Approximate Shortest Distances Among Smooth Obstacles in 3D

Christian Scheffer[1] **and Jan Vahrenhold**[2]

1 **Department of Computer Science, Technische Universität Braunschweig, Mühlenpfordtstraße 23, 38106 Braunschweig, Germany**
scheffer@ibr.cs.tu-bs.de

2 **Department of Computer Science, Westfälische Wilhelms-Universität Münster, Einsteinstr. 62, 48149 Münster, Germany**
jan.vahrenhold@uni-muenster.de

Abstract

We consider the classic all-pairs-shortest-paths (APSP) problem in a three-dimensional environment where paths have to avoid a set of smooth obstacles whose surfaces are represented by discrete point sets with n sample points in total. We show that if the point sets represent ε-samples of the underlying surfaces, $(1 \pm \mathcal{O}(\sqrt{\varepsilon}))$-approximations of the distances between all pairs of sample points can be computed in $\mathcal{O}(n^{5/2} \log^2 n)$ time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Geodesic distances, approximation algorithm, epsilon sample

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.60

1 Introduction

Computing shortest distances between pairs of points is one of the classic problems in Computational Geometry. The problem is well-understood in (geometric) graphs and in the Euclidean plane [11]. Computing exact shortest paths among obstacles in three-dimensional Euclidean space, however, was shown to be $\mathcal{NP}$-hard [7]. Subsequently, authors have considered special cases such as exact distances on a convex polyhedron [16] or approximate distances on a general, possible weighted polyhedron [2], see also the survey by Bose et al. [6].

We consider a set of smooth obstacles in $\mathbb{R}^3$ given as an ε-sample, i.e., a point set on the union of the obstacles' boundaries locally dense enough to faithfully capture curvature and folding. As usual, ε is a sampling parameter unknown to the algorithm [4, 14]. In line with previous approaches (see [15] and the references therein), we assume that ε is upper-bounded by a constant $\varepsilon_0 > 0$ which only depends on the algorithm but neither on the input size nor on the curvature or folding of the underlying surface. We obtain the following result:

▶ **Theorem 1.** *There is a global and shape-independent constant $\varepsilon_0 > 0$ such that it holds for $\varepsilon \leq \varepsilon_0$: Given an ε-sample S of a set of smooth obstacles in $\mathbb{R}^3$, we can compute $(1 \pm \mathcal{O}(\sqrt{\varepsilon}))$-approximations of all $\binom{n}{2}$ distances in $\mathcal{O}(n^{5/2} \log^2 n)$ time, where $n := |S|$.*

In general, shortest paths among obstacles alternate between geodesic subpaths on obstacles and straight-line segments in free space. The standard approach to computing such *free-space geodesics* would be to compute both geodesic distances and visibility edges between each pair of points, model these distances by a weighted graph G with vertices corresponding to the sample points on the obstacles, and then to combine the results using an all-pairs shortest path algorithm on G. Due to the complexity of visibility maps in three-dimensional

space, this approach would lead to an at least cubic runtime. We will alleviate this problem by simultaneously restricting the degree of G and locally bounding the length of the edges.

Related Work

A free-space geodesic is modeled by two types of edges in G that correspond to either geodesics on obstacles or straight-line segments in free space. For the geodesics on obstacles, we note that exact shortest path computations on general polyhedra are considered complex and challenging. We refer to recent surveys [3, 6, 11] for a detailed discussion and focus on two approximation algorithms: the best algorithm currently known for weighted polyhedra is the algorithm by Aleksandrov et al. [3], while the algorithm by Scheffer and Vahrenhold [14] works on (unweighted) 2-manifolds in $\mathbb{R}^3$. The efficiency of the algorithm by Aleksandrov et al. depends on the triangles obeying a "fatness" condition. In general, however, the aspect ratio and, hence, the runtime can be arbitrarily large. Both algorithms result in a shortest-path graph of quadratic size. Exploring this structure in combination with a visibility graph (see below) as part of a shortest-path algorithm leads to at least cubic runtime.

The second type of edges of a free-space geodesic corresponds to straight-line segments connecting points on obstacles by crossing the free space; these *bridge edges* are obtained by computing visibility information between points on the obstacles. Despite recent advances in algorithms for "realistic terrains" [12], the complexity of the visibility map of a three-dimensional surface is quadratic in the worst case—see [12] and the references therein. As discussed above, a visibility map of quadratic size leads to a cubic overall running time. A subquadratic complexity currently can be obtained only under standard assumptions about the fatness of the triangles and the (bounded) ratio of shortest and longest edges; in particular the latter assumption is infeasible in the case we are considering.

2 Outline of the Algorithm

To simplify the exposition, we will assume that the obstacles' boundary consists of exactly one surface Γ. Since even in that case straight-line segments, i.e., *bridge edges*, may be needed on a free-space geodesic to bridge cavities in the manifold $\mathcal{F}$ bounded by Γ, it is easy to see that our algorithm easily generalizes to multiple non-intersecting obstacles.

From a high-level perspective, our algorithm proceeds by first constructing a weighted graph $G := \left(S^{sub}, E_{loc} \cup E_{bri}\right)$ on a subset $S^{sub} \subseteq S$ of sample points where the edges in E_{loc}, called *local edges*, represent approximate free-space geodesic distances between points on Γ and the edges in E_{bri} represent straight-line segments avoiding Γ; in either case, the weight of an edge denotes the length of the respective connection. The algorithm then approximates the free-space geodesic distances $L^{\star}_{\Gamma}(s_1, s_2)$ for all $s_1, s_2 \in S^{sub}$ by computing exact all-pairs shortest paths in G and extends these results to compute approximate distances $\mathcal{L}(\cdot, \cdot)$ between all points in S. The algorithm is given below and will be discussed in the following.

As sketched above, the main challenge in obtaining an algorithm with subcubic running time lies in working with a shortest-path graph with bounded or at least sublinear degree. As we will detail below, we can obtain such a graph by first avoiding to compute approximate free-space geodesic distances between *all* pairs of points. Instead, we will compute such distances only for points within a locally bounded distance of each other. These distances will be represented by the set E_{loc} of local edges connecting points in a carefully coarsened subsample $S^{sub} \subseteq S$. We then compute a set E_{bri} of bridge edges such that E_{bri} is a superset of the visibility graph of S^{sub} w.r.t. (the sample points of) Γ. In both cases, we can simultaneously guarantee a sublinear node degree and a good approximation quality.

Algorithm 1 Approximating Geodesic Distances.

```
1: function APX3DGeodesicDistances(S)
2:     ψ(·) ← ControlFunction(S);   ▷ Setup approximation using Lemma 4 [9, 14]...
3:     afs(·) ← APXLocalFeatureSize(S);   ▷ ... and algorithm by Aichholzer et al. [1]
4:     δ ← max_{s∈S} ψ(s)/afs(s);   ▷ Lower-bound local feature size
5:     S^sub ← CoarsenSample(S, δ, afs(·));   ▷ Use Algorithm 2
6:     E_loc ← ComputeLocalEdges(S^sub, δ, afs(·));   ▷ Use Algorithm 3
7:     E_bri ← ComputeBridgeEdges(S^sub, S, δ, afs(·));   ▷ Use Algorithm 5
8:     G ← (S^sub, E_loc ∪ E_bri);   ▷ Assemble graph G
9:     return APXDistancesFromGraph(S, G);   ▷ Expand result from S^sub to S
```

In the remainder of this section, we consider both types of edges in turn and show how to efficiently compute them. We then discuss how to combine local and bridge edges into a graph G that is sparse enough to be traversed efficiently. Finally, we analyze the resulting algorithm w.r.t. to its running time and approximation quality.

2.1 Computing Local Edges

The situation we are facing is similar to the construction of spanner graphs approximating the full Euclidean graph. One way of constructing a spanner graph is by means of the *well-separated pair decomposition* [10]. Informally, this approach connects (representative points of) clusters that are "far away" from each other, where the notion of "far away" depends on the radius of the clusters. Doing so, the intra-cluster distances are approximated by the length of the edge connecting the representatives.

We proceed along similar lines: we consider only edges between points that are at bounded distance from each other where the notion of "bounded" depends on the sampling density and the curvature and folding of Γ at the points in question. This allows us to relate Euclidean and geodesic distances and thus to upper-bound the approximation quality of the geodesic distances computed. A naive implementation of this approach, however, might lead to a linear number of edges per point. To avoid such situations, we need to ensure that only few "short" edges are constructed per point; this will be done by applying a standard preprocessing step in which the sample is coarsened appropriately without affecting the sampling quality [14].

In contrast to the construction of the spanner graph, the construction steps sketched do not guarantee a linear number of edges to be sufficient to obtain a good approximation quality. What we can show, however, is that by considering only edges for points whose Euclidean distance is upper-bounded as sketched above, we obtain a graph with $\mathcal{O}(n^{3/2})$ edges which still results in the desired approximation quality.

2.1.1 Characterizing and Approximating the Sampling Density

To measure the density of a point sample, it is customary to consider the *local feature size* $lfs(\cdot)$ that is defined as the distance function to the medial axis of Γ [4]. To formalize notation, a discrete subset S of $\Gamma \subset \mathbb{R}^3$ is an *ε-sample* of Γ if for every point $x \in \Gamma$ there is a sample point $s \in S$ such that its distance $|xs|$ to s is upper-bounded by $\varepsilon \cdot lfs(x)$. The local feature size $lfs(\cdot)$ is *c-Lipschitz* for $c = 1$, i.e., $lfs(x) \leq lfs(y) + c|xy| = lfs(y) + |xy|$, $x, y \in \mathbb{R}^3$.

In [14], we show that $|x_1x_2|$ for $x_1, x_2 \in \Gamma$ is an $(1 + \mathcal{O}(\varepsilon))$-approximation of the geodesic distance $L_\Gamma(x_1, x_2)$ on Γ between x_1 and x_2 if $|x_1x_2| \leq \sqrt{\varepsilon} \cdot \min\{lfs(x_1), lfs(x_2)\}$. Put differently, if points are "close enough" to each other, their Euclidean distance approximates

their geodesic distance. While we use this upper bound in [14] to prove the approximation quality of the geodesic distances computed, the discussion above suggests that our algorithm needs to actually evaluate these expressions to compute the radii of the locally bounded neighborhoods mentioned above and thus to be able to exclude points at larger distance from comparison. Unfortunately, neither ε nor $\mathit{lfs}(\cdot)$ can be computed exactly as Γ is unknown.

What we can do, however, is to compute an approximate upper bound $\mathit{afs}(\cdot)$, the *approximate local feature size*, for $\mathit{lfs}(\cdot)$ such that $\mathit{lfs}(s) \leq \mathcal{O}(1) \cdot \mathit{afs}(s)$ for all $s \in S$. Furthermore, we compute a so-called control function $\psi(\cdot)$ such that $\psi(s) \leq \mathcal{O}(\varepsilon) \cdot \mathit{lfs}(s)$ for all $s \in S$. Finally, we compute an approximate lower bound $\delta := \max_{s \in S}(\psi(s) / \mathit{afs}(s))$ for ε.

We follow Aichholzer et al. [1] and apply the distance from $s \in S$ to the closest *pole*.

▶ **Definition 2** ([4]). The *poles* of some $s \in S$ are the two vertices of the Voronoi cell $\mathit{Vor}_S(s)$ of s in the Voronoi diagram of S which are farthest from s, one on either side of Γ (note that Γ is the boundary of a 2-manifold, hence the inside and outside are well-defined). A pole p_s of s is called an *outer* pole if it lies outside Γ, and an *inner* pole otherwise.

Aichholzer et al. observed that this distance is the desired approximate upper bound $\mathit{afs}(s)$ for the local feature size $\mathit{lfs}(s)$, i.e., that $\mathit{lfs}(s) \leq 1.2802 \cdot \mathit{afs}(s)$ holds. Obviously, we can compute the (Voronoi diagram and the) poles and, hence, $\mathit{afs}(\cdot)$ in quadratic time. While the algorithm only needs to know the values of $\mathit{afs}(\cdot)$ for all points in S, the analysis will use a version of this function lifted to $\Gamma \subset \mathbb{R}^3$. For this analysis, we can assume that we can extend the domain of $\mathit{afs}(\cdot)$ to Γ– if needed, pointwise – such that $\mathit{lfs}(x) \leq 1.2802 \cdot \mathit{afs}(x)$ holds for all $x \in \Gamma$ (note that $\mathit{lfs}(\cdot)$ is defined for each point $x \in \Gamma$).

▶ **Observation 3.** *$\mathit{afs}(\cdot)$ is 1-Lipschitz and can be computed in $\mathcal{O}(n^2)$ time.*

We then obtain the approximate lower bound δ for ε using a *control function*:

▶ **Lemma 4** ([14]). *We can compute in time $\mathcal{O}(n^2)$ a control function $\psi : S \longrightarrow \mathbb{R}^+$ such that: (1) $\forall s \in S : \psi(s) \leq 1.19 \cdot \varepsilon \cdot \mathit{lfs}(s)$, (2) $\forall s \in S : \forall x \in \mathit{Vor}(s) \cap \Gamma : |xs| \leq \psi(s)$, and (3) ψ is $\frac{1}{18}$-Lipschitz.*

In line with the above argumentation, for $s_1, s_2 \in S$ we show $|s_1 s_2| \leq (1 + \mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1, s_2)$ if $|s_1 s_2| \leq \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{\mathit{afs}(s_1), \mathit{afs}(s_2)\}$ holds, i.e., the above approximation scheme still yields meaningful results for sample points $s_1, s_2 \in S$ "close enough" to each other.

In our previous work [15], we proved:

▶ **Lemma 5** ([15, Lemma 23]). *There is a global and shape-independent constant ε_0 such that for all $\varepsilon \leq \epsilon_0$, the approximate lower bound δ for ε satisfies $\frac{1}{\delta} \in \mathcal{O}(\sqrt{n})$.*

2.1.2 Coarsening the Sample

It remains to discuss how to avoid high-degree nodes in the distance graph, or, equivalently, to avoid connecting points to "too many" other points that fulfill the above distance criterion. For this, we use the control function implied by Lemma 4 to compute a coarsened subsample $S^{sub} \subseteq S$ in which the following two conditions hold:

1. For each point $x \in \Gamma$, there is a sample point $s \in S^{sub}$, such that $|xs| \leq \mathcal{O}(\delta) \cdot \mathit{afs}(s)$.
2. For any two sample points $s \neq s' \in S^{sub}$, $|ss'| \geq \mathcal{O}(\delta) \cdot \mathit{afs}(s)$ holds.

Algorithm 2 Compute a coarsened subsample $S^{sub} \subseteq S$ (see [9, 14]).

```
1: function CoarsenSample(S, δ, afs(·))
2:     S^sub ← ∅; β ← 0.1;          ▷ Fix constant in "big-oh"-notation for later analysis
3:     while S ≠ ∅ do
4:         s ← arbitrary point in S;
5:         S^sub ← S^sub ∪ {s};
6:         S ← S \ B_{β·δ·afs(s)}(s);          ▷ B_r(x): ball with radius r centered at x
7:     return S^sub;
```

Algorithm 3 Compute the set E_{loc} of local edges.

```
1: function ComputeLocalEdges(S^sub, δ, afs(·))
2:     E_loc ← ∅;
3:     for all (s_1, s_2) ∈ S^sub × S^sub do
4:         if |s_1 s_2| ≤ (1/3) · √δ · min{afs(s_1), afs(s_2)} then
5:             e ← (s_1, s_2); weight(e) ← |s_1 s_2|; E_loc ← E_loc ∪ {e};
6:     return E_loc;
```

2.1.3 Intermediate Summary: Computing Local Edges

Summarizing the above discussion, we first coarsen the subsample using Algorithm 2 and then construct local edges between all points that are close enough—see Algorithm 3.

▶ **Lemma 6.** *Let S be an ε-sample and E_{loc} the set of local edges computed for a coarsened subsample $S^{sub} \subseteq S$ according to Algorithm 2 and 3. Then, the following properties hold:*

(LE1) *The length of a local edge is a $(1 \pm \mathcal{O}(\sqrt{\varepsilon}))$-approximation of the geodesic distance of its endpoints. More precisely, for all $s_1, s_2 \in S$ such that $|s_1 s_2| \leq \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{afs(s_1), afs(s_2)\}$ holds, we have $L^{\star}_{\Gamma}(s_1, s_2) \leq (1 + \mathcal{O}(\delta)) \cdot |s_1 s_2| \leq (1 + \mathcal{O}(\sqrt{\varepsilon})) \cdot |s_1 s_2|$.*

(LE2) *Local edges can be asymptotically longer than the sampling density (described in terms of $afs(\cdot)$ and δ) by a factor of $\Theta(\sqrt{\delta})$, i.e., they connect points that are relatively "far away" from each other similar to cluster centers in a well-separated pair decomposition.*

(LE3) *Each sample point $s \in S^{sub}$ is incident to at most $\mathcal{O}(\sqrt{n})$ local edges.*

Proof.

(LE1) (Omitted due to space constraints.)

(LE2) Fix a point $s \in S^{sub}$ and let x be any point in $\mathrm{Vor}_{S^{sub}}(s)$. By Lemma 7 there is some $s' \in S^{sub}$ such that $|xs'| \leq 1.17 \cdot \delta \cdot afs(s')$. Since $afs(\cdot)$ is Lipschitz, it follows that $|xs'| \leq 1.2 \cdot \delta \cdot afs(x)$. Since $x \in \mathrm{Vor}_{S^{sub}}(s)$, we have $|xs| \leq |xs'| \leq 1.2 \cdot \delta \cdot afs(x)$. Again, since $afs(\cdot)$ is Lipschitz, we conclude that $|xs| \leq \Theta(\delta) \cdot afs(s)$. Now, consider any local edge $(s_1, s_2) \in S^{sub} \times S^{sub}$. By construction, the maximum length $\nu_{\max}$ of any such edge is $\nu_{\max} := \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{afs(s_1), afs(s_2)\}$. Even if $afs(s_1) = \max\{afs(s_1), afs(s_2)\}$, the fact that $afs(\cdot)$ is Lipschitz, together with the small distance of s_1 and s_2, implies that $\nu_{\max} \geq afs(s_1) \cdot \Theta(\sqrt{\delta})$. Thus, $\nu_{\max} \cdot \Theta(\sqrt{\delta}) \geq |xs_1|$. The same argument applies to s_2.

(LE3) [Sketch] As $afs(\cdot)$ is 1-Lipschitz, we can show $afs(s') \geq (1 - \frac{1}{3} \cdot \delta) \cdot afs(s) \geq \frac{1}{2} \cdot afs(s)$ for $s' \in B_{\frac{1}{3} \cdot \sqrt{\delta} \cdot afs(s)}(s)$. Algorithm 2 guarantees $|ss'| \geq 0.1 \cdot \delta \cdot afs(s)$ for all $s, s' \in S^{sub}$, $s \neq s'$. A standard packing argument yields $|B_{\frac{1}{3} \cdot \sqrt{\delta} \cdot afs(s)}(s) \cap S^{sub}| \leq \frac{1}{\delta}$. As all sample points connected to s by local edges lie inside $B_{\frac{1}{3} \cdot \sqrt{\delta} \cdot afs(s)}(s)$, we can upper-bound the number of local edges incident to s by $\frac{1}{\delta} \in \mathcal{O}(\sqrt{n})$; see Lemma 5. ◀

▶ **Lemma 7.** *For each $x \in \Gamma$, there is a $s' \in S^{sub}$ with $|xs'| \leq 1.17 \cdot \delta \cdot afs(s')$.*

Proof. Let $q \in S$ be the sample point closest to x. We distinguish between two cases:

1. $q \in S^{sub}$: We define $s' := q$. By definition, $\delta = \max_{p \in S} \frac{\psi(p)}{afs(p)} \geq \frac{\psi(s')}{afs(s')}$. Since, by Lemma 4, $|xs'| \leq \psi(s')$, we have $|xs'| \leq \frac{\psi(s')}{afs(s')} \cdot afs(s') \leq \delta \cdot afs(s')$.
2. $q \in S \setminus S^{sub}$: Define $s' \in S^{sub}$ to be the sample point that was processed by Algorithm 2 when q was excluded from further consideration (Line 6). With $\beta = 0.1$, this implies that $|s'q| \leq 0.1 \cdot \delta \cdot afs(s')$. As $afs(\cdot)$ is 1-Lipschitz, we get $afs(q) \leq (1 + 0.1 \cdot \delta) \cdot afs(s')$. Since q is the sample point closest to x, we have $|xq| \leq \delta \cdot afs(q)$ (see above). The triangle inequality implies then $|xs'| \leq |xq| + |qs'| \leq \delta \cdot afs(q) + 0.1 \cdot \delta \cdot afs(s') \leq 0.1 \cdot (1 + 0.1 \cdot \delta) \cdot \delta \cdot afs(s') + \delta \cdot afs(s') \leq 1.17 \cdot \delta \cdot afs(s')$ (since $\delta^2 < \delta$). ◀

2.2 Computing Bridge Edges

The second type of edges used in our construction is the set E_{bri} of bridge edges. While we would ideally compute the visibility graph of S^{sub} w.r.t. Γ, we cannot do so as the exact geometry of Γ is unknown. We thus compute E_{bri} as a superset of the edges in the visibility graph making sure that the additional edges that may intersect the interior of Γ do so not too deep; this will enable us to bound the approximation error.

2.2.1 Computing Approximate Visibility Information

As the exact nature of Γ is unknown, we cannot compute the visibility map of $s' \in S^{sub}$ w.r.t. Γ. Neither can we use a polyhedral reconstruction of Γ as the visibility map of s' w.r.t. such a reconstruction may have quadratic complexity [12, Sec. 2.1]. To circumvent this problem, we refrain from reconstructing Γ at all. Instead, we discretize Γ by a set of carefully constructed cubes corresponding to all points in S and compute the visibility maps w.r.t. theses cubes.

For this, we require that the visibility information obtained in this way approximates the true visibility information. First, we require that the cubes indeed cover Γ (recall that Γ is the boundary of a manifold F, not F itself) such that no obstacles are ignored or holes appear, and, second, we require that the cubes do not cover too much of the space outside Γ in the sense that they block visibility rays that Γ does not block. To fulfill these requirements, we compute, for each $s'' \in S^{sub}$, a cube that is centered at s'' and contains a ball of radius $2 \cdot \delta \cdot afs(s'')$ (intuitively, this means that the cubes are large enough to overlap with "neighboring" cubes and thus cover Γ). We then push all cubes towards the interior of F such that they do not protrude from F and thus block visibility rays that F would not block. Based upon Amenta and Bern's [4] observation that for any point $s \in S$ the vector from s towards one of its poles approximates the respective surface normals in s, we push the cubes along the vector from s towards its inner pole—see Algorithm 4.

Algorithm 4 Compute centers of the cubes pushed towards the interior of the manifold F.

```
1: function ComputeCubeCenters(S, δ, afs(·))
2:     for all s ∈ S do
3:         p_s ← inner pole of s;
4:         s↓ ← s + 15 · δ · afs(s) · sp_s→ / |sp_s|;
5:         S↓ ← S↓ ∪ {s↓};
6:     return S↓;
```

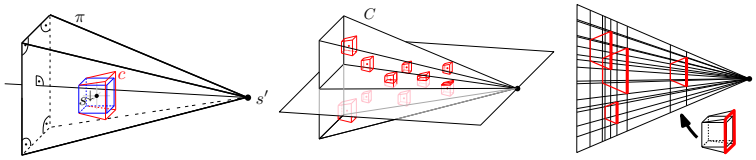

Figure 1 Left: Construction of a skewed cube $c \in \mathbb{C}(\pi, s')$ inside a pyramid. The base of c is the ball with center in s and a radius of $2 \cdot \delta \cdot lfs(s)$ w.r.t. the ℓ_∞-metric. Thus $B_{2 \cdot \delta \cdot lfs(s)}(s) \subset c$. The sides of c are slanted outwards from the back face by the same angle as the aperture of π, see cube c. Finally, c is pushed in the interior of the solid F bounded by Γ by pushing s' into the direction of p_s by $15 \cdot \delta \cdot afs(s)$ where p_s is the *inner pole that corresponds to* s. Middle: Cross-section of the pyramid during the sweep. Right: Projection of the cubes.

For technical reasons, when computing the visibility information of a point $s' \in S^{sub}$, we cover the space with a constant number of pyramids with apex at s' and compute the visibility information for each pyramid separately. After we have identified all cubes intersecting a pyramid π with apex s' (this takes linear time per each of the $\mathcal{O}(1)$ pyramids), we perform a top-down sweep (in decreasing z-order) over all cubes crossing π and all sample points in S^{sub} inside the cone and maintain only the cross-section of the sweeping plane with the scene. Whenever we encounter a sample point, we locate this point in the cross-section and check its visibility from s'. To ensure that maintaining the cross-section is not too costly, we do not work with the axis-aligned cubes we just computed. Instead, we approximate the scene by a set of carefully skewed cubes such that their cross-section with the sweeping plane can be maintained efficiently and their geometry does not induce too many events where the combinatorial nature of the cross-section changes.

More precisely, for a a pyramid π with apex s', we construct for each sample point $s \in S^{sub} \setminus \{s'\}$ a skewed cube $c := c_s$ such that the following properties hold—see Figure 1:

1. The front and back face of the cube c are parallel to the base of π.
2. The sides of c are slanted outwards from the back face by the same angle as π's aperture. (Here, we need that we are working with a constant number of pyramids per point.)
3. The cube c is centered at $s^{\downarrow}$.

For fixed $s' \in S^{sub}$ and π, we denote the set of all skewed cubes intersecting π by $\mathbb{C}(\pi, s')$. We define the *visible neighborhood* $V(\pi, s')$ *of* s' *w.r.t.* π as the union of all points from $S^{sub} \cap \pi$ that are visible from s' w.r.t. $\mathbb{C}(\pi, s')$.

▶ **Lemma 8.** *For fixed* $s' \in S^{sub}$ *and* π*, we can compute* $V(\pi, s')$ *in* $\mathcal{O}(n \log^2 n)$ *time.*

Proof. We perform a standard space-sweep in which we process the points and the skewed cubes' front faces in radial order from the top to the bottom face of the pyramid—see Figure 1 (middle). The important fact to note is that, by construction, the visible silhouette of the set of these cubes is exactly the projection of the set of their front faces onto the base of the pyramid–see Figure 1 (right). The sweep-line structure maintained by the algorithm is a segment tree $\mathcal{T}$ over the x-coordinates of the projections of these front faces. Whenever we encounter the top edge of a front face f, we add f to the set of obstacles currently active but inserting its x-interval into the sweep-line structure. At each node of $\mathcal{T}$ whose extent is covered by f, we insert f into a list of faces sorted by their distance to s'. Analogously, we remove a face from $\mathcal{T}$ once we encounter its bottom edge. Because of the way the skewed cubes have been constructed, i.e., because the aperture of the pyramid and the slanting angle of the cube coincide, the intersection of the sweeping plane and the skewed cubes changes only at the top and bottom edges of the cubes. Whenever we encounter a sample point s, we

query $\mathcal{T}$ with the x-coordinate of s. At each node of $\mathcal{T}$ visited, we check the sorted list of faces to see whether there is any face currently stored in $\mathcal{T}$ that blocks s from s'. If no such face is found along the root-to-leaf path in $\mathcal{T}$, s can be seen from s', otherwise s is blocked. The running time is easily seen to be $\mathcal{O}(n \log^2 n)$, as preprocessing takes $\mathcal{O}(n \log n)$ time and all update and query operations take at most $\mathcal{O}(\log n)$ time per node visited. ◀

Finally, we define the *visibility neighborhood* of s as $V(s') := \bigcup_{\pi \in \Pi} V(\pi, s')$.

▶ **Corollary 9.** *For each $s' \in S^{sub}$, we can compute $V(s')$ in $\mathcal{O}(n \log^2 n)$ time.*

2.2.2 Bounding the Degree of the Approximate Visibility Graph

Summarizing the above, we would like to connect each $s' \in S^{sub}$ with all $s \in V(s')$ by bridge edges. This approach can result in $|V(s')| \in \Theta(n)$. The final challenge thus is to compute an approximation of $V(s')$ that results in sublinear-degree vertices in the visibility graph.

▶ **Definition 10** ([13])**.** Let $X \subset \mathbb{R}^3$ be a discrete point set and let x be an arbitrary point in X. An approximate neighborhood $AH(x) := AH_\zeta(x)$ of x w.r.t. X is defined as a subset of $X \setminus \{x\}$, such that there exists a set of cones $\mathcal{C}(x) := \mathcal{C}_\zeta(x)$, with apex at x and an angular radius of ζ that covers $\mathbb{R}^3$, such that a point $x' \in X \setminus \{x\}$ belongs to the approximate neighborhood $AH(x) := AH_\zeta(x)$ iff there is a cone $C \in \mathcal{C}(x)$ such that x' is the point in $C \cap X$ minimizing the distance from its orthogonal projection onto the axis of C to x.

▶ **Lemma 11** ([13])**.** *For $\zeta > 0$, approximate neighborhoods, each one of size $\Theta\left(\zeta^{-2}\right)$, for all points from X can be computed in overall time $\mathcal{O}\left(|X|/\zeta^2 \cdot \log^2(|X|)\right)$*

Stated in terms of Lemma 11, we compute the set E_{bri} of bridge edges (see Section 2.2) in the weighted graph $G = (S^{sub}, E_{loc} \cup E_{bri})$ as follows: we iterate over all $s' \in S^{sub}$, compute $V(s')$ and then, for $\zeta := \sqrt{\delta} > 0$, approximate neighborhoods for the points in $V(s')$.

As a technicality, we wish to guarantee that bridge edges are not of local nature. Hence, we just consider edges that are longer than local edges (see Section 2.1.3). Thus, for $s' \in S^{sub}$, we define $A(s')$ as the $\sqrt{\delta}$-approximate neighborhood of s' w.r.t. $V(s') \setminus B_{\frac{1}{3} \cdot \sqrt{\delta} \cdot afs(s')}(s')$.

▶ **Corollary 12.** *For $s' \in S^{sub}$, we can compute $A(s')$ in $\mathcal{O}(\max\{n \log^2 n, n/\delta \log^2 n\})$ time.*

Algorithm 5 Compute the set E_{bri} of bridge edges.

```
1:  function ComputeBridgeEdges(S^sub, S, δ, afs(·))
2:      E_bri ← ∅;
3:      S^↓ ← ComputeCubeCenters(S, δ, afs(·));                ▷ Use Algorithm 4
4:      for s' ∈ S^sub do
5:          Π ← ComputePyramids(s', S^↓);                      ▷ See Figure 1
6:          V(s') ← ComputeVisibleNeighborhood(s', π, S^sub, S^↓);   ▷ Use Corollary 9
7:          A(s') ← APXVisibleNeighborhood(s', δ, V(s'));      ▷ Use Lemma 11
8:          for x ∈ A(s') do
9:              e ← (s', x); weight(e) ← |s'x|; E_bri ← E_bri ∪ {e};
10:     return E_bri;
```

We show that E_{bri} fulfils the requirements outlined at the beginning of Section 2.2.1:

▶ **Lemma 13.** *Let S be an ε-sample and E_{bri} the set of edges computed for a coarsened subsample $S^{sub} \subseteq S$ according to Algorithm 2 and 5. Then, the following properties hold:*

(BE1) *E_{bri} is a superset of the visibility edges of S^{sub} w.r.t. Γ: for $s \in S^{sub}$, E_{bri} contains all edges (s, s') such that $s' \in S^{sub}$ and $ss' \cap F^\circ = \emptyset$ (F is the solid bounded by Γ).*

(BE2) *Let $(s, s') \in E_{bri}$ such that $ss' \cap F^\circ \neq \emptyset$. The intersection is not too deep, hence, the shortcut taken not too short. More formally, for each edge $(s, s') \in E_{bri}$ and for any point $x \in ss' \cap F$, there is a sample point $s_x \in S^{sub}$ such that $|xs| \leq 18 \cdot \delta \cdot \min\{afs(x), afs(s_x)\}$.*

(BE3) *Each sample point $s \in S^{sub}$ is incident to at most $\mathcal{O}(\sqrt{n})$ bridge edges.*

Proof.

(BE1) We guarantee that each skewed cube c_s lies inside the solid F bounded by Γ. In order to do this, we first show that c lies inside a ball with radius $3.82 \cdot \delta \cdot afs(s)$ and centered in $s^\downarrow$, where s denotes the sample point corresponding to c. Furthermore, we show $s^\downarrow \in F$ and that the distance between $s^\downarrow$ and Γ is lower-bounded by $9 \cdot \delta \cdot afs(s^\downarrow)$. Finally, the triangle inequality implies (BE1). We omit the details due to space constraints.

(BE2) See Section 3.1.2 for the proof.

(BE3) For each $s' \in S^{sub}$, i.e., for each node of G, we use the algorithm by Ruppert and Seidel [13], to compute an approximate $\sqrt{\delta}$-neighborhood for s' w.r.t. $V(s')$ and connect s' to one representative per cone. With $\zeta := \sqrt{\delta}$, the nodes of G thus are incident to $\mathcal{O}(\zeta^{-2}) = \mathcal{O}(\delta^{-1})$ edges. Using again Lemma 5, we observe that $\delta^{-1} \in \mathcal{O}(\sqrt{n})$. ◀

Combining Lemma 6 and Lemma 13, we see that the weighted graph $G = (S^{sub}, E_{loc} \cup E_{bri})$ has a sublinear node degree and is well-suited to approximate the sought geodesic distances since the edges are either bridge edges, i.e., (approximate) visibility edges, or local edges, whose lengths are approximations of the geodesic distances of their endpoints.

2.3 Approximating All Distances / Runtime Analysis

We have described how to construct a weighted graph $G := \left(S^{sub}, E_{loc} \cup E_{bri}\right)$ on the coarsened set of sample points. While we can now use Dijkstra's algorithm to compute shortest distances in this graph, i.e., between points in S^{sub}, we also need to discuss how to compute distances between all sample points in S and not only between those in S^{sub}.

The main idea is borrowed from the construction of spanner graphs based upon well-separated pair decompositions [10]. If a sample point s has been excluded from S^{sub} because it was found to lie inside a ball $B_{\beta \cdot \delta \cdot afs(s')}(s')$ of some sample point $s' \in S^{sub}$, the distances to/from s' are good enough approximations of the distances to/from s as long as the destination is "far away"; otherwise, we use the Euclidean distance as an approximation.

Algorithm 6 Deriving an approximation $\mathcal{L}(\cdot, \cdot)$ of $L^\star_\Gamma(\cdot, \cdot)$ from G.

```
1:  function APXDistancesFromGraph(S, G)
2:      Compute shortest path distances L_G(s1, s2) for all s1, s2 ∈ S^sub.
                                  ▷ Use Dijkstra's algorithm from each point in S^sub;
3:      for all s ∈ S do
4:          ν_s ← sample point in S^sub closest to s;        ▷ ν_s = s for all s ∈ S^sub
5:      for all s1, s2 ∈ S do
6:          if |s1 s2| ≤ (1/3) · √δ · min{afs(s1), afs(s2)} then
7:              L(s1, s2) ← |s1 s2|;                  ▷ Approximate by Euclidean distance
8:          else
9:              L(s1, s2) ← L_G(ν_s1, ν_s2);          ▷ Approximate using closest points in S^sub
10:     return L(·, ·);
```

We now have collected all ingredients needed to analyze the running time of Algorithm 1.

▶ **Lemma 14.** *Algorithm 1 has a running time of* $\mathcal{O}(n^{5/2}\log^2 n)$.

Proof. To recall Algorithm 1: we first compute a control function (Lemma 4) and approximate the local feature size (Observation 3). Based upon these results, we can (asymptotically) lower-bound the local feature size. Since we need the poles for this, we construct the Voronoi diagram of all points, which can be done in $\mathcal{O}(n^2)$ time. In the next phase of the algorithm, we work with a coarsened subsample S^{sub} which can be constructed in $\mathcal{O}(n^2)$ time as well (Algorithm 2). Since we compute the set of local edges by iterating over all pairs of points in S^{sub} (Algorithm 3), this step takes $\mathcal{O}(n^2)$ time as well. Computing the set of bridge edges takes $\mathcal{O}(n \cdot \max\{n\log^2 n, n/\delta\log^2 n\}) \leq \mathcal{O}(n^{5/2}\log^2 n)$ time (Corollary 12, Lemma 5). The running time of the final step (Algorithm 6) is dominated by the $\Theta(n)$-fold invocation of Dijkstra's algorithm on a graph with $\mathcal{O}(n)$ vertices and $\mathcal{O}(n^{3/2})$ edges (Lemma 6 (LE3), Lemma 13 (BE3)). Using an efficient priority queue implementation, the running time for this step is $\mathcal{O}(n^{5/2}\log n)$. Hence, the overall running time of Algorithm 1 is $\mathcal{O}(n^{5/2}\log^2 n)$. ◀

3 Analysis of the Approximation Quality

▶ **Lemma 15.** $\mathcal{L}(\cdot,\cdot)$ *is an* $(1 \pm \mathcal{O}(\sqrt{\varepsilon}))$*-approximation of* $L^\star_\Gamma(\cdot,\cdot)$.

To prove Lemma 15, we first relate the value of δ to ε (Lemma 16). In Subsection 3.1, we then show $\mathcal{L}(\cdot,\cdot) \geq (1-\mathcal{O}(\sqrt{\varepsilon}))\cdot L^\star_\Gamma(\cdot,\cdot)$. Finally, we show $\mathcal{L}(\cdot,\cdot) \leq (1+\mathcal{O}(\sqrt{\varepsilon}))\cdot L^\star_\Gamma(\cdot,\cdot)$ (see Subsection 3.2). This concludes the proof of Lemma 15 and, hence, of Theorem 1.

▶ **Lemma 16.** $\delta \in \mathcal{O}(\varepsilon)$.

Proof. We combine Lemma 4 and Aichholzer et al.'s observation that $lfs(s) \leq 1.2802 \cdot afs(s)$ holds for all $s \in S$ [1, Lemma 5.1]: $\delta = \max_{s\in S}\frac{\psi(s)}{afs(s)} \leq \max_{s\in S}\frac{\varepsilon/(1-\varepsilon)\cdot lfs(s)}{1.2802^{-1}\cdot lfs(s)} \leq \mathcal{O}(\varepsilon)$. ◀

3.1 Lower-Bounding the Approximation Quality

To ensure $\mathcal{L}(s_1,s_2) \geq (1-\mathcal{O}(\sqrt{\varepsilon}))L^\star_\Gamma(s_1,s_2)$ for all $s_1, s_2 \in S$, we consider a shortest path ϕ in the distance graph G between ν_{s_1} and ν_{s_2}. We then construct a curve γ between s_1 and s_2 in the *free space* $\Lambda := \overline{\mathbb{R}^3 \setminus F}$ such that $|\gamma| \leq (1+\mathcal{O}(\sqrt{\varepsilon}))\cdot|\phi|$, or, equivalently, $(1-\mathcal{O}(\sqrt{\varepsilon}))\cdot|\gamma| \leq |\phi|$ holds. To show that $|\gamma| \leq (1+\mathcal{O}(\sqrt{\varepsilon}))\cdot|\phi|$ holds, we separately consider the local edges and the bridges edges on ϕ.

3.1.1 Lower-Bounding the Approximation Quality of Local Edges

Lemma 19 gives the lower bound for the length of local edges. In a previous paper [14], we proved a corresponding lower bound for edges whose lengths are related to ε and $lfs(\cdot)$:

▶ **Lemma 17** ([14, Lemma 20])**.** *For* $x, y \in \Gamma$ *with* $|xy| \leq \sqrt{\varepsilon}\cdot\min\{lfs(x), lfs(y)\}$*, we have* $L_\Gamma(x,y) \leq (1+\mathcal{O}(\varepsilon))\cdot|xy|$*, where* $L_\Gamma(x,y)$ *is the geodesic distance of* x *and* y *on* Γ.

Lemma 17 cannot be applied directly to a local edge $(p,q) \in E_{loc}$, since $|pq|$ depends on δ and $afs(\cdot)$ instead of ε and $lfs(\cdot)$. To extend this result to local edges, we can show that a similar statement also applies to free-space geodesic distances in our case:

▶ **Lemma 18.** *For* $x, y \in \Lambda$ *with* $|xy| \leq \sqrt{\varepsilon}\cdot\min\{lfs(x), lfs(y)\}$*, we have* $L^\star_\Gamma(x,y) \leq (1+\mathcal{O}(\varepsilon))\cdot|xy|$.

Lemma 18 allows us to iteratively construct the curve γ discussed above by connecting points on Γ and in Λ. We use γ to prove Lemma 19 (proof omitted due to space constraints):

▶ **Lemma 19.** *For $s_1, s_2 \in S$ with $|s_1 s_2| \leq \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{afs(s_1), afs(s_2)\}$, we have $L^\star_\Gamma(s_1, s_2) \leq (1 + \mathcal{O}(\sqrt{\varepsilon})) \cdot |s_1 s_2|$, i.e., $\mathcal{L}(s_1, s_2) \geq (1 - \mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1, s_2)$.*

3.1.2 Lower-Bounding the Approximation Quality of Bridge Edges

Lemma 26 gives the lower bound for the length of bridge edges. In a nutshell, we ensure that, given some $(s, q) \in E_{bri}$, for each $x \in \overline{sq} \cap \mathcal{F}$ there is a $s_x \in S^{sub}$ such that $|x s_x| \leq \mathcal{O}(\varepsilon) \cdot \min\{afs(s), afs(q)\}$. Applying Lemma 19 multiple times yields $L^\star_\Gamma(s, q) \leq (1 + \mathcal{O}(\sqrt{\varepsilon})) \cdot |sq|$.

To ensure the existence of s_x for $x \in \overline{sq} \cap \mathcal{F}$, we ensure that x lies not "too deep" inside $\mathcal{F}$, see (BE2). For this, we consider the restricted Delaunay triangulation of S w.r.t. Γ.

▶ **Definition 20** ([5])**.** Let t be the triangle induced by three sample points $s_1, s_2, s_3 \in S$. t is an element of the *restricted Delaunay triangulation* T iff $\mathrm{Vor}_S(s_1) \cap \mathrm{Vor}_S(s_2) \cap \mathrm{Vor}_S(s_3) \cap \Gamma \neq \emptyset$.

▶ **Theorem 21** ([5, Theorem 19])**.** *T is homeomorphic to Γ for $\varepsilon < 0.06$.*

▶ **Definition 22.** For $t \in T$ with corners $s_1, s_2, s_3 \in S$ let $t^\downarrow$ be the triangle that is induced by $s_1^\downarrow$, $s_2^\downarrow$, and $s_3^\downarrow$ (see Algorithm 4 for a definition of $\cdot^\downarrow$). We define $T^\downarrow := \{t^\downarrow \mid t \in T\}$.

For each constructed visibility edge, we can show that the skewed cubes cover $T^\downarrow$:

▶ **Lemma 23.** *For each $s \in S$ and $\pi \in \Pi$, we have $T^\downarrow \subset \bigcup_{c \in \mathbb{C}(\pi, s)} c$.*

We formalize the space Δ "between" $T^\downarrow$ and Γ as follows: For each $t = \triangle(s_1, s_2, s_3) \in T$ with $s_1, s_2, s_3 \in S$ and $\zeta \in [0, 1]$, we define $t_\zeta := \triangle(s_1 + \zeta(s_1^\downarrow - s_1), s_2 + \zeta(s_2^\downarrow - s_2), s_3 + \zeta(s_3^\downarrow - s_3))$. Also, for $x \in \Gamma$ and $\zeta \in [0, 1]$, we define $x_\zeta := x + \zeta(\mu_1(x) - x)$. Finally, we denote $\Delta := \left(\bigcup_{t \in T, \zeta \in [0,1]} t_\zeta\right) \cup \left(\bigcup_{x \in \Gamma, \zeta \in [0,1]} x_\zeta\right)$.

Assume now that there were some $x \in \overline{sq} \cap \mathcal{F}$ not "between" $T^\downarrow$ and Γ, i.e., $\overline{sq}$ were penetrating $\mathcal{F}$ "too deeply". Theorem 21 and the construction of $T^\downarrow$ then would imply the existence of some intersection point y of $\overline{sq}$ and some $t \in T^\downarrow$. Lemma 23 would then imply y to lie in one of the skewed cubes used during the construction of the visibility edge between s and q—a contradiction to the correctness of the space-sweep algorithm.

More formally, Theorem 21 implies there is a homeomorphism $\mu_1 : \Gamma \to T$. By construction, there is a continuous and surjective function $\mu_2 : T \to T^\downarrow$. Thus, $\mu := \mu_2 \circ \mu_1$ is surjective and continuous. This construction of μ implies that $T^\downarrow$ has the same genus as Γ, i.e., has no extra holes. Note that it is not guaranteed that triangles from $T^\downarrow$ are intersection free. Using elementary manipulations, we can show:

▶ **Lemma 24.** *For each $x \in \Delta$, there is an $s_x \in S^{sub}$ such that $|x s_x| \leq 18 \cdot \delta \cdot afs(s_x)$.*

As μ is surjective and continuous, $\mathcal{F} \setminus \Delta$ is bounded by a subset of $T^\downarrow$. Combining this with Lemmas 24 and 23 implies that $\overline{sq}$ does not penetrate $\mathcal{F}$ "too deeply":

▶ **Lemma 25.** *For each $x \in \overline{sq} \cap \mathcal{F}$ there is a $s_x \in S^{sub}$ with $|x s_x| \leq 18 \cdot \delta \cdot afs(x)$.*

Proof. Assume that there is some $x \in \overline{sq} \cap \mathcal{F}$ such that there is no $s_x \in S^{sub}$ with $|x s_x| \leq 18 \cdot \delta \cdot afs(s_x)$. The contraposition of Lemma 24 implies $x \in \mathcal{F} \setminus \Delta$. As $\mu : \Gamma \to T^\downarrow$ is surjective and continuous, $T^\downarrow$ has the same genus as Γ, i.e., has no holes. Thus, there exists some $y \in \overline{sq} \cap T^\downarrow$. This implies for all $\pi \in \Pi$, there is some cube $c \in \mathbb{C}(\pi, s)$ such that $\overline{sq} \cap c \neq \emptyset$. Analogously, we obtain for all $\pi \in \Pi$, there is some cube $c \in \mathbb{C}(\pi, q)$ such that $\overline{sq} \cap c \neq \emptyset$. As $(s, q) \in E_{bri}$, this is a contradiction to the correctness of the space-sweep algorithm. ◀

▶ **Lemma 26.** *For $(s,q) \in E_{bri}$, we have $L^\star_\Gamma(s,q) \leq (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot |sq|$.*

Combining Lemmas 19 and 26 yields the lower bound for all edges.

3.2 Upper-Bounding the Approximation Quality

To ensure $\mathcal{L}(s_1,s_2) \leq (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1,s_2)$ for all $s_1,s_2 \in S$, we again distinguish whether $|s_1s_2| \leq \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{afs(s_1), afs(s_2)\}$ holds. If this is the case, we have $\mathcal{L}(s_1,s_2) = |s_1s_2|$, which is trivially upper-bounded by $(1+\mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1,s_2)$.

If $|s_1s_2| > \frac{1}{3} \cdot \sqrt{\delta} \cdot \min\{afs(s_1), afs(s_2)\}$, Lemma 7 yields $|s_1\nu_{s_1}| \leq \mathcal{O}(\delta) \cdot afs(\nu_{s_1})$ and $|s_2\nu_{s_2}| \leq \mathcal{O}(\delta) \cdot afs(\nu_{s_2})$. This implies $L^\star_\Gamma(\nu_{s_1},\nu_{s_2}) \leq (1+\mathcal{O}(\sqrt{\delta})) \cdot L^\star_\Gamma(s_1,s_2)$. As $\nu_{s_1},\nu_{s_2} \in S^{sub}$, we can show the required upper bound for $\mathcal{L}(s_1,s_2)$ by applying Lemma 25.

▶ **Lemma 27.** *For all $s_1,s_2 \in S^{sub}$, $\mathcal{L}(s_1,s_2) \leq (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1,s_2)$.*

Thus, $\mathcal{L}(s_1,s_2) \leq (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot (1+\mathcal{O}(\sqrt{\delta})) \cdot L^\star_\Gamma(s_1,s_2) \leq (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1,s_2)$.

▶ **Corollary 28.** *For all $s_1,s_2 \in S$, $\mathcal{L}(s_1,s_2) \leq (1+\mathcal{O}(\sqrt{\varepsilon})) \cdot L^\star_\Gamma(s_1,s_2)$.*

In conclusion, the discussion in this section consitutes a proof of Lemma 15, i.e., we have shown that $\mathcal{L}(\cdot,\cdot)$ is a $(1 \pm \mathcal{O}(\sqrt{\varepsilon}))$-approximation of $L^\star_\Gamma(\cdot,\cdot)$. Together with Lemma 14, where we showed the running time of our algorithm to be in $\mathcal{O}(n^{5/2}\log^2 n)$, this constitutes a proof of our main result (Theorem 1).

References

1 Oswin Aichholzer, Franz Aurenhammer, Thomas Hackl, Bernhard Kornberger, Simon Plantinga, Günter Rote, Astrid Sturm, and Gert Vegter. Recovering Structure from r-Sampled Objects. *Computer Graphics Forum*, 28(5):1349–1360, 2009.

2 Lyudmil Aleksandrov, Hristo Djidjev, Anil Maheshwari, and Jörg-Rüdiger Sack. An approximation algorithm for computing shortest paths in weighted 3-d domains. *Discr. Comp. Geom.*, 50(1):124–184, 2013.

3 Lyudmil Aleksandrov, Hristo N. Djidjev, Hua Guo, Anil Maheshwari, Doron Nussbaum, and Jörg-Rüdiger Sack. Algorithms for approximate shortest path queries on weighted polyhedral surfaces. *Discr. Comp. Geom.*, 44(4):762–801, 2010.

4 Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. *Discr. Comp. Geom.*, 22(4):481–504, 1999.

5 Nina Amenta, Sunghee Choi, Tamal K. Dey, and Naveen Leekha. A simple algorithm for homeomorphic surface reconstruction. *Intl. J. Comp. Geom. Appl.*, 12(1–2):125–141, 2002.

6 Prosenjit Bose, Anil Maheshwari, Chang Shu, and Stefanie Wuhrer. A survey of geodesic paths on 3D surfaces. *Comp. Geom.: Theory & Appl.*, 44(9):486–498, 2011.

7 John F. Canny and John H. Reif. New lower bound techniques for robot motion planning problems. In *Proc. Symp. Found. Comp. Sci.*, pp. 49–60, 1987.

8 Mark de Berg, Herman J. Haverkort, and Constantinos P. Tsirogiannis. Visibility maps of realistic terrains have linear smoothed complexity. *J. Comp. Geom.*, 1(1):57–71, 2010.

9 Stefan Funke and Edgar A. Ramos. Smooth-surface reconstruction in near-linear time. In *Proc. 13th Symp. Discr. Alg.*, pp. 781–790, 2002.

10 Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Approximate distance oracles for geometric graphs. In *Proc. 13th Symp. Discr. Alg.*, pp. 828–837, 2002.

11 Joseph S. B. Mitchell. Shortest paths and networks. In Jacob Eli Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, Discrete Mathematics and its Applications, chapter 27, pp. 607–641. CRC Press, 2nd ed., 2004.

12 Esther Moet, Marc J. van Kreveld, and A. Frank van der Stappen. On realistic terrains. *Comp. Geom.: Theory & Appl.*, 41(1–2):48–67, 2008.

13 Jim Ruppert and Raimund Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proc. 3rd Conf. Comp. Geom.*, pp. 207–210, 1991.

14 Christian Scheffer and Jan Vahrenhold. Approximating geodesic distances on 2-manifolds in $\mathbb{R}^3$. *Comp. Geom.: Theory & Appl.*, 47(2):125–140, 2014.

15 Christian Scheffer and Jan Vahrenhold. Approximating geodesic distances on 2-manifolds in $\mathbb{R}^3$: The weighted case. *Comp. Geom.: Theory & Appl.*, 47(8):789–808, 2014.

16 Yevgeny Schreiber and Micha Sharir. An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. *Discr. Comp. Geom.*, 39(1–3):500–579, 2008.