e, then either f' belongs to the same patch as f, or f' is a buffer face in which case e is a frontier edge.

2. In any patch, there are only $O(\sqrt{r})$ faces that have a frontier edge incident on them.

Since each face of P is a triangle, any patch has only a total of $O(\sqrt{r})$ frontier edges incident to its faces. These are referred to as the *frontier edges* of the patch. Note that the frontier edges of a polyhedral patch constitute its boundary, so we can also refer to them as the *boundary* edges of the patch. The total number of frontier edges is at most $cn/\sqrt{r}$, for some constant $c > 0$. Let

$$\alpha = cn/\sqrt{r}$$

The following lemma, whose proof is included in the appendix, describes the main application of the above partitioning scheme, but we first need a definition.

Definition 3.1 Given a frontier edge e in an r-partition of ∂P, a collection of points $\phi(e)$ on e is a *ε-portal set on e* if the following holds: For any point x on e that lies on the shortest path $\pi_P(s, t)$, there exists $p \in \phi(e)$ whose distance from x is at most $(\varepsilon/2\alpha) \times \rho$. Each point of a ε-portal set on e is called a portal on e. With respect to an r-partition, a path is *legal* if it lies completely within a single patch (resp. buffer face), and each of its two endpoints is a portal on a frontier edge of the patch (resp. buffer face).

Lemma 3.2 *Given an r-partition of the polyhedron P and a ε-portal set on each frontier edge of the r-partition, there is a path π^* between s and t such that (1) its length is at most $(1+\varepsilon)d_P(s, t)$, and (2) it is a concatentaion of legal paths.*

4 The Algorithm

In this section, we present our algorithm for computing a path between s and t on ∂P whose length is at most $7(1+\varepsilon)d_P(s,t)$. For the sake of clarity, we first present an algorithm that approximates the shortest-path distance, i.e., it returns a quantity that is at least $d_P(s,t)$ and at most $7(1+\varepsilon)d_P(s,t)$. Subsequently, we modify this algorithm so that it also returns an approximate shortest path.

4.1 Approximating the shortest path distance

In order to describe the algorithm, we need to define a β-approximator, which is similar to the well-studied notion of a spanner (see [4, 30]) with Steiner points.

Definition 4.1 Let P' be a polyhedral patch, and S be a set of points on P'. A *β-approximator* for S on P' is a weighted graph $G = (S \cup S', E)$, where S' is a set of additional points on P', that has the following properties: (1) If there is an arc $(u,v) \in E$ with weight l, there is a path of length l between u and v on P'. (2) The length of the shortest path in G between u and v, denoted $d_G(u,v)$, is at most $\beta d_{P'}(u,v)$.

Algorithm: Approximating the Shortest-Path Distance

1. We compute an r-partition of ∂P, where $r = n^{1/3}\log^{1/3} n$. Let $P'_1, \ldots, P'_{k_1}$, where $k_1 = O(n/\sqrt{r})$, denote the resulting buffer faces, and $P'_{k_1+1}, \ldots, P'_{k_1+k_2}$, where $k_2 = O(n/r)$, denote the resulting patches, and B, where $|B| \leq \alpha = O(n/\sqrt{r})$, denote the set of resulting frontier edges. Recall that for $1 \leq i \leq k_1$, each buffer face P'_i has three frontier edges incident to it, and for $k_1+1 \leq i \leq k_1+k_2$, each patch P'_i has at most $O(\sqrt{r})$ frontier edges incident on its boundary.

2. On each $e \in B$, we compute a $\varepsilon/2$-portal set of size $O((\alpha/\varepsilon)\lg n) = O((n/\sqrt{r})\log n)$, using the algorithm described in Section 5.

3. Let V denote the set of vertices of P, and M denote the set of all the portals introduced above. Let $V_i = V \cap P'_i$ be the set of vertices of P'_i, and $M_i = M \cap P'_i$ be the set of portals on the frontier edges of P'_i. Let $V_i^B \subseteq V_i$ denote those vertices that are incident on some frontier edge of P'_i. Let $|V_i^B| = n_i$ and $|M_i| = m_i$. Let $\delta = 7(1+\varepsilon/3)$. We compute a δ-approximator $G_i = (V_i \cup M_i \cup M'_i, E_i)$ for $V_i^B \cup M_i$ on P'_i. (The graph G_i δ-approximates legal paths on P_i.)

4. We compute the graph $G = (N, E)$, where $N = \bigcup_i (V_i \cup M_i \cup M'_i)$, and $E = \bigcup_i E_i$. Using Dijkstra's algorithm [11], we compute the shortest path π_G between s and t in G, and return its length $|\pi_G|$ as our estimate of ρ.

Theorem 4.2 *The above algorithm computes, in $O(n^{5/3}\log^{5/3} n)$ time, a quantity between ρ and $7(1+\varepsilon)\rho$.*

Proof: Assuming the correctness of the various sub-procedures, it follows from Lemma 3.2 that the algorithm returns an estimate $|\pi_G|$ such that $\rho \leq |\pi_G| \leq 7(1+\varepsilon)\rho$.

We now analyze the running time of the algorithm. In Step 1 of the algorithm, we can compute an r-partition in $O(n \log n)$ time. The number of frontier edges in the r-partition is $\alpha = O(n/\sqrt{r})$.

In Step 2, a ε-portal set of size $O((n/\sqrt{r}) \log n)$ for each frontier edge can be computed in $O((n/\sqrt{r}) \log n)$ time. Over all frontier edges, this results in a running time of $O((n^2/r) \log n)$ for step 2 and a set M of $O((n^2/r) \log n)$ portals.

To analyze Step 3, we consider the buffer faces and patches separately. For each buffer face P'_i, $1 \leq i \leq k_1$, we compute a δ-approximator G_i using Clarkson's algorithm [10]; it computes a graph $G_i = (V_i^B \cup M_i, E_i)$, with $|E_i| = O((n_i + m_i)/\delta)$, in $O((n_i + m_i) \log(n_i + m_i))$ time. A buffer face P'_i has at most three vertices and frontier edges incident on it, so $n_i \leq 3$ and $m_i = O((n/\sqrt{r}) \log n)$. Hence, the running time for computing G_i using Clarkson's algorithm is $O((n/\sqrt{r}) \log^2 n)$. The time taken in Step 3 over all buffer faces is thus $O((n^2/r) \log^2 n)$.

For a patch P'_i, $k_1 + 1 \leq i \leq k_1 + k_2$, we use the algorithm described in Section 6.2 to construct a 13-approximator for $V_i \cup M_i$ on P'_i. The algorithm constructs a graph $G_i = (V_i \cup M_i \cup M'_i, E_i)$ with at most $O(m_i)$ steiner points M'_i and $O(r^2 + m_i)$ edges. Furthermore, the algorithm takes $O(r^3 \log r + m_i \log r + m_i \log m_i)$ time to construct the graph. A patch P'_i has at most $|V_i| = O(r)$ vertices. Since P'_i has $O(\sqrt{r})$ frontier edges on its boundary and each edge has $O((n/\sqrt{r}) \log n)$ portals, $m_i = O(n \log n)$. The running time for computing G_i is then $O(r^3 \log r + n \log^2 n)$, and the running time for Step 3 summed over all patches is $O(nr^2 \log r + (n^2/r) \log^2 n)$. By choosing $r = n^{1/3} \log^{1/3} n$, the running time of Step 3 is $O(n^{5/3} \log^{5/3} n)$. It can be shown that the running time of step 4 is also $O(n^{5/3} \log^{5/3} n)$, which completes the proof of the theorem. □

Choosing $r = n^{2/5} \log^{2/5} n$ in step 1, $\delta = 15$ in step 3, and using the algorithm of Section 6.3 in step 3 to construct a δ-approximator, we obtain the following result.

Theorem 4.3 *Given a simple, nonconvex polyhedron P, with n vertices, and two points $s, t \in \partial P$, we can compute in time $O(n^{8/5} \log^{8/5} n)$ a quantity between $d_P(s,t)$ and $15(1+\varepsilon)d_P(s,t)$.*

4.2 Computing an approximate shortest path

We now describe the modifications to the algorithm presented above that will allow us to compute an approximate-shortest path from s to t. Let G_i be the δ-approximator, computed in Step 3, for $V_i \cup M_i$ on P'_i. We want to be able to answer the following *path query* for an arc in G_i: given $(p, p') \in E_i$, compute a path on P'_i between p and p' whose length is at most the weight of (p, p'). If P'_i is a buffer face, we can answer such a query by simply taking the line segment joining p and p'. For a patch P'_i, we augment G_i with a data structure that can answer a path query in $O(r)$ time. It will be evident in Section 6 that such a data structure can be computed within the time bound for computing G_i.

Let π_G be the shortest path in G that is computed by Step 4 of the algorithm. We can compute a path π in ∂P between s and t by simply doing a path query for each arc traversed by π_G, and concatenating the resulting paths. This could be expensive if π_G traverses too many arcs of G. However, as described below, we really need to do path queries for only some arcs in π_G. For a face f of P, if p_1 and p_2 are the first and last nodes in f that appear in π_G, we can bypass the arcs in π_G between p_1 and p_2 by including in π the line segment joining p_1 and p_2. If we apply this trick to every face, we will be left with only $O(n)$ path queries to perform, which will require a total of $O(nr)$ time.

Choosing the appropriate value of r, we get the main result of this paper.

Theorem 4.4 *Given a simple, nonconvex polyhedron P, with n vertices, and two points $s, t \in \partial P$, we can compute, in $O(n^{5/3} \log^{5/3} n)$ (resp. $O(n^{8/5} \log^{8/5} n)$) time, a path on ∂P between s and t whose length is at most $7(1+\varepsilon)d_P(s,t)$ (resp. $15(1+\varepsilon)d_P(s,t)$).*

5 Computing Portal Sets

In this section, we present an algorithm that, given an edge e of P (and an r-partition of ∂P), computes a ε-portal set $\phi(e)$ on e. We first describe a scheme for computing a crude estimate ρ^* of the shortest path distance ρ such that $\rho^* \leq \rho \leq O(n)\rho^*$. The estimate ρ^* will prove useful in constructing portal sets. Let $C(s,\omega)$ be a cube of side ω centered at s, and let $P(\omega)$ be the portion of ∂P lying within $C(s,\omega)$, i.e., $P(\omega) = \partial P \cap C(s,\omega)$. Note that $\partial P = P(\infty)$. For a face f of P, let $f_\omega = f \cap C(s,\omega)$. Since f is a triangle, f_ω is convex. This implies that $P(\omega)$ consists of $O(n)$ faces. We let ρ^* be the smallest value of ω for which s and t are connected by a path lying completely in $P(\omega)$.

The proof of the following lemma is included in the appendix.

Lemma 5.1 *(1) $\rho^* \leq \rho \leq c_1 n \rho^*$, for some constant c_1, and (2) the estimate ρ^* can be computed in $O(n \log n)$ time.*

Using lemma 5.1, we can show the following.

Lemma 5.2 *Let B be the set of frontier edges in an r-partition of ∂P, where $|B| \leq \alpha$. Given any edge $e \in B$, we can compute a ε-portal set $\phi(e)$ of size $O((\alpha/\varepsilon)\log n)$ in time $O((\alpha/\varepsilon)\log n)$.*

Proof: We use the estimate ρ^* to compute $\phi(e)$ as follows. For $1 \leq i \leq \lceil \lg c_1 n \rceil$, we add $O(\alpha/\varepsilon)$ portals so that they subdivide $e \cap C(s, \rho^* 2^i)$ into equal-sized intervals of length $(\varepsilon/2\alpha)\rho^* 2^i$.

Clearly, the total number of portals added is $O((\alpha/\varepsilon)\lg n)$. To see that $\phi(e)$ constructed above is a ε-portal set, let x be any point on $\pi_P(s,t) \cap e$. Since $\rho \leq c_1 n \rho^*$, $x \in C(s, \rho^* 2^i)$ for some i between 1 and $\lceil \lg c_1 n \rceil$. Let i_x be the smallest such i. If $i_x = 1$, then among the portals that were added by the procedure to $e \cap C(s, 2\rho^*)$ when $i = 1$, there is some portal p such that $d(p,x) \leq (\varepsilon/2\alpha)\rho^* \leq (\varepsilon/2\alpha)\rho$. If $i_x > 1$, then x does not lie in $C(s, \rho^* 2^{i_x - 1})$, so $d(s,x) \geq \rho^* 2^{i_x - 1}$. Among the portals that were added by the procedure to $e \cap C(s, 2^{i_x}\rho^*)$ when $i = i_x$, there is a portal p such that $d(p,x) \leq (\varepsilon/2\alpha)2^{i_x - 1}\rho^* \leq (\varepsilon/2\alpha)d(s,x) \leq (\varepsilon/2\alpha)\rho$. □

6 Approximating Legal Paths

Let P' be a polyhedral patch with r (triangular) faces, V be the set of its vertices, and B the set of its boundary edges. We assume that $|B| = O(\sqrt{r})$. Let $V^B \subseteq V$ denote those vertices of P' that are incident to some boundary edge. Let M be a set of m points in the interior of edges in B. In this section, we describe algorithms for computing a β-approximator for $V^B \cup M$ on P'. That is, we construct a weighted graph $G = (V \cup M \cup M', E)$, where M' is a set of additional points on P', that has the following properties: (1) If there is an arc $(u,v) \in E$ with weight l, there is a path of length l between u and v on P'. (2) The length of the shortest path in G between u and v, denoted $d_G(u,v)$, is at most $\beta d_{P'}(u,v)$. We first introduce two kinds of Voronoi diagrams on P' that are used by our algorithm.

6.1 Voronoi Partitions

The sites of the first Voronoi diagram are the vertices V of P'. For each boundary edge $e \in B$, we define a partition of $V^1(e)$ into Voronoi cells: a point $x \in e$ belongs to the Voronoi cell for a given site $q' \in V$ if $d_{P'}(q', x) = \min_{q \in V} d_{P'}(q, x)$. $V^1(e)$ is just the restriciton to e of the standard (geodesic) Voronoi diagram (on terrains) that is discussed in Mount's paper [24]. As a consequence of his algorithm, we have the following lemma.

Lemma 6.1 *We can compute the partition $V^1(e)$, for each boundary edge $e \in B$, in a total of $O(r^2 \log r)$ time. Moreover, we can compute, within the same time bound, an associated data structure that will allow us to perform the following query in $O(\log r)$ time: given a point $x \in e$ on a boundary edge, compute the site q containing x in its Voronoi cell in $V^1(e)$, and the distance $d_{P'}(q, x)$. We can also compute a path $\pi(q, x)$ whose length is $d_{P'}(q, x)$ in $O(\log r + k)$ time, where $k \leq r$ is the number of faces of P' traversed by $\pi(q, x)$.*

For the second Voronoi diagram, the sites are the vertices V (vertex-sites) and the boundary edges B (edge-sites). Let Q denote this collection. An edge site is regarded as open, that is, without its endpoints. In allowing boundary edges to be sites of the Voronoi diagram, we depart from the work of Mount [24], which allows only point sites. As we shall see, allowing edge-sites is a useful idea in constructing β-approximators.

We define the distance $d_{P'}(e', x)$ of a point $x \in P'$ from a boundary edge e' to be the infimum $\inf_{o \in e'} d_{P'}(o, x)$. We are now ready to define the Voronoi partition $V^2(e)$ on a boundary edge e. The partition of $V^2(e)$ into Voronoi cells is defined as follows: a point $x \in e$ belongs to the Voronoi cell for a given site $q' \in Q \setminus \{e\}$ if $d_{P'}(q', x) = \min_{q \in Q \setminus \{e\}} d_{P'}(q, x)$; i.e., we take the Voronoi diagram with respect to all sites of Q except e. We will show the following result in the full version of the paper.

Lemma 6.2 *We can compute the partition $V^2(e)$, for a boundary edge $e \in B$, in $O(r^2 \log r)$ time. Furthermore, we can compute, within the same time bound, an associated data structure that will allow us to perform the following query. Given a point $x \in e$ on a boundary edge, we can compute the site q containing x in its Voronoi cell in $V^2(e)$, the distance $d_{P'}(q, x)$, and a point $o \in q$ such that $d_{P'}(o, x) = d_{P'}(q, x)$ in $O(\log r)$ time. We can also compute the path $\pi(o, x)$ whose length is $d_{P'}(o, x)$ in $O(\log r + k)$ time, where k is the number of faces traversed by $\pi(o, x)$.*

The following lemma, which states the most useful property of $V^2(e)$, plays a crucial role in the construction of β-approximators. Its proof is given in the appendix.

Lemma 6.3 *Let $e, e' \in B$ be two distinct boundary edges of P', and let $x \in e$ and $y \in e'$. Let $\pi(x, y)$ be a shortest path of length l between x and y. If $d_{P'}(x, v) > 3l$ for every vertex $v \in V$, then x belongs only to the Voronoi cell of e' in $V^2(e)$, and $d_{P'}(e', x) \leq l$.*

6.2 Computing a $7(1+\varepsilon)$-approximator

We now present our algorithm for computing a $7(1+\varepsilon)$-approximator $G = (V \cup M \cup M', E)$ for $V^B \cup M$ on P'; here M' is a set of at most m new points on P'. For the sake of simplicity, we describe an algorithm for computing a 13-approximator, and then mention how to obtain a $7(1+\varepsilon)$-approximator.

Algorithm: 13-APPROXIMATOR

1. We first compute a 1-approximator for V on P'. That is, for each pair $u, v \in V$, we compute the shortest path distance $d_{P'}(u, v)$ on P' between u and v, and introduce an arc $(u, v) \in E$ whose weight is $d_{P'}(u, v)$. We can easily compute these distances in $O(r^3 \log r)$ time, by invoking algorithm of Mitchell et al. [26] for each vertex of P'.

2. We compute the Voronoi partition $V^1(e)$ for each boundary edge $e \in B$. For each $p \in M$, we locate the edge e containing p, and a vertex $v \in V$ such that p belongs to the Voronoi cell of the vertex v in $V^1(e)$. We compute the distance $d_{P'}(v, p)$ and introduce an arc (p, v) in G with weight $d_{P'}(v, p)$.

3. For every $e \in B$, we compute the Voronoi partition $V^2(e)$. For each $p \in M$ that lies in e, we locate a site q such that p belongs to the Voronoi cell of q in $V^2(e)$. If q is an edge site, we compute a point $o \in q$ such that $d_{P'}(o, p) = d_{P'}(q, p)$. We introduce in G a new node o, and an arc (p, o) with weight $d_{P'}(x, p)$.

4. Let M' be the set of points that are introduced as nodes in G in step 3. For each edge $e \in B$, we sort the points $(M \cup M' \cup V) \cap e$ along e and add an arc (p, p') of weight $d(p, p')$ if p and p' are consecutive in the sorted order.

Lemma 6.4 *The algorithm runs in $O(r^3 \log r + m \log r + m \log m)$ time, and constructs a graph with $O(r + m)$ nodes and $O(r^2 + m)$ arcs.*

Lemma 6.5 *The graph G constructed by the algorithm is a 13-approximator for $V^B \cup M$ on P'.*

Proof: Clearly, G is a 1-approximator for V (step 1), and for the points in $(V \cup M \cup M') \cap e$ (step 4), for any $e \in B$.

Let π be the shortest path on P' between $p, p' \in V^B \cup M$, and let $|\pi| = l$. We will show that $d_G(p, p') \leq 13l$. Let $w, w' \in V$ be the vertices closest to p and p', respectively. We consider two cases.

Case 1: $d(p, w), d(p', w') \leq 3l$. We claim that there is a vertex $u \in V$ such that $d_G(p, u) = d_{P'}(p, u) \leq 3l$. If $p \in V$, the claim is obviously true. If p is not a vertex, step 2 of the algorithm must have introduced an edge (p, u) to some $u \in V$ that is closest to p. The distance from p to u is at most $3l$, establishing the claim. Similarly, there is a vertex $u' \in V$ such that $d_G(p', u') = d_{P'}(p', u') \leq 3l$. Now, the shortest-path distance $d_{P'}(u, u')$ is at most $d_{P'}(u, p) + d_{P'}(p, p') + d_{P'}(p', u') \leq 7l$.

Since G is a 1-approximator for V, $d_G(u, u') \leq 7l$. Finally, we have

$$d_G(p, p') \leq d_G(p, u) + d_G(u, u') + d_G(u', p') \leq 13l.$$

Case 2: $\max\{d(p, w), d(p', w')\} > 3l$. Assume, without loss of generality, that $d(p, w) > 3l$. In this case, neither p nor p' is a vertex. Let e (resp. e') be the boundary edge containing p (resp. p') in its interior. If e and e' are the same edge, the lemma follows because G is a 1-approximator for points on the same edge.

Let us assume that e and e' are distinct. Then, by Lemma 6.3, p belongs only to the Voronoi cell of e' in $V^2(e)$, and $d_{P'}(e', p) \leq l$. When we processed p in step 3 of the algorithm, we must have introduced a point $o \in e'$ as a node of G, and must have added an arc (o, p) with weight $d_{P'}(e', p)$. Thus, $d_G(o, p), d_{P'}(o, p) \leq l$. By the triangle inequality, $d_{P'}(o, p') \leq d_{P'}(o, p) + d_{P'}(p, p') \leq 2l$.

Since G is a 1-approximator for points on e', $d_G(o, p') \leq 2l$. Finally, we have $d_G(p, p') \leq d_G(p, o) + d_G(o, p') \leq 3l$. □

In case 1 of Lemma 6.5 above, there is a path on P' between p and p' that passes through a vertex of P' and has length at most $7l$. Using a scheme based on a recent result of Har-Peled [17] for constructing approximate shortest path maps on a polyhedron from a single source, we can compute a graph that $7(1+\varepsilon)$-approximates any path between $u, v \in V^B \cup M$ that passes through a vertex of P'. By plugging this construction into the above algorithm, we can construct, within the same time bounds, a $7(1+\varepsilon)$-approximator for $V^B \cup M$ on P'.

6.3 Computing a 15-approximator

We now present a slightly different algorithm that computes a 15-approximator $G = (V \cup M \cup M', E)$ for $V^B \cup M$ on the patch P'. This algorithm, which we call 15-APPROXIMATOR, runs in $O(r^{5/2} \log r + m \log r + r \log r)$ time. Let $e \in B$ be a boundary edge. We define the *distance* of a point $x \in P'$ from e, which we denote by $d_{P'}(e, x)$, to be $\min_{o \in e} d_{P'}(o, x)$. The only difference from the definition used in Section 6.1 is that we now regard e as closed, i.e., e includes its endpoints. If $\pi(o, x)$ is a path from $o \in e$ to x, and its length $|\pi(o, x)|$ equals $d_{P'}(e, x)$, we say that $\pi(o, x)$ is the shortest path from the boundary edge e to x. By suitably modifying the algorithm by Mitchell et al. [26], we can compute shortest paths from a single boundary edge of P' to every vertex of P' in $O(r^2 \log r)$ time.

Algorithm 15-APPROXIMATOR is identical to 13-APPROXIMATOR, except that we replace step 1 by the following procedure. From each $e \in B$, we compute a shortest path to every vertex of V using the $O(r^2 \log r)$ algorithm that was mentioned above. Let $\pi(o, u)$, where $o \in e$ and $u \in V$, be the shortest path from e to u that is computed. We introduce o as a new node in G, and add an arc (o, u) with weight $|\pi(o, u)|$.

Proceeding in a way similar to previous subsection, we can show the following lemma. We have omitted its proof from this abstract.

Lemma 6.6 *We can compute, in time $O(r^{5/2} \log r + m \log m + m \log r)$, a 15-approximator for $V^B \cup M$ on P'.*

References

[1] P. K. Agarwal, S. Har-Peled, M. Sharir, and K. R. Varadarajan, Approximating shortest paths on a convex polytope in three dimensions, submitted to *J. ACM*, 1996.

[2] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[3] V. Akman, *Unobstructed Shortest Paths in Polyhedral Environments*, *Lecture Notes in Computer Science*, vol. 251, Springer Verlag, 1987.

[4] S. Arya, D. M. Mount, and M. Smid, Randomized and deterministic algorithms for geometric spanners of all diameter, *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, 703–712, 1994.

[5] Ta. Asano, Te. Asano, L. Guibas, J. Hershberger, and H. Imai, Visibility of disjoint polygons, *Algorithmica*, 1:49–63, 1986.

[6] C. Bajaj, The algebraic complexity of shortest paths in polyhedral spaces, *Proc. 23rd Allerton Conf. Commun. Control Comput.*, 510–517, 1985.

[7] J. Canny and J. H. Reif, New lower bound techniques for robot motion planning problems, *Proc. 28th Annu. IEEE Symp. Found. Comput. Sci.*, 49–60, 1987.

[8] J. Chen and Y. Han, Shortest paths on a polyhedron, *Proc. 6th Annu. ACM Symp. Comput. Geom.*, 1990.

[9] J. Choi, J. Sellen, and C. H. Yap, Approximate Euclidean shortest path in 3-space, *Proc. 10th Annu. ACM Symp. Comput. Geom.*, 41–48, 1994.

[10] K. L. Clarkson, Approximation algorithms for shortest path motion planning, *Proc. 19th Annu. ACM Symp. Theory Comput.*, 56–65, 1987.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[12] X. Doug, *Finding a Geodesic Path on a 3D Triangulated Surface*, MS Thesis, Missisipi State University, 1995.

[13] R. Franklin and V. Akman, Shortest paths between source and goal points located on/around a convex polyhedron, *Proc. 22nd Allerton Conf. Commun. Control Comput.*, 1984.

[14] R. Franklin and V. Akman, Shortest paths in 3-space, Voronoi diagrams with barriers, and related complexity and algebraic issues, *Proc. NATO Advanced Study Institute on Fundamental Algorithms for Computer Graphics*, Springer-Verlag, 895–917, 1985.

[15] R. Franklin, V. Akman, and C. Verrilli, Voronoi diagrams with barriers and on polyhedra for minimal path planning, *Visual Comput*, 1:133–150, 1985.

[16] G. N. Frederickson, Fast algorithms for shortest paths in planar graphs, with applications, *SIAM J. Comput.*, vol. 16, no. 6, pp. 1004–1022, 1987.

[17] S. Har-Peled, Constructing approximate shortest path maps in three dimensions, *Manuscript*, 1997.

[18] J. Hershberger and S. Suri, Efficient computations of Euclidean shortest paths in the plane, *Proc. 34th Annu. IEEE Symp. Found. Comput. Sci.*, 508–517, 1993.

[19] J. Hershberger and S. Suri, Practical methods for approximating shortest paths on a convex polytope in $\mathcal{R}^3$, *Proc. 6th Annu. ACM-SIAM Symp. Discrete Algo.*, 447–456, 1995.

[20] K. Kant and S. Zucker, Toward efficient planning: The path-veolcity decomposition, *Int. J. Robotics Research*, 5:72–89, 1986.

[21] M. Lanthier, A. Maheshwari, and J. Sack, Approximating weighted shortest paths on polyhedral surfaces, to appear in *Proc. 13th Annu. ACM Symp. Comput. Geom.*, 1997.

[22] R. J. Lipton and R. E. Tarjan, A separator theorem for planar graphs, *SIAM J. Appl. Math.*, 36 (1979), pp. 177–189.

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

[24] D. M. Mount, Voronoi diagrams on the surface of a polyhedron, Tech. Rep. 1496, University of Maryland, 1985.

[25] J. S. B. Mitchell, Shortest paths among obstacles in the plane, *Proc. 9th Annu. ACM Symp. Comput. Geom.*, 308–317, 1993.

[26] J. S. B. Mitchell, D. Mount, and C. H. Papadimitrou, The discrete geodesic problem, *SIAM J. Comput.*, 16:647–668, 1987.

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

[28] B. Porter, S. Mohamad, T. Crossley, Genetic computation of geodesic on three-dimensional curved surfaces, *Proc. 1st Intl. IEE/IEEE Conf. on Genetic Algorithms in Engineering System: Innovations and Applications*, 1995.

[29] J. Reif and J. Storer, Shortest paths in Euclidean space with polyhedral obstacles, J. ACM, 41:1013–1019, 1994.

[30] J. S. Salowe, Constructing multidimensional spanner graphs, *Internat. J. Comput. Geom. Appl.*, 1(2):99–107, 1991.

[31] M. Sharir, On shortest paths amidst convex polyhedra, *SIAM J. Comput.*, 16:561–572, 1987.

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

[33] E. Welzl, Constructing the visibility graph of n line segments in $O(n^2)$ time, *Inform. Process. Letters*, 20:167–171, 1985.

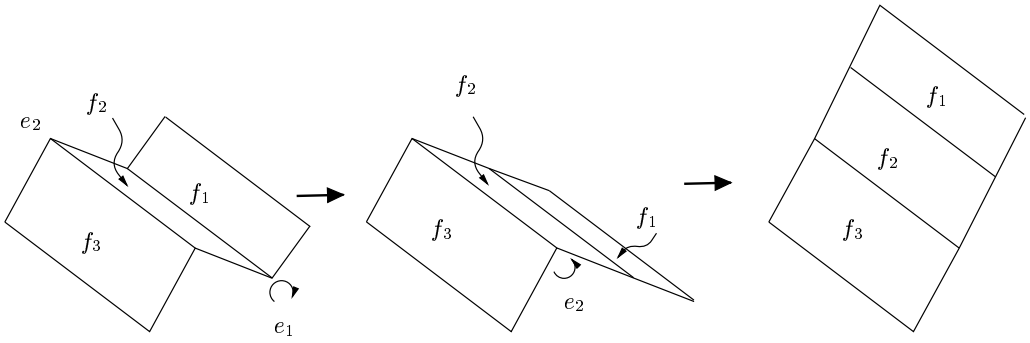

Figure 1: Unfolding an edge sequence.

A Appendix: Proofs of a Few Lemmas

Lemma 3.2 *Given an r-partition of the polyhedron P and a ε-portal set on each frontier edge of the r-partition, there is a path π^* between s and t such that (1) its length is at most $(1+\varepsilon)d_P(s,t)$, and (2) it is a concatentaion of legal paths.*

Proof: Let $\pi = (v_1 = s, \mathcal{E}_1, v_2, \mathcal{E}_2, v_3, \ldots, v_k, \mathcal{E}_k, v_{k+1} = t)$ represent the shortest path between s and t, where $v_1, \ldots, v_k$ are the vertices (in order) through which π passes, and each $\mathcal{E}_i$ is the (possibly empty) edge sequence which $\pi[v_i, v_{i+1}]$ connects. Let e be a frontier edge that occurs in some edge sequence $\mathcal{E}_i$ above. Let x be the (unique) point on e that lies on π, and let p be the portal on e so that $d_P(x,p) = d(x,p) \leq \varepsilon/2 \times d_P(s,t)/\alpha$. We alter π slightly by introducing a loop from x to p and back. This increases the length of the path by at most $\varepsilon d_P(s,t)/\alpha$. If we perform this surgery on every frontier edge that occurs in some edge sequence, we get a path π^* whose length is at most $(1+\varepsilon)d_P(s,t)$. It is clear that π^* is a concatenation of legal paths, since s and t are assumed to be the vertices of P. □

Lemma 5.1 *(1) $\rho^* \leq \rho \leq c_1 n \rho^*$, for some constant c_1, and (2) the estimate ρ^* can be computed in $O(n \log n)$ time.*

Proof: (1) We claim that any shortest path $\pi_P(s,t)$ must intersect the boundary of $C(s,\rho^*)$. Assume for the sake of contradiction that the claim is false. Then, there exists a $\rho' < \rho^*$ such that $\pi_P(s,t)$ lies completely within $C(s,\rho')$. Thus, $\pi_P(s,t)$ connects s and t in $P(\rho')$, contradicting the fact that ρ^* is the smallest value of ω for which s and t are path-connected within $P(\omega)$.

Since $\pi_P(s,t)$ intersects the boundary of $C(s,\rho^*)$, we have $\rho = |\pi_P(s,t)| \geq \rho^*$.

To establish the second inequality, consider the shortest path π' between s and t in $P(\rho^*)$. Since $P(\rho^*)$ consists of at most $O(n)$ faces, and π' intersects each of these faces in a single (possibly empty) line segment, π' can be specified as a concatenation of at most $O(n)$ such line segments. Each such line segment lies completely within $C(s,\rho^*)$, and so has length at most $2\sqrt{3}\rho^*$. It follows that $|\pi'| \leq c_1 n \rho^*$, for some constant c_1. Since π' is also a path on ∂P, $\rho \leq |\pi'| \leq c_1 n \rho^*$.

(2) $P(\omega)$ is a disjoint union of path-connected components. For each face f of P, f_ω is contained in at most one such component. For a path-conencted component C of $P(\omega)$, let F_C be the collection

f of faces of F such that $f_\omega \neq \emptyset$ and $f_\omega \subseteq C$. Note that $C = \bigcup_{f \in F_C} f_\omega$. Observe that the sets F_C partition the set of faces of P that have a non-empty intersection with $C(s, \omega)$.

Let f^s (resp. f^t) be some face of P containing s (resp. t). Then, ρ^* is the smallest value of ω for which (1) $s, t \in C(s, \omega)$ (2) f^s_ω and f^t_ω belong to the same path-connected component of $P(\omega)$. Thus, computing ρ^* boils down essentially to finding the smallest ω when (2) occurs. We sketch below an algorithm to do this.

Our algorithm represents each path-connected component C of $P(\omega)$ implicitly by the set F_C. As ω increases form zero, our representation undergoes changes. Two sets F_{C_i} and F_{C_j} may get merged. A new face f may be added to one of the already existing sets F_C. A set consisting of just a single new face f may be created, if f_ω forms a single path-connected component in $P(\omega)$. Clearly, such changes in our representation occur only when some vertex, edge, or face of P first intersects $C(s, \omega)$. Hence, there are only $O(n)$ critical values of ω at which our representation needs to be updated.

We first compute and sort (in increasing order) these critical values of ω. Starting from $\omega = 0$, we go through them in order, updating our representation of the path-connected components of $P(\omega)$ appropriately. We stop and report the value of ω when f^s and f^t are in the same set in our representation. If we use any data structure for disjoint sets [11] to maintain the representation of the path-connected components, the entire procedure can be implemented in $O(n \log n)$ time. $\square$

Lemma 6.3 *Let $e, e' \in B$ be two distinct boundary edges of P', and let $x \in e$ and $y \in e'$. Let $\pi(x, y)$ be a shortest of length l between x and y. If $d_{P'}(x, v) > 3l$ for every vertex $v \in V$, then x belongs only to the Voronoi cell of e' in $V^2(e)$, and $d_{P'}(e', x) \leq l$.*

We first establish a preliminary claim before proving Lemma 6.3. We refer to any path that originates from x and that has length at most l as a *short* path. Since $d_{P'}(x, v) > 3l$ for every $v \in V$, no short path passes through a vertex. It follows that every short, geodesic path connects some edge sequence beginning with the edge e. Let $\mathcal{E} = e\mathcal{E}_0 e'$ be the edge sequence which $\pi(x, y)$ connects (e and e' are respectively the first and last edges of $\mathcal{E}$).

Claim A.1 *Any short, geodesic path connects some edge sequence which is a prefix of $\mathcal{E}$.*

Proof: Suppose, for a contradiction, that the claim is false. That is, there is a short, geodesic path π' that connects $\mathcal{E}' e_1 e_3$, whereas the edge sequence connected by $\pi(x, y)$ is $\mathcal{E} = \mathcal{E}' e_1 e_2 \mathcal{E}''$.

Since both paths are geodesic, the edges e_1, e_2, and e_3 must be incident on the same (triangular) face, say f. Let v_i be the vertex of f opposite e_i. Let $a_1 \in \pi(x, y) \cap e_1$, $a_2 \in \pi(x, y) \cap e_2$ and $a_3 \in \pi' \cap e_3$. Then,

$$d(a_i, a_j) \leq d_{P'}(a_i, x) + d_{P'}(x, a_j) \leq 2l.$$

In triangle f, the angle at some vertex is at least $\pi/3$. Assume that this vertex is v_1. (The other cases are treated identically). Then,

$$\min\{d(v_1, a_2), d(v_1, a_3)\} \leq d(a_2, a_3) \leq 2l.$$

Therefore,

$$\begin{aligned} d_{P'}(x, v_1) &\leq \min\{d_{P'}(x, a_2) + d(a_2, v_1), d_{P'}(x, a_3) + d(a_3, v_1)\} \\ &\leq l + \min\{d(v_1, a_2), d(v_1, a_3)\} \\ &\leq 3l \end{aligned}$$

This contradicts the assumption that $d_{P'}(x, v) > 3l$ for every $v \in V$. □

We can now prove Lemma 6.3.

Proof of Lemma 6.3: Since $\pi(x, y)$ is a shortest path, the edge sequence $\mathcal{E}$ is simple. Since $\pi(x, y)$ is geodesic, there can be no boundary edge in $\mathcal{E}_0$. As a consequence of the claim, e' is the only edge site in $V^2(e)$ that can be reached by a short, geodesic path. We have already established that no short path can reach a vertex site. Thus, x belongs only to the Voronoi region of e' in $V^2(e)$, and $d_{P'}(e', x) \leq l$.

Lemma 6.4 *The algorithm runs in $O(r^3 \log r + m \log r + m \log m)$ time, and constructs a graph with $O(r + m)$ nodes and $O(r^2 + m)$ arcs.*

Proof: Step 1 of the algorithm runs in $O(r^3 \log r)$ time. In this step, we introduce $O(r^2)$ arcs. In step 2, $V^1(e)$ can be computed for every edge e in a total of $O(r^2 \log r)$ time. The computation involving a single $p \in M$ takes $O(\log r)$ time, so the overall computation for every $p \in M$ takes $O(m \log r)$ time. In step 3, we can compute $V^2(e)$ for a single edge in $O(r^2 \log r)$ time. We can therefore compute $V^2(e)$ for every $e \in B$ in $O(r^{5/2} \log r)$ time. We can perform the computation for each $p \in M$ in a total of $O(m \log r)$ time. We introduce at most m new nodes and arcs in this step. After these steps, we have a total of $O(r + m)$ nodes in G. So we can implement step 4 in $O(r + m \log m)$ time. We introduce only $O(r + m)$ arcs in this step.

Summing up, we have spent $O(r^3 \log r + m \log m + m \log r)$ time, and constructed a graph G with $O(r + m)$ nodes and $O(r^2 + m)$ arcs. □