DOI: 10.1111/cgf.13248

Eurographics Symposium on Geometry Processing 2017
J. A. Bærentzen and K. Hildebrandt
(Guest Editors)

Volume 36 (2017), Number 5

Fast and Memory-Efficient Voronoi Diagram Construction on Triangle Meshes

Yipeng Qin Hongchuan Yu Jianjun Zhang

National Centre for Computer Animation, Bournemouth University, UK

Abstract

Geodesic based Voronoi diagrams play an important role in many applications of computer graphics. Constructing such Voronoi diagrams usually resorts to exact geodesics. However, exact geodesic computation always consumes lots of time and memory, which has become the bottleneck of constructing geodesic based Voronoi diagrams. In this paper, we propose the window-VTP algorithm, which can effectively reduce redundant computation and save memory. As a result, constructing Voronoi diagrams using the proposed window-VTP algorithm runs 3-8 times faster than Liu et al.'s method [LCT11], 1.2 times faster than its FWP-MMP variant and more importantly uses 10-70 times less memory than both of them.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

1. Introduction

Computing geodesic-metric-based Voronoi diagrams on triangle meshes works as a foundation for various applications in computer graphics, including remeshing [PC06,LCT11], surface reconstruction [PM15] and point pattern analysis [LCT11], etc. In these applications, geodesics are used as the distance metric because they reflect the intrinsic properties of surfaces and are invariant to isometric deformations. To construct accurate Voronoi diagrams, Liu et al. [LCT11] employed the MMP algorithm [SSK*05] to it. Compared to other exact geodesic algorithms (e.g. ICH [XW09], VTP [QHY*16]), the MMP algorithm has a unique feature: all the propagated windows are stored and trimmed on edges. The distinct advantage is to bring necessary geodesic information to edges for Voronoi diagram construction. However, as the MMP algorithm always consumes lots of time and memory, it has become the bottleneck of constructing geodesic based Voronoi diagrams. Recently, Xu et al. [XWL*15] proposed the FWP-MMP algorithm as an accelerated version of the MMP algorithm. But it still occupies too much memory to be applied to large scale models.

The main deficiency of the MMP algorithm is to propagate all windows to edges, which results in lots of computation on redundant windows, and even invalid ones. To speed up geodesic computation and save memory, we propose to use the Vertex-sorted Triangle Propagation (VTP) exact geodesic algorithm [QHY*16], which can identify and remove the maximum invalid windows. Moreover for the Voronoi diagram over a mesh, the boundaries of Voronoi cells only occupy a small number of triangles on it (Fig. 1). Thus, most of the windows are redundant in constructing Voronoi diagrams.

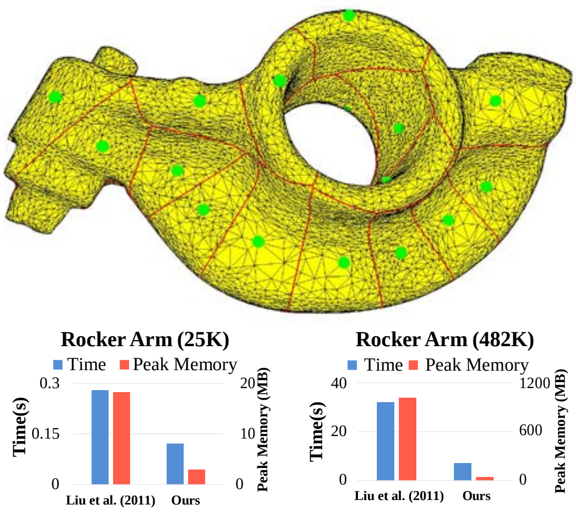


Figure 1: *Our algorithm outperforms Liu et al.'s method [LCT11] in both running time and peak memory. The upper figure shows the Voronoi diagram on the Rocker Arm model (25K faces). The lower charts compare the performance of Liu et al.'s method and ours on two Rocker Arm models (25K and 482K faces).*

This paper aims to reduce redundant computation so as to save time and memory as shown in Fig. 1. To this end, the Redundant Window Removal (RWR) process is proposed to remove redundant windows during the construction of a Voronoi diagram, and is involved in our *window*-VTP algorithm by selectively retaining windows on edges. The key point is to detect and remove redundant windows simultaneously with the geodesic wavefront propagation.

In summary, the contributions of this paper are:

- A novel Redundant Window Removal (RWR) method to remove redundant windows during the Voronoi diagram construction.
- The high efficiency of Voronoi diagram construction. Our method runs 3-8 times faster than Liu et al.'s method [LCT11], 1.2 times faster than its FWP-MMP variant and more importantly uses 10-70 times less memory than both of them, which is ideal for large scale models.

2. Related Work

Discrete Geodesic Computation. Mitchell et al. first formulated the computation of geodesic distances on triangle meshes as the *Discrete Geodesic Problem* (DGP) [MMP87]. To solve DGP quickly, PDE-based approximation algorithms have been proposed [KS98, CWW13]. However, these algorithms are sensitive to mesh quality and may produce potentially large errors [LCT11]. Thus, we prefer the exact geodesic algorithms as used in this paper.

The *window propagation framework* is employed by all the state-of-the-art exact geodesic algorithms [SSK*05, XW09, XWL*15, QHY*16]. In this framework, geodesics are encoded in a geometric data structure called *window* and propagated from the source over the mesh surface. To improve its performance, windows are sorted by a priority queue and propagated according to their distances in a continuous-Dijkstra style. During propagation, effective rules are applied to remove the redundant windows that cannot define geodesics, e.g. the window pruning rule [QHY*16]. Among these algorithms, the ICH algorithm [XW09], the FWP-CH algorithm [XWL*15] and the VTP algorithm [QHY*16] aim to compute geodesic distances of vertices. Thus, propagated windows are not stored on edges in these algorithms. On the other hand, the MMP algorithm [SSK*05] and the FWP-MMP algorithm [XWL*15] retain all propagated windows on edges and trim them into non-overlapping ones. Hence, the geodesic distance of a point within one triangle can be computed.

Voronoi Diagram Construction. The construction of Voronoi diagrams is studied in various metric spaces like Euclidean space [CM07, HR08] and Non-Euclidean spaces, e.g. spheres [NLC02], hyperbolic spaces [OT95], and Riemannian manifolds [OI03]. Refer to [Aur91] for a detailed survey.

In computer graphics, geodesic-metric-based Voronoi diagrams usually lie on top of triangular meshes. Kimmel and Sethian proposed the fast marching method [KS98] to compute such Voronoi diagrams [KS99]. However, since it is based on PDE, potentially large errors may occur on bad triangulated meshes. To compute Voronoi diagrams accurately, Liu et al. [LCT11] used the MMP algorithm for exact geodesic distance computation. Their method is extended by [XLS*14] to compute polyline-sourced Voronoi diagrams.

3. Redundant Window Removal (RWR)

Since the boundaries of Voronoi cells only cross a minority of the meshes' triangles, most of the windows stored on edges are redundant. Thus, this section aims to remove such windows which occupy a large amount of memory during the Voronoi diagram construction.

3.1. Preliminaries

For a triangular mesh M, its Voronoi diagram is a set of Voronoi cells partitioning M. As Fig. 2 shows, the boundaries separating Voronoi cells are closed curves spread over a small number of triangles. The definitions of Voronoi cells and their boundaries are presented as follows:

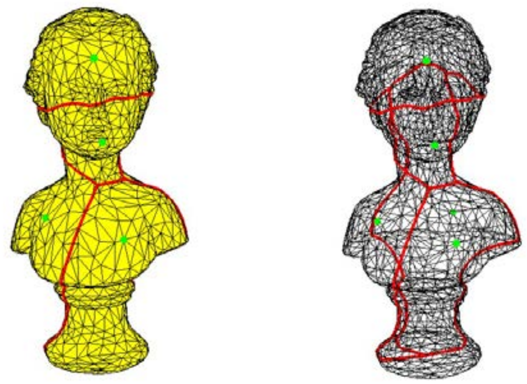


Figure 2: *Voronoi diagram on the Buste model (3K faces). Left: Voronoi diagram on the rendered model. Right: Voronoi diagram on the wireframe model. The green points are sources. The red curves are the boundaries of Voronoi cells.*

Voronoi Cell Definition [LCT11]. For a given set of source points $s_0, s_1, ..., s_n$ on mesh M, let $D_{s_i}(p)$ be the geodesic distance from source s_i to point p on M. Consequently, the Voronoi cell (VC) of each source point is defined as:

$$VC(s_i) = \{p|D_{s_i}(p) \le D_{s_j}(p), i \neq j, p \in M\}$$

Voronoi Boundary Definition. With the Voronoi cell definition above, the boundaries of Voronoi cells are formed by the collection of points q satisfying:

$$\exists i, j \text{ and } \forall k \text{ such that } D_{s_i}(q) = D_{s_j}(q) \le D_{s_k}(q), i \neq j \neq k \tag{3.1}$$

In this paper, geodesics on edges are encoded in "windows", which are used as the primitives for wavefront propagation in the state-of-the-art exact geodesic algorithms [SSK*05, XW09, XWL*15, QHY*16]. The definition of a window is presented as follows:

Window Definition. As Fig. 3 shows, a window w is located on edge AB, all the geodesic paths in w are from the same source s_i or

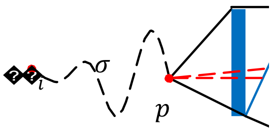


Figure 3: *Illustra*

pseudo-source p and share
defined as $w = (\Delta ABC, a_0,$, , , , ,
for the triangle it enters and
scalar parameters, a_0 and a
lies on the edge AB. Every
tex s_i or a pseudo source, w
represents the projection of
mined by ΔABC, and d_0, d_1 ,
spectively. σ denotes the g
to the source vertex s_i. **n**

Redundant Window Defi
intersection point of an edge
satisfy the condition Eq.3.1
originating from two diffe
occupied by the Voronoi b

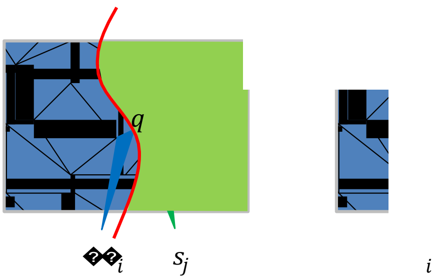


Figure 4: *Illustration of intersections b Voronoi cell boundaries. The left Voronoi source s_i and the right one (in green) is f curve denotes the boundary between the section shared by two windows from s_i a $D_{s_i}(q) = D_{s_j}(q)$. The two figures show t source positions.*

Definition 3.1 Given a mesh M with com
dundant primitives on M are (Fig. 5):

- **Redundant triangle.** A triangle is red
on its three edges are from the same so
- **Redundant edge.** An edge is redunda
gles are redundant triangles.

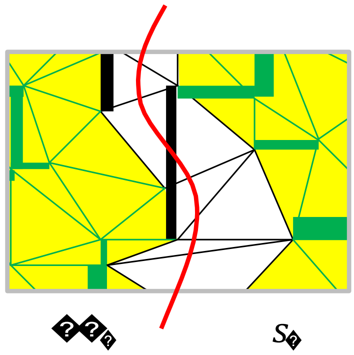


Figure 5: *Illustration of redundant primitives, including redundant triangles (yellow) and redundant edges (green).*

3.2. Redundant Windows Removal (RWR)

Definition 3.1 can be directly used to identify redundant windows after the termination of geodesic computation on a mesh. However, too much memory have been consumed. To avoid it, the redundant windows must be identified and removed as early as possible *during* the geodesic computation. To this end, we define the **inactive region** as follows:

Definition 3.2 An **inactive region** is a region behind the geodesic wavefront, in which all the windows will be no longer updated.

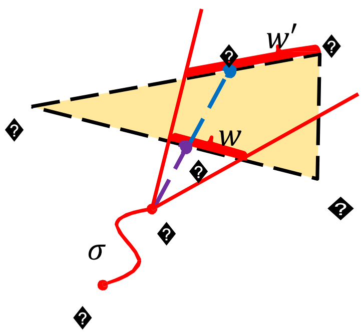


Figure 6: *Illustration of the monotonicity for window propagations. Point r (blue) resides in the window w propagated from w, segment pr intersects edge AB at point q (purple).*

Monotonicity. Mitchell et al. [MMP87] proposed the "continuous Dijkstra" technique to organize geodesic wavefront propagation from near to far *monotonically*. Herein, the wavefront consists of all the windows to be propagated and these windows are managed by a priority queue. In the priority queue, the priority of a window

w is defined as $-d_{min}(w)$, i.e. the negative minimum distance of a window. As Fig. 6 shows, if w' is a child window propagated from w, we have:

$$d_{min}(w') = \min(\sigma + |p'r|) \geq \min(\sigma + |pq|) \geq d_{min}(w)$$

That is, the minimum distances of windows popped from the priority queue are *monotonously* increasing.

Inactive Region Formation. To compute geodesics, windows are

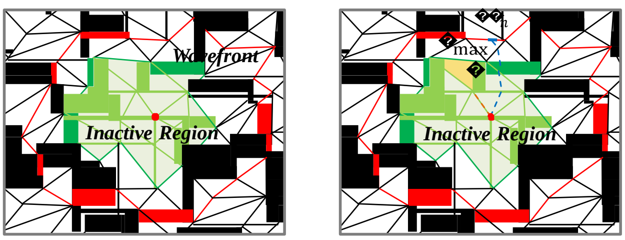


Figure 7: *Illustration of an inactive region. Left: the segments in red denote the propagation wavefront wf and the green shadowed area is the Inactive Region. Right: $d_{min}(f)$ is the length of the orange path, e_{max} is the longest edge of face f, $d_{min}(w_n)$ is the length of the blue path.*

Proposition 3.1 The *inactive region* is formed by all triangles satisfying $d_{min}(f) + e_{max} \leq d_{min}(w_n)$ and none of the windows in it can be updated by later window propagations.

Proof See Appendix B. □

Redundant Windows Removal (RWR) Redundant windows always appear within inactive regions. Thus, RWR works on inactive regions. Let f be a redundant triangle for removal, $d = d_{min}(w_n)$ be the distance of the nearest window on the propagation wavefront. Then, RWR is performed in two steps:

Step 1. Judge if f is in the inactive region with Proposition 3.1. If so, continue to Step 2; else, finish.
Step 2. Check f's redundancy with Definition 3.1. If f is redundant, also check if its edges are redundant and remove all windows on the redundant edges.

This process is summarized in Procedure 1.

3.3. Performance Verification

To verify that the proposed RWR procedure effectively reduces memory cost, this section compares memory costs against nearest distance $d_{min}(w_n)$ of the wavefront between two scenarios of

Procedure 1 Redundant Windows Removal (RWR)

Input: f - Face;
d - Distance of the nearest window on the wavefront;
Output: f - The face after redundancy removal;

```
procedure RWR(f, d)
    Let e_max be the longest edge of f;
    if d_min(f) + e_max ≤ d then
        Check f's redundancy;
        if f is redundant then
            for each edge e_i ∈ f do
                Let f_i be the face sharing edge e_i with f;
                if f_i is redundant then
                    Empty the windows on e_i;
                end if
            end for
        end if
    end if
end procedure
```

Voronoi diagram construction: with and without RWR. The tests are performed on ten models selected from the model set.

Fig. 8 shows the results on two models (Armadillo and Asian Dragon) and the rest of the results have been included in the supplementary materials. It can be seen that applying RWR dramatically reduces the memory cost of Voronoi diagram construction. Specifically, methods without RWR (e.g. [LCT11]) store all propagated windows on edges of the mesh and their memory costs are *cumulative*. On the contrary, RWR removes redundant windows in

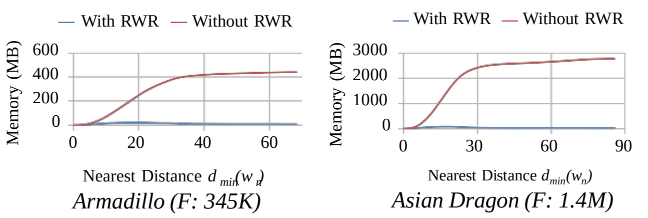


Figure 8: *Performance verification on RWR. The x-axis represents the distance of the nearest window on the wavefront during propagation, i.e. $d_{min}(w_n)$. The y-axis represents real-time memory cost during propagation.*

4. Applying RWR in Geodesic Computation

To construct geodesic-metric-based Voronoi diagrams, we propose the *window*-VTP algorithm by revising the original VTP algorithm [QHY*16]. The overall workflow is shown in Fig. 9. Our algorithm is essentially a multi-source geodesic algorithm and takes triangles as the primitive for distance propagation. For each source, all visited triangles form its own traversed area. We define the boundary of the traversed area as the propagation wavefront.

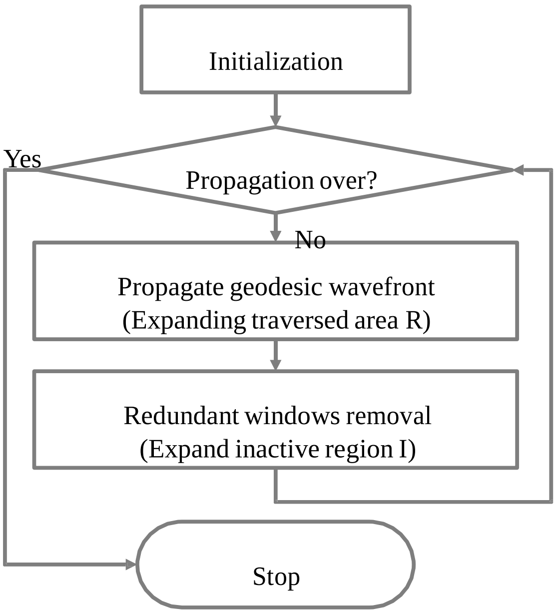


Figure 9: window-*VTP algorithm workflow.*

For simplicity, consider the one source scenario here. Our al rithm expands its traversed area *R* and inactive region *I* at the sa time (Fig. 10). Note that the **inactive region** *I* is a *proper subset* the **traversed area** *R*, i.e. $I \subset R$, and the windows in *I* will not updated. Both *R* and *I* are expanded in continuous Dijkstra st and gradually involving unvisited triangles abutting the wavefr First, the proposed algorithm creates the initial windows of e source within its 1-ring neighbourhood and pushes all the adjac vertices of each source into a priority queue *Q*. Note that we o define *one* priority queue *Q* for all traversed areas since eve

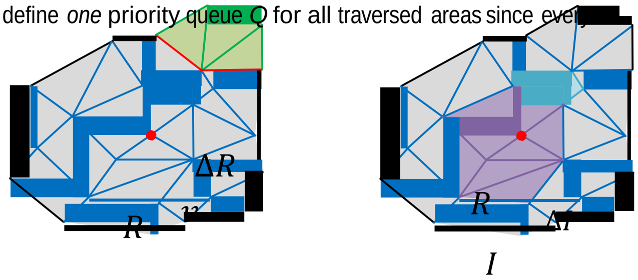


Figure 10: *Illustration of the triangle-oriented region expansion scheme. (a) Expansion of the traversed area R. (b) Expansion of the inactive region I.*

the unvisited triangles in *v*'s 1-ring neighbourhood. Then, *R* is expanded by involving ΔR into *R*, and the wavefront is also updated accordingly. Then, the windows on the previous wavefront (e.g. *vE* and *vB* in Fig. 11) are propagated through ΔR and *R* either till they reach the wavefront, or are eliminated during propagation. To manage windows on the wavefront for the Voronoi diagram construction, the propagated windows are trimmed on edges using the windows trimming and binary insertion methods proposed by the MMP algorithm [SSK*05].

- **Expanding Inactive region *I*.** As Fig. 10 (b) shows, the expansion of *I* is limited inside *R*. In the region between *I* and *R*, let ΔI

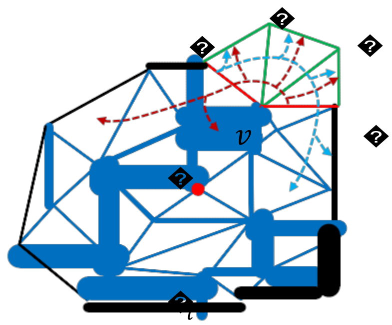


Figure 11: *Vertex-sorted Triangle Propagation [QHY*16].*

The outline of our algorithm is shown in Algorithm 2.

Two challenges are rising as below.

1. How to deal with the collision of the wavefronts? Note that it may be a self-intersection of one wavefront or meeting of two wavefronts.
2. How to define the priorities for triangles and vertices in Q_i and *Q* properly (in ***Step 4, 5***)?

4.1. Wavefront Collision

Proposition 4.1 The proposed *window*-VTP algorithm automatically handles the wavefront collisions and requires no extra operations.

As Fig. 12 shows, the propagation wavefront consists of different parts corresponding to different sources. When different parts of the wavefront collide with each other, we simply let the windows propagate through the wavefront and enter the interior of the

proposed *window*-VTP algorithm performs the following:

- **Expanding traversed area *R*.** As Fig 10 (a) shows, let ΔR be traversed areas. The propagations of these windows will stop when they reach the updated wavefront or be eliminated by the retained windows on edges in the traversed areas using the windows trimming rule [SSK*05]. Thus, no extra operation is required. For example in Fig. 12, the wavefront collides when ΔABC is added to the traversed areas. Then, the windows on edges AB, AC, BC are propagated into the interior of R_1, R_2 and R_3 (the dashed arrows in Fig. 12). These propagations will stop upon reaching the updated wavefront (the bold red, green, blue line segments in Fig. 12) or be eliminated on the interior edges (the grey line segments in Fig. 12).

Algorithm 2 *window*-VTP algorithm

Input: M - Mesh;
S - Source set;
Output: M - Mesh with sufficient geodesic information for Voronoi diagram constructions;

```
procedure window-VTP(M, S)
    Step 0. Perform Initialization.
    • For each source S_i, create a single window for every o
    posite edge of S_i in its 1-ring neighborhood (bold blue li
    around S_i in Fig. 11).
    • Push all adjacent vertices of S_i into a priority queue Q
    • Define a priority queue Q_i, which is used to organize
    expansion of the inactive regions;
    while !Q.empty() do
        Step 1. Pop a vertex v from Q;
        Step 2. Update the wavefront and traversed areas;
        Step 3. Expanding the traversed areas.
        • Push the windows on edges of the wavefront incid
        to v into FIFO queue W;
        while !W.empty() do
            • Pop a window w from W;
            • Propagate w across a triangle;
            • Retain and trim the propagated windows;
            • Push the propagate windows into W if they s
            vives the trimming and haven't reached the wa
            front;
        end while
        Step 4. Expanding the inactive regions.
        while !Q_i.empty() do
            • Let f be Q_i.front();
            • Perform RWR() on f to check if f is in the in
            tive regions; If so, remove the redundant windo
            on it; else, break the loop;
        end while
        Step 5. Update vertices' and triangles' priorities;
        Step 6. Push the faces newly added to the traversed
        eas into Q_i;
    end while
end procedure
```

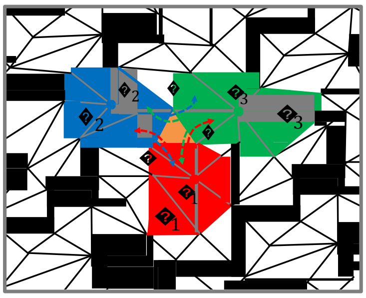
4.2. Priorities Definition

The key point of performing the procedure $RWR()$ during wavefront propagation is to form the **inactive region**, which resort to two priorities: the face's priority and the vertex's. Recall that the inequality of $d_{\min}(f) + \epsilon_{\max} \leq d_{\min}(w_n)$ is used to identify whether a face f is in the inactive region (Proposition 3.1). In our algorithm, the priorities are defined as follows:

Face's Priority. A face f's priority in the priority queue Q_i is defined as $-(d_{\min}(f) + \epsilon_{\max})$.

Vertex's Priority. A vertex v's priority in the priority queue Q is defined as the negative minimum of the current shortest distances to v's incident edges on the wavefront. For example in Fig. 13, $-d_{\min}(A) = -\min\{d_{\min}(AB), d_{\min}(AC)\}$. In addition, if w_n is on AB or AC, $-d_{\min}(A) = -d_{\min}(w_n)$.

Note that the two defined priorities are just the left and right sides



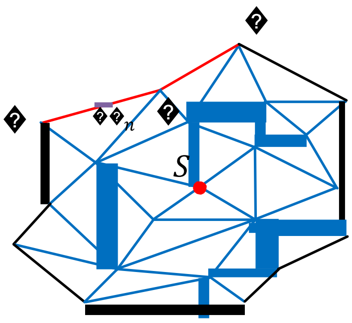


Figure 13: *Illustration of the vertex's priority definition. The propagation wavefront are the black and red line segments. w_n is the nearest window on the wavefront.*

of inequality $d_{\min}(f) + \epsilon_{\max} \leq d_{\min}(w_n)$ (Proposition 3.1), and thus they can be directly used when performing procedure $RWR()$.

5. Complexity Analysis

This section focuses on the complexity of geodesic computation since it is the dominant part of the Voronoi diagram construction [LCT11].

Let n be the number of vertices on a mesh. It is easy to verify that the proposed *window*-VTP algorithm is an improved version of the original MMP algorithm [MMP87]. In the worst case, the number of windows generated in the geodesic computation part is $O(n^2)$ and the time complexity of geodesic computation is $O(n^2 \log n)$. For the redundant windows removal (RWR) part, the checking and deletion processes are performed on each window and thus accounts for $O(n^2)$ time. In addition, the expansion of the inactive region is triangle-oriented and thus costs $O(n \log n)$ time for $O(n)$ triangles.

In summary, the time complexity of *window*-VTP is bounded by $O(n^2 \log n + n^2 + n \log n) = O(n^2 \log n)$. Since the redundant windows removal process does not consume extra memory, the space complexity of the proposed algorithms is bounded by $O(n^2)$.

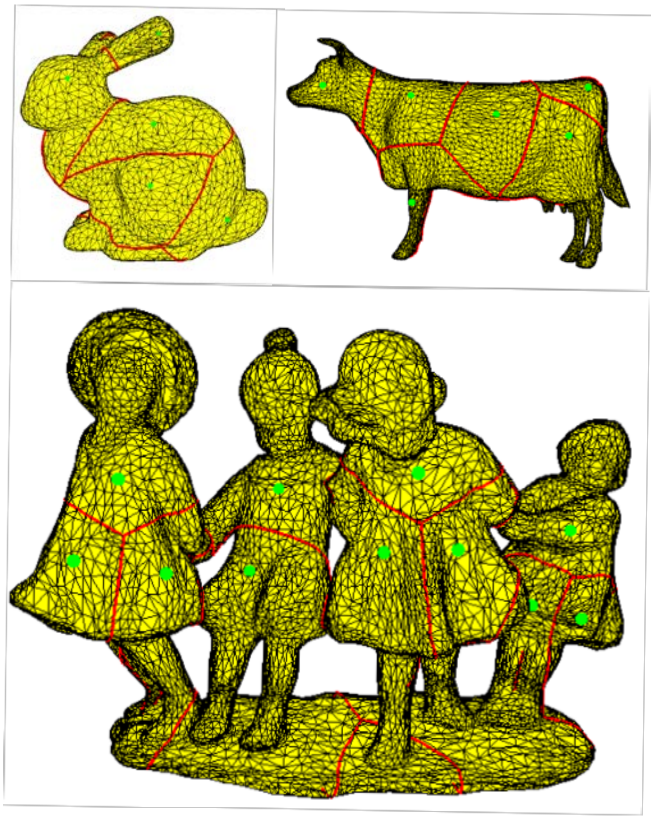


Figure 14: *Examples of Voronoi diagrams on meshes. The faces of the models are: Bunny (5K faces), Cow (*
dren (20K faces).

6. Experimental Results

To evaluate the performance of the pr
ments have been conducted on a variety of
test models are selected from the model se
including sculptures, animals and manma
of these models (number of faces) ranges f
algorithms are tested using a HP Z420 W
Xeon E5-1650 3.20GHz CPU and 32G
fied, the experiments randomly select 3
on meshes, as shown in [LCT11]. Fig. 1
Voronoi diagrams on some example mesh

6.1. Comparison with [LCT11]

Overall Performance According to [L
geodesic-metric-based Voronoi diagram c

- **Stage 1.** Compute geodesic distance fi
- **Stage 2.** Extract the valid triangles w h boundaries. March them to track and r of Voronoi cells' by linking the inters e edges of M.

The overall performance of the proposed algorithm is evaluated by two measures on the two stages: running time and peak memory usage respectively. As Table 1 shows, the geodesic computation part consumes the majority of time and memory in both Liu et al.'s ([LCT11]) method and ours. However, when replacing the MMP algorithm used in [LCT11] by the proposed *window*-VTP algorithm for geodesic computation, the Voronoi diagram construction runs 3-8 times faster and uses 10-70 times less memory.

Model	Performance	Liu et al. (2011)	Ours	Ratio
Horse	Time(s)	1.966 + 0.015	0.66 + 0.015	2.93
(F: 96K)	Peak memory(MB)	109.40 + 0.035	9.98 + 0.035	10.93
Bunny	Time(s)	3.637 + 0.028	1.07 + 0.028	3.34
(F: 144K)	Peak memory(MB)	187.00 + 0.046	14.86 + 0.046	12.55
Igea	Time(s)	10.916 + 0.048	3.019 + 0.048	3.57
(F: 268K)	Peak memory(MB)	478.06 + 0.065	26.50 + 0.065	18.00
Armadillo	Time(s)	9.863 + 0.046	2.982 + 0.046	3.27
(F: 345K)	Peak memory(MB)	440.33 + 0.066	21.09 + 0.066	20.81
Pulley	Time(s)	23.917 + 0.115	5.345 + 0.115	4.40
(F: 392K)	Peak memory(MB)	792.08 + 0.086	39.69 + 0.086	19.91
Rocker arm	Time(s)	32.012 + 0.091	6.985 + 0.091	4.54
(F: 482K)	Peak memory(MB)	1013.34 + 0.099	41.50 + 0.099	24.36
Asian dragon	Time(s)	110.083 + 0.255	20.281 + 0.255	5.37
(F: 1,400K)	Peak memory(MB)	2770.81 + 0.143	76.75 + 0.144	36.04
IsidoreHorse	Time(s)	89.538 + 0.211	21.229 + 0.211	4.17
(F: 2,209K)	Peak memory(MB)	2574.06 + 0.189	46.79 + 0.189	54.79
Happy buddha	Time(s)	482.715 + 1.291	58.946 + 1.291	8.04
(F: 2,583K)	Peak memory(MB)	8218.60 + 0.406	161.98 + 0.406	50.61
Neptune	Time(s)	832.83 + 0.784	96.843 + 0.784	8.54
(F: 4,008K)	Peak memory(MB)	13070.70 + 0.262	176.30 + 0.262	74.03

Table 1: *Performance comparison with [LCT11]. The results are shown in an addition manner as: "geodesic computation" +*

	MMP vs. window-VTP	*FWP-MMP vs. window-VTP*
Time	3.98/1.55	1.21/0.18
# windows stored	48.96/38.98	48.96/38.99
Peak Memory	21.24/15.16	21.24/15.16

Table 2: *The mean and standard deviation of the performance ratios between other algorithms and the proposed window-VTP algorithm on running time, the number of windows stored and peak memory usage. The table value is shown in "mean / standard deviation" format.*

The mean and standard deviation of performance ratios are calculated between MMP, FWP-MMP (the latest implementation of the MMP algorithm [XWL*15]) and the proposed *window*-VTP algorithm. The details are shown in Table 2. It can be seen that *window*-VTP on average runs 4 times as fast as MMP and comparable to FWP-MMP (1.2 times faster). The *window*-VTP algorithm on average uses 95.29% less memory than MMP and FWP-MMP. Furthermore, the *window*-VTP algorithm stores 97.96% less windows than MMP and FWP-MMP algorithms after propagation, which shows that it removes redundant windows effectively. Note that the proposed *window*-VTP algorithm is impressive since it resolves the memory bottleneck of Voronoi diagram oriented compu-

Model	Performance	Algorithms		
		MMP	FWP-MMP	*window*-VTP
Bunny (F:144K)	Time(s)	3.637	1.27	1.07
	# windows stored	2,451,104	2,451,105	85,959
	Peak Memory(MB)	187.00	187.00	14.86
Rocker Arm (F:482K)	Time(s)	32.012	9.088	6.985
	# windows stored	13,282,080	13,282,139	271,040
	Peak Memory(MB)	1013.34	1013.35	41.50
Asian Dragon (F:1,400K)	Time(s)	110.083	28.247	20.281
	# windows stored	36,317,620	36,317,847	346,142
	Peak Memory(MB)	2770.81	2770.83	76.75
Neptune (F:4,008K)	Time(s)	832.83	173.055	96.843
	# windows stored	171,319,703	171,374,203	857,068
	Peak Memory(MB)	13070.70	13074.80	176.30
Lucy (F:14,464K)	Time(s)	Out of memory	Out of memory	806.118
	# windows stored			12,071,796
	Peak Memory(MB)			921.005

Table 3: *Performance comparison between MMP, FWP-MMP and ours on five representative models.*

Number of Sources This section studies how the proposed algorithm performs with varying number of sources. First, three test models (Maxplanck, Angel, RedCircularBox) are chosen. For each model, eleven sets of sources are chosen randomly whose sizes

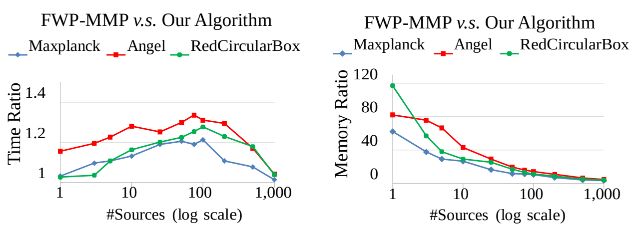


Figure 15: *Performance comparison between FWP-MMP based Voronoi diagram construction algorithm and ours on the number of sources. The x-axis represents the number of sources in logarithmic scale, and the y-axis represents the performance (time, memory) ratio.*

As illustrated in Figure 15, the time ratios increase within the range of source number at [1,100] and drop within the range at (100,1000]. This inconsistency is caused by RWR and the VTP wavefront propagation. When the number of sources increases,

- RWR is invoked less times. This is because the more triangles the Voronoi boundary occupies, the fewer the redundant windows.
- The performance of VTP wavefront propagation depends on the scale of the models, i.e. VTP performs better than the others on large scale meshes [QHY*16]. Herein, the size of Voronoi cells becomes smaller when the number of sources increases. VTP has to work within each cell, that is, the models' size becomes smaller for VTP.

The time ratio in Fig. 15 shows that in the range of [1,100], reducing RWR dominantly causes the time ratio increasing. In the range of (100,1000], the size of Voronoi cells becomes smaller, which leads to the performance of VTP decreasing. The low performance of VTP dominantly causes the time ratio decreasing at that time.

However, the memory ratio in Fig. 15 shows that the memory cost is close to that of FWP-MMP with an increasing number of sources. Nevertheless, the proposed algorithm still runs faster than the FWP-MMP based Voronoi diagram construction algorithm and uses more than 3 times less memory for 1000 sources.

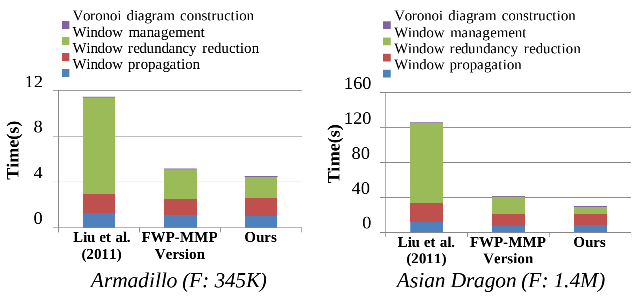


Figure 16: *Comparison of running times of four common components in Voronoi diagram construction on two models. The comparison is performed on three versions of the solution: (1) the original method in [LCT11]; (2) the FWP-MMP version which replaces the MMP algorithm used in [LCT11] with the FWP-MMP algorithm [XWL*15]; (3) Our version which replaces the MMP algorithm used in [LCT11] with the proposed window-VTP algorithm.*

agram construction contains two components: the computation of geodesics and the construction of a Voronoi diagram. In addition, the *geodesic computation* component can be further subdivided into three components [QHY*16]:

- **Window propagation** This component performs window propagations across the faces of a mesh.
- **Window redundancy reduction** This component identifies the redundant windows and removes them during propagation, including the window trimming and RWR processes.

- **Window management** This component manages the window propagations in order, which makes the *wind*
duction component more effective. In the p
the VTP framework [QHY*16] is employ
window lists and remove the redundant o
ing to their distances, which is implemented
and faces in priority queues. Compared to t
MMP [SSK*05, XWL*15] algorithms, the p
achieves low window management overhe
vertices/faces instead of $O(n^2)$ windows in
where n is the number of vertices on the me

The running times of these four individual co
ticipating algorithms are profiled on ten models
model set.

Fig. 16 shows the results on two models, A
Dragon (the rest of the results have been incl
mentary materials). Compared to the *geodesic c*
nents, the time cost of *Voronoi diagram constr*
small and can be neglected. For *geodesic comp*
it can be seen that the VTP framework effective
dow management cost of the Voronoi diagram c
ing vertices or faces in the priority queue rather
thermore, although an extra RWR process is a
the running time of the window redundancy r
is not dramatically increased as its time cost is
other computations (e.g. binary insertion and w

Scalability First, three test models (Cow, Shark
sen. Let each of them have six different resolutio

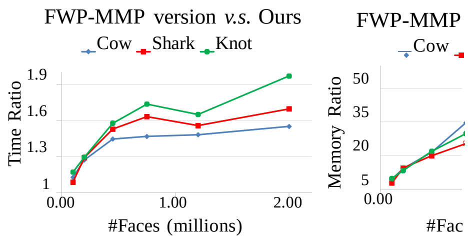
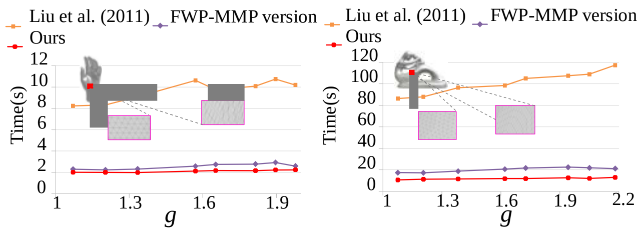


Figure 17: *Comparison of scalability against Voronoi diagram construction algorithm. The x mesh resolution, and the y-axis represents ru memory cost ratio.*

peak memory of FWP-MMP based Voronoi di
algorithm and that of ours on all six resolutions
experiments are designed to show how the rati
changing resolution. As illustrated in Fig. 17, b
and memory cost ratios increase with an incre
shown, the rate of increase in performance for t
rithm is proportional to the size of the models.

Robustness This section further validates
algorithm is robust to mesh triangulation q
[XWL*15], a sequence of meshes (eight) with

of anisotropy but a fixed resolution on two testing models (Fertility



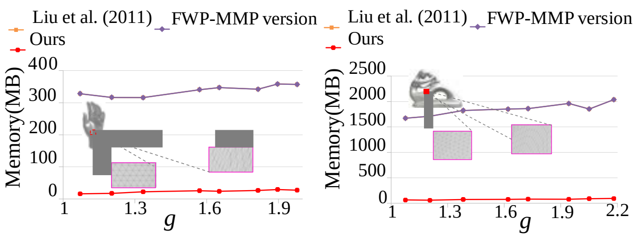


Figure 19: *Comparison of robustness against anisotropic triangulation (Memory). The x-axis represents the degree of anisotropy, and the y-axis represents peak memory.*

The curves in Fig. 18 and Fig. 19 show how the running times and peak memories change with increasing anisotropy (g) respectively. Note that the peak memories of Liu et al.'s method ([LCT11]) and its FWP-MMP based version are almost the same since both of them store all propagated windows on edges. The proposed *window*-VTP algorithm is the most robust among all algorithms since its running time and peak memory does not obviously increase when the input mesh has a much larger anisotropy.

6.2. Comparison with [XLS*14]

As Xu et al. have used the MMP algorithm to compute geodesics [XLS*14], its performance has already been compared in the preceding section and thus not discussed here.

Xu et al. proposed another method to reduce the memory cost of Voronoi diagram construction rather than the proposed RWR technique [XLS*14]. The main deficiency in their method is the *inefficiency* of the redundancy check. In their method, the redundancy check is performed on all unlabelled triangles rather than

Model	Performance	*window*-VTP + Xu et al. (2014) (c = [illegible])	Ours
Horse (F: 96K)	Time(s)	1.16	0.68
	Peak memory(MB)	13.38	10.01
Bunny (F: 144K)	Time(s)	1.93	1.10
	Peak memory(MB)	19.95	14.90
Igea (F: 268K)	Time(s)	5.42	3.07
	Peak memory(MB)	35.97	26.56
Armadillo (F: 345K)	Time(s)	5.04	3.03
	Peak memory(MB)	33.75	21.16
Pulley (F: 392K)	Time(s)	12.60	5.46
	Peak memory(MB)	58.17	39.78
Rocker arm (F: 482K)	Time(s)	12.41	7.08
	Peak memory(MB)	63.53	41.60
Asian dragon (F: 1,400K)	Time(s)	42.17	20.54
	Peak memory(MB)	132.99	76.90
IsidoreHorse (F: 2,209K)	Time(s)	29.73	21.51
	Peak memory(MB)	128.62	46.98
Happy buddha (F: 2,583K)	Time(s)	160.47	60.24
	Peak memory(MB)	493.70	162.39
Neptune (F: 4,008K)	Time(s)	195.45	97.63
	Peak memory(MB)	514.98	176.56

Table 4: *Performance comparison with [XLS*14].*

just the ones in the *inactive region* (Proposition 3.1). Thus, windows on many triangles are repeatedly checked since they are not inactive and will be updated by later propagated windows. In addition, since the cost of their redundancy check is large, performing it frequently is time-consuming. Thus, their method suffers from the *trade-off* between running time and memory-cost. In more details, they perform one redundancy check with every cn window propagations, where n is the face number of the mesh and c is a *user-defined* parameter to balance the performance. A smaller c means that the redundancy check is performed more frequently, reducing memory cost but sacrificing the running time.

On the contrary, the proposed RWR technique performs the redundancy check efficiently in the *inactive region* every t ir
tex is popped from the priority queue. To make a fair con
we compare our algorithm with an improved version of [
which uses the proposed *window*-VTP for geodesic co
but still employs their redundancy reduction method rather
RWR (Table 4). In the experiments, we set the parameter
a balanced performance. It can be seen that our algorith
forms [XLS*14] in both running time and peak memory.

6.3. Comparison with [QHY*16]

The original VTP algorithm does not retain windows,
revised version keeps partial windows. Compared to the
VTP, this experiment shows how the change influences t
mance.

As [QHY*16], in this experiment, we compare the pe
using the proposed *window*-VTP with the original VTP
the *single-source discrete geodesic problem*, with the fi
set as the source on the mesh. As Table 5 shows, our method runs approximately two times slower than VTP. The main reason is that the *window*-VTP has to strictly sort windows on edges by binary insertion. However, Voronoi diagrams are usually more sparse than meshes and there is no distinct decline in performance.

Model	Performance	VTP	Ours
Horse (F: 96K)	Time(s)	0.64	1.13
	Peak memory(MB)	1.25	5.67
Bunny (F: 144K)	Time(s)	0.88	1.50
	Peak memory(MB)	1.08	4.56
Igea (F: 268K)	Time(s)	2.04	4.11
	Peak memory(MB)	2.00	9.10
Armadillo (F: 345K)	Time(s)	1.68	2.68
	Peak memory(MB)	1.31	5.62
Pulley (F: 392K)	Time(s)	3.97	8.71
	Peak memory(MB)	4.53	18.58
Rocker arm (F: 482K)	Time(s)	4.26	9.32
	Peak memory(MB)	3.26	14.32
Asian dragon (F: 1,400K)	Time(s)	9.74	20.95
	Peak memory(MB)	3.72	16.77
IsidoreHorse (F: 2,209K)	Time(s)	10.41	17.72
	Peak memory(MB)	2.76	12.19
Happy buddha (F: 2,583K)	Time(s)	31.44	68.75
	Peak memory(MB)	8.44	40.46
Neptune (F: 4,008K)	Time(s)	51.62	91.14
	Peak memory(MB)	14.42	37.26

Table 5: *Performance comparison with VTP [QHY*16].*

6.4. Application to Remeshing

Due to that the Delaunay triangulation of a point set S is the dual of its Voronoi diagram, the proposed algorithm can be applied to remesh the dense models reconstructed from range data. In this context, the number of sources is usually fairly large and reaches the order of hundreds. Fig. 20 shows the remeshing result of the Neptune model with 4K randomly selected sources.

Figure 20: *Illustration of remeshing with the proposed algorithm.*

To show the performance of our method, we compare it with the

FWP-MMP version of [LCT11] on six dense models selected from the dataset of [QHY*16], whose numbers of faces range from 1.4M to 6.4M. For each model, we randomly select 2K sources if its number of faces is less than 2M; otherwise, 4K sources are selected. As Table 6 shows, our method runs faster and uses much less memory than the FWP-MMP version of [LCT11] in the remeshing problem.

# Samples: 2000			
Model	**Performance**	**FWP-MMP version**	**Ours**
Asian dragon	Time(s)	14.07	11.18
(F: 1,400K)	Peak memory(MB)	863.93	170.65
Pensatore	Time(s)	25.02	17.24
(F: 1,996K)	Peak memory(MB)	1503.96	251.48
Seahorse	Time(s)	23.24	17.26
(F: 2,014K)	Peak memory(MB)	1455.77	230.63

# Samples: 4000			
Model	**Performance**	**FWP-MMP version**	**Ours**
Happy buddha	Time(s)	28.21	23.48
(F: 2,583K)	Peak memory(MB)	1690.61	310.59
Neptune	Time(s)	52.26	39.07
(F: 4,008K)	Peak memory(MB)	2925.16	422.98
Vase lion	Time(s)	111.381	72.22
(F: 6,370K)	Peak memory(MB)	5567.37	673.80

Table 6: *Performance comparison with the FWP-MMP version of [LCT11] on remeshing.*

7. Conclusion

In this paper, the RWR procedure is presented to reduce the memory cost of constructing the geodesic-metric-based Voronoi diagrams, in which windows on edges are grouped within the inactive regions so that they can be removed together in time. The proposed *window*-VTP algorithm incorporates the RWR procedure in the vertex-oriented wavefront propagation framework. As a result, the *window*-VTP algorithm effectively resolves the memory bottleneck of the Voronoi diagram construction while not sacrificing the speed. In terms of experiments, our algorithm runs 3-8 times faster than Liu et al.'s method [LCT11], 1.2 times faster than its FWP-MMP variant and more importantly uses 10-70 times less memory than both of them.

In addition, the proposed method may be extended to compute other distances (e.g. anisotropic geodesic distances) on surfaces. All the Dijkstra-like approaches depend on the monotonicity of distance propagation. Thus, if the monotonicity is required, our method can work well.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments. This work was partially supported by the Royal Society Newton Mobility (Ref. IE151018) and EU H2020 RISE project-AniAge (Ref. 691215).

References

[Aur91] Aurenhammer F.: Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv. 23*, 3 (Sept. 1991), 345–405. 2

[CM07] Coeurjolly D., Montanvert A.: Optimal separable algorithms to compute the reverse euclidean distance transformation and discrete medial axis in arbitrary dimension. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 3 (March 2007), 437–448. 2

[CWW13] Crane K., Weischedel C., Wardetzky M.: Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph. 32*, 5 (Oct. 2013), 152:1–152:11. 2

[HR08] Hesselink W. H., Roerdink J. B. T. M.: Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*, 12 (Dec 2008), 2204–2217. 2

[KS98] Kimmel R., Sethian J. A.: Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences 95*, 15 (1998), 8431–8435. 2

[KS99] Kimmel R., Sethian J. A.: Fast voronoi diagrams and offsets on triangulated surfaces. In *Proc. of AFA Conf. on Curves and Surfaces* (1999), University Press, pp. 193–202. 2

[LCT11] Liu Y., Chen Z., Tang K.: Construction of iso-contours, bisectors, and voronoi diagrams on triangulated surfaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 33*, 8 (Aug 2011), 1502–1517. 1, 2, 4, 6, 7, 8, 9, 11

[MMP87] Mitchell J. S. B., Mount D. M., Papadimitriou C. H.: The discrete geodesic problem. *SIAM Journal on Computing 16*, 4 (1987), 647–668. 2, 3, 6

[NLC02] Na H.-S., Lee C.-N., Cheong O.: Voronoi diagrams on the sphere. *Computational Geometry 23*, 2 (2002), 183 – 194. 2

[OI03] Onishi K., Itoh J.-i.: Estimation of the necessary number of points in riemannian voronoi diagram. In *Proc. 15th Canadian Conf. Comput. Geom* (2003), pp. 19–24. 2

[OT95] Onishi K., Takayama N.: Construction of voronoi diagram on the upper half-plane. *IEICE Transactions 79* (1995), 533–539. 2

[PC06] Peyré G., Cohen L. D.: Geodesic remeshing using front propagation. *International Journal of Computer Vision 69*, 1 (2006), 145. 1

[PM15] Peethambaran J., Muthuganapathy R.: Reconstruction of water-tight surfaces through delaunay sculpting. *Computer-Aided Design 58* (2015), 62 – 72. Solid and Physical Modeling 2014. 1

[QHY*16] Qin Y., Han X., Yu H., Yu Y., Zhang J.: Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. *ACM Trans. Graph. 35*, 4 (July 2016), 125:1–125:13. 1, 2, 4, 5, 7, 8, 9, 10, 11

[SSK*05] Surazhsky V., Surazhsky T., Kirsanov D., Gortler S. J., Hoppe H.: Fast exact and approximate geodesics on meshes. *ACM Trans. Graph. 24*, 3 (July 2005), 553–560. 1, 2, 5, 9

[XLS*14] Xu C., Liu Y.-J., Sun Q., Li J., He Y.: Polyline-sourced geodesic voronoi diagrams on triangle meshes. *Comput. Graph. Forum 33*, 7 (Oct. 2014), 161–170. 2, 9, 10

[XW09] Xin S.-Q., Wang G.-J.: Improving chen and han's algorithm on the discrete geodesic problem. *ACM Trans. Graph. 28*, 4 (Sept. 2009), 104:1–104:8. 1, 2

[XWL*15] Xu C., Wang T., Liu Y.-J., Liu L., He Y.: Fast wavefront propagation (fwp) for computing exact geodesic distances on meshes. *Visualization and Computer Graphics, IEEE Transactions on 21*, 7 (July 2015), 822–834. 1, 2, 8, 9

[ZGW*13] Zhong Z., Guo X., Wang W., Lévy B., Sun F., Liu Y., Mao W.: Particle-based anisotropic surface meshing. *ACM Trans. Graph. 32*, 4 (July 2013), 99:1–99:14. 9

Appendix A: Lemma A.1

Lemma A.1 Given a triangle whose three edges' lengths are a, b and c respectively. Let l be the length of a line segment in the

, ,

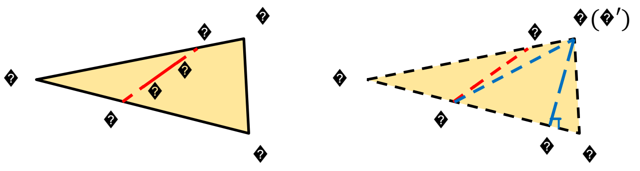


Figure 21: *Illustration of Lemma A.1.*

distance function over a line segment from G reaches extrema at the endpoints of the triangle edges, i.e. triangle vertices. Put H at any of such endpoints (e.g. F in Fig. 21) as H and thus $GH \leq GH$. Let I be a point on DE and $FI \perp DE$. Consider vertex $D \in \{D, E\}$ that D and G are on the same side of FI. It can be derived from the

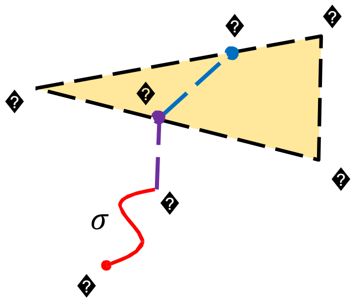


Figure 22: *Illustration of Proposition 3.1.*

Proof Let f be a face satisfying $d_{\min}(f) + \ e_{\max} \ \leq d_{\min}(w_n)$ and q is the point determining $d_{\min}(f)$, i.e. $d_{\min}(f) = \delta + \ pq \ = d(q)$ (Fig. 22).

Let r be an arbitrary point in any window on the edges of f, construct a path to r by linking q and r with a line segment. Then, the geodesic distance $d(r)$ of r must not be larger than the length of the constructed path, i.e. $d(r) \leq d_{\min}(f) + \ qr$. Since $qr \ \leq \ e_{\max}$ (Lemma A.1),

$$d(r) \leq d_{\min}(f) + \ qr$$
$$\leq d_{\min}(f) + \ e_{\max}$$

Knowing that f satisfies $d_{\min}(f) + \ e_{\max} \ \leq d_{\min}(w_n)$, then

$$d(r) \leq d_{\min}(w_n).$$

Thus, $d(r)$ cannot be updated by w_n since w_n cannot provide a shorter distance to r.

Let w_o be any other window on the propagation wavefront that $d_{\min}(w_n) \leq d_{\min}(w_o)$. Then, according to the *monotonicity* of window propagations,

$$d_{\min}(w_n) \leq d_{\min}(w_n)$$
$$d_{\min}(w_o) \leq d_{\min}(w_o)$$

where w_n and w_o are child windows propagated from w_n and w_o respectively. Then, it can be derived that,

$$d(r) \leq d_{\min}(w_n) \leq d_{\min}(w_n)$$
$$d(r) \leq d_{\min}(w_n) \leq d_{\min}(w_o) \leq d_{\min}(w_o)$$

Thus, $d(r)$ cannot be updated by all later window propagations. Since r is arbitrarily selected, all windows on f's edges will not be updated. □