Pacific Graphics 2014
J. Keyser, Y. J. Kim, and P. Wonka
(Guest Editors)

Volume 33 (2014), Number 7

Polyline-sourced Geodesic Voronoi Diagrams on Triangle Meshes

Chunxu Xu[1], Yong-Jin Liu[†1], Qian Sun[2], Jinyan Li[3] and Ying He[†2]

[1]TNList, Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2]School of Computer Engineering, Nanyang Technological University, Singapore
[3]Advanced Analytics Institute, University of Technology, Australia

Abstract

This paper studies the Voronoi diagrams on 2-manifold meshes based on geodesic metric (a.k.a. geodesic Voronoi diagrams or GVDs), which have polyline generators. We show that our general setting leads to situations more complicated than conventional 2D Euclidean Voronoi diagrams as well as point-source based GVDs, since a typical bisector contains line segments, hyperbolic segments and parabolic segments. To tackle this challenge, we introduce a new concept, called local Voronoi diagram (LVD), which is a combination of additively weighted Voronoi diagram and line-segment Voronoi diagram on a mesh triangle. We show that when restricting on a single mesh triangle, the GVD is a subset of the LVD and only two types of mesh triangles can contain GVD edges. Based on these results, we propose an efficient algorithm for constructing the GVD with polyline generators. Our algorithm runs in $O(nNlogN)$ time and takes $O(nN)$ space on an n-face mesh with m generators, where $N = \max\{m,n\}$. Computational results on real-world models demonstrate the efficiency of our algorithm.

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

2-manifold triangle meshes such as terrains and surfaces bounding 3D physical objects are widely used in computational geometry, computer graphics and robotics research. Computing Voronoi diagrams on 2-manifold meshes, which serve as a fundamental spatial data structure, can find a wide range of applications in motion planning [KKB98], graphical model remeshing [LWL*09], skeleton extraction [LCT11], and shape segmentation using line-segment Voronoi diagrams [LLW12], etc.

The Voronoi diagram in Euclidean space have been widely studied and understood [OBSC00]. However, Voronoi diagrams defined on 2-manifold triangle meshes based on geodesic metric (also known as geodesic Voronoi diagram or GVD) received only little attention. Due to the fundamental difference between Euclidean metric and geodesic metric, many 2D Euclidean properties do no hold on meshes. For example, a 2D Euclidean Voronoi cell is always convex, whereas a geodesic Voronoi cell is often concave.

Figure 1: *The polyline-sourced geodesic Voronoi diagram (GVD) on triangle meshes. The generators, bisectors and iso-distance contours are drawn in pink, red and black, respectively. The color indicates the distance to the generators.*

† Corresponding authors

In this paper, we investigate the GVDs in a more *general* setting, where the generators are polylines (Figure 1). We show that a typical GVD bisector may contain line segments, hyperbolic segments and parabolic segments. To tackle this challenge, we introduce a new concept, called *local Voronoi diagram*, or LVD, which is a combination of additively weighted Voronoi diagram and line-segment Voronoi diagram defined locally in the plane of a single mesh triangle. We show that when restricting on a mesh triangle, the GVD is a subset of the LVD. Moreover, only two types of mesh faces can contain GVD edges.

Guided by these results, we propose an efficient algorithm for constructing the exact GVD with polyline generators. Our algorithm can be integrated into the MMP framework [MMP87] in a seamless manner: once the MMP algorithm terminates, both the geodesic distance and the GVD are readily available. Such a feature fundamentally distinguishes our method from the existing GVD methods, which often separate the geodesic distance computation and Voronoi diagram construction. Inheriting the high performance of the MMP algorithm [SSK*05,LZH07,Liu13], our algorithm runs in $O(nN\log N)$ time and takes $O(nN)$ space on an n-face mesh with m generators, where $N=\max\{m,n\}$. Computational results on real-world models demonstrate the efficieny and robustness of our algorithm.

The rest of the paper is organized as follows. Section 2 briefly reviews the realted work. Section 3 presents preliminary background on geodesic distance computation. Section 4 documents our main results on GVD with polyline generators, followed by our algorithm for constructing GVD in Section 5. Section 6 shows the experimental results and Section 7 concludes the paper. Due to the space limit, we present the lengthy proof and some implementation details in the Supplemental Material.

2. Related Work

2.1. Discrete Geodesics

Finding shortest paths on triangle meshes is often referred to as discrete geodesic problem [MMP87]. The discrete geodesic can be computed by either PDE methods [KS98, CWW13] or computational geometry methods [MMP87, CH90, Liu13, YWH13]. Although PDE methods are fast, they provide only the approximate solutions (e.g., the first-order approximation by the fast marching method [KS98]). Since we need geometric structures that provide exact discrete geodesic information and sufficient information of trimmed bisectors for mutiple sources [LCT11], in this work we focus on the MMP method [MMP87].

The MMP method partitions each mesh edge into a set of intervals, called windows, over which the exact geodesic distance and path can be computed. The windows are propagated across the mesh faces using a priority queue in a conitnuous Dijkstra-like manner, and child windows are generated during the propagation. Different implementation techniques have been proposed [SSK*05, LZH07, Liu13] that make the MMP method one of the fastest exact geodesic computation methods.

2.2. Voronoi Diagrams

There is a large body of literature of Voronoi diagrams with non-point sources in Euclidean space, including line segments, circular arcs, parametric curves, etc. The reader is referred to [OBSC00] as an overview.

For non-Euclidean domains, a few research efforts have been devoted to constructing Voronoi diagrams with point-sources on smooth manifolds, such as the sphere $\mathbb{S}^2$ [AP85, NLC02], regular parametric surfaces [KWR97], the hyperbolic space $\mathbb{H}^2$ [OT96] and Riemannian manifolds [BDG13], where one can measure the geodesic distance analytically using differential forms.

In computer graphics and related fields, many models are represented by the non-differentiable polyhedral surfaces. Computing Voronoi diagrams on discrete surfaces is challenging, since many properties on the smooth manifold do not hold any longer. Kimmel and Sethian [KS99] computed Voronoi diagrams on meshes using the fast marching method (FMM) [KS98]. It is known that the FMM provides only the first-order approximation of the geodesic distance and it may produce poor results on meshes with irregular triangulation. Based on the exact discrete geodesic distance, Liu et al. [LCT11, LT13, LXHK14] studied the analytic structure of isocontours, bisectors and GVD with point sources. They also proposed practical algorithm for computing GVD with point sources in $O(n^2\log n)$ time, where n is the number of triangle faces. The medial residue, a concept related to GVD, was studied in [CJL13] that is a finite curve network homotopy equivalent to the original mesh.

Unlike existing research efforts, which were devoted to either Voronoi diagrams with non-point sources in Euclidean domain, or Voronoi diagrams with point sources on surfaces, our paper focuses on geodesic Voronoi diagrams with polyline sources, a more general and challenging problem.

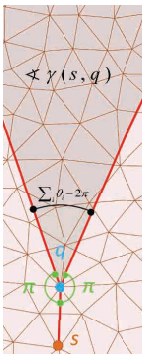
3. Preliminary

Let $M=(V,E,F)$ be the triangle mesh, where V, E and F are the set of vertices, edges and faces, respectively. Given points $p,q\in M$, denote by $\gamma(p,q)$ the geodesic path between p and q, and $d(p,q)$ the geodesic distance.

3.1. Point-source geodesic distance

Given an internal vertex $v\in V$, we call v *convex*, *Euclidean*, or *saddle*, if v's total angle is *less than*, *equal to*, or *greater than* 2π. Assume $s\in V$ is the source point. Imagine a point

light source is placed at s, the geodesic paths can be visualized as rays emanating from the source s in all tangent directions. Mitchell et al. [MMP87] showed that a geodesic path from the source s to a vertex v_i passes through a sequence of mesh faces. Inside a triangle, a geodesic path must be a straight line. When crossing over an edge, a geodesic path must correspond to a straight line if the two adjacent faces are unfolded into a common plane. It is well known that a geodesic path cannot pass through a spherical vertex, since perturbing the path a bit off the spherical vertex reduces the path length. When passing through a saddle vertex q, a geodesic path $\gamma(s,q)$ can split into multiple paths, which form a fan shaped area, denoted by $\lhd\gamma(s,q)$. See the right inset. Saddle vertex is also called *pseudo* source in the discrete geodesic algorithm, since it can illuminate the fan shaped area.



To compute the geodesic paths and distances on meshes, the MMP algorithm partitions each mesh edge into a set of intervals (called *windows*), in which all geodesic paths to the source share the same face sequence.

Definition A window w associated to a half edge e is a 6-tuple (a,b,d_0,d_1,σ,e), where

- a and b are the left and right endpoints of the interval;
- σ is the distance from the source to the pseudo source, which is the nearest saddle vertex (if exists) to w; $\sigma = 0$ when there is no saddle vertex on the geodesic path;
- d_0 and d_1 are the distances from the edge endpoints to the pseudo source.

To simplify the expression, we also use 2-tuple $w = (A,B)$ to represent the endpoints of window w. Clearly, the geodesic distance from the source to a window w can be computed by positioning the (pseudo-)source in $\mathbb{R}^2$ and measuring the Euclidean distance. To compute the single-source geodesic distance on M, we need to iteratively propagate the windows across the adjacent triangle, which yields new windows on the opposite half edge(s).

The MMP algorithm maintains a priority queue $\mathcal{Q}$ of windows, which represents the wavefront. The window's priority is determined by its distance to the source. The shorter the distance, the higher the priority the window has. Initially, $\mathcal{Q}$ consists of the windows that cover the edges facing the source. The algorithm computes the geodesic distance iteratively. For each iteration, it takes the window w with the highest priority from $\mathcal{Q}$, propagates w across the adjacent triangle to produce child window(s) $\hat{w}$. The algorithm repeats the above steps until the set $\mathcal{Q}$ is empty.

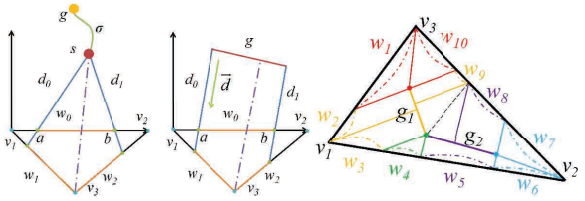


Figure 2: *Left: the point-source window w_0 on edge $\overline{v_1v_2}$ admits light rays emanating from the (pseudo-)source into the triangle $\triangle v_1v_2v_3$ and its borders define the illuminated regions (child windows w_1 and w_2) on the other sides of $\triangle v_1v_2v_3$. Middle: the line-source window acts similarly to the point-source window, except all the light rays are parallel. Right: consider two line-sources g_1 and g_2 in a triangle $\triangle v_1v_2v_3$. There are* 10 *windows covering the three sides of $\triangle v_1v_2v_3$. Among them, w_2, w_3, w_5, w_8 and w_9 are line-source windows, which are created by orthogonal projection of the line sources onto the sides. The other windows are point-source windows, where the sources are the end points of the generators.*

3.2. Polyline-source Geodesic Distance

The window defined in Section 3.1 is also called point-source window, since it admits light rays emanating from a point light source. The polyline-source geodesic distance was studied in [BK07, FS07, XYH11]. To compute geodesic distance with polyline generators, one needs to extend the point-source window to the line-source window so that directional light can come into the window. Similar to the point-source window, a line-source window associated to a half edge e is also a 6-tuple $(\vec{d},a,b,d_0,d_1,e)$, where $\vec{d}$ is the light direction (i.e., perpendicular to the source line segment), and the other 5 arguments are the same as the point-source window. Figure 2 shows an example of point- and line-source windows in a triangle. However, unlike the point-source window, a line-source window does not have pseudo-source, since all of the light rays emanate directly from the line segment.

We denote by $s(w)$ the *pseudo-source* of a window w if it is a point-source window, or the *source line segment* if w is a line-source window. Furthermore, when we use the symbol $s(w)$ in a 2D context, i.e., parameterizing the window w onto $\mathbb{R}^2$, it refers to the *2D position* of the pseudo-source or the source line segment.

When two windows w_1 and w_2 on edge e have a nonempty intersection $\delta = w_1 \cap w_2$, we must decide which of the windows defines the minimal distance for each point in δ. This can be done by finding the point $p \in \delta$ where the distance provided by w_1 and w_2 are equal.

4. GVD with Polyline Generators

Let $\mathcal{G} = \{g_i | g_i \in M, i = 1,\cdots,m\}$ be the set of generators (either points or polylines). The geodesic Voronoi cell asso-

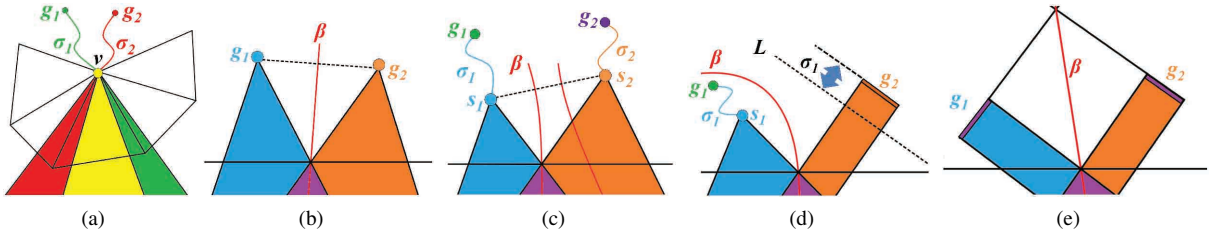


Figure 3: *The bisector of generators g_1 and g_2. (a) The bisector is a 2D region, when both g_1 and g_2 are points, a saddle vertex v is equidistant from the sources (i.e., $\sigma_1 = \sigma_2$), and the two fan shaped regions overlap (i.e., $\lhd\gamma(g_1,v) \cap \lhd\gamma(g_2,v) \neq \emptyset$). Then any point in the overlapped region (in yellow) is on the bisector. (b) The bisector is a line segment when both generators are points and $\sigma_1 = \sigma_2 = 0$. (c) The bisector is a hyperbolic segment when both generators are points and $\sigma_1\sigma_2 \neq 0$. Assume $\sigma_1 > \sigma_2$, then the bisector is the branch close to g_1; (d) The bisector is a parabolic segment when one generator (say g_1) is a point and the other is a line. The focus of the parabola is the pseudo source of the point-source window w_1, while the directrix L is parallel to the line-source window w_2, and is σ_1 apart. (e) The bisector is a line segment when both w_1 and w_2 are line-source windows and $\beta(g_1,g_2)$ bisects the angle formed by the (extension) of two generators.*

ciated with generator g_i is defined as

$$\{VC(g_i) | x \in M, d(x,g_i) \leq d(x,g_j), \forall g_j \in \mathcal{G}\}$$

In this section, we first investigate the geometry of bisectors, and then introduce a concept, called local Voronoi diagram, which is built upon the local information inside a single triangle. We show that a GVD restricted on a triangle is a subset of an LVD. Next, we show that there are only two types of triangles that can contain GVD bisectors.

4.1. Bisectors

We denote by $\beta(p,q)$ the bisector between generators p and q. We also abuse the notation by using $\beta(w_i,w_j)$ to denote the bisector between w_i and w_j's pseudo-sources. The following property shows that the bisectors of GVDs with polyline generators have more complicated situation than 2D Euclidean Voronoi diagrams and GVDs with point sources.

Property 4.1 Let $g_1, g_2 \in M$ be two distinct generators (points or polylines) on the mesh M. Bisector $\beta(g_1,g_2)$ can contain line segments, hyperbolic segments, parabola segments and even a 2D region.

Proof Let $p \in \beta(g_1,g_2)$ be a point on the bisector. The locus of p depends on the following conditions:
Case 1: when both g_1 and g_2 are points. Let s_i be the pseudo-source on path $\gamma(g_i,p)$, $i = 1,2$. Note s_i coincides with g_i if $\gamma(g_i,p)$ does not pass through any saddle vertex.

- Case 1.1: when $\sigma_1 = \sigma_2 = 0$, p is on the line segment bisecting the parameterized points g_1 and g_2. See Figure 3(b).
- Case 1.2: when $\sigma_1 + \sigma_2 > 0$, i.e., at least one of them is non-zero. Point p satisfies $\sigma_1 + d(s_1,p) = \sigma_2 + d(s_2,p)$ and it is on a hyperbola with foci s_1 and s_2. See Figure 3(c).
- Case 1.3: when $s_1 = s_2 \triangleq v$. This implies the pseudo-source v is *equidistant* to the two generators. Thus, any point $q \in \lhd\gamma(g_1,v) \cap \lhd\gamma(g_2,v)$ is equidistant to g_1 and g_2. See Figure 3(a).

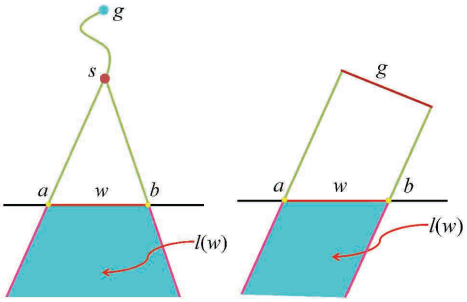


Figure 4: *Given a point light at the pseudo-source or a directional light from the line-source, the endpoints of the window w define the illuminated region $l(w)$.*

Case 2: when one generator (say g_1) is a line segment and the other is a point. Let s_2 be the pseudo-source on path $\gamma(p,g_2)$. Then p satisfies $d(p,g_1) = d(g_2,s_2) + d(p,s_2)$ and p is on a parabola with foci g_1 and directrix parallel to g_2. See Figure 3(d).
Case 3: when both generators are line segments. Then p is on the bisector of the angle formed by the g_1 and g_2 or their extensions. See Figure 3(e). □

Due to floating-point computation, it is very unlikely that two geodesic paths are of exactly the same length. Therefore, Case 1.3 is extremely rare in reality and we ignore it to simplify our analysis and computation.

To measure how much light can admit into the window, we define the window's illuminated region.

Definition Let w be a window on edge e. The *illuminated region* of w, denoted by $l(w)$, is the region lying on the side which is opposite to $s(w)$. The *border* of $l(w)$ consists of w and the two rays emanating from s. See Figure 4.

Definition Given an edge e and two *adjacent* windows $w_1 =$

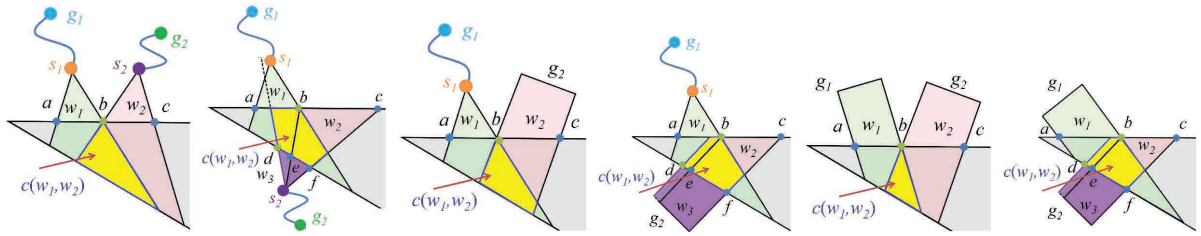


Figure 5: *Co-illuminated regions are defined for two adjacent windows $w_1 = (a,b)$ and $w_2 = (b,c)$ on edge e. If the generators are on the same side of e, the co-illuminated region (in yellow) is the intersection of w_1 and w_2's illuminated region. Otherwise, let w_3 be the parent window that can illuminate e from the other side. w_2 is obtained by trimming w_3's child window with w_1. Then the co-illuminated region (in yellow) is $c(w_1,w_3) = l(w_1) \cap l(w_3)$. (a)(b) Both generators are points. (c)(d) One generator is a point and the other is a line segment. (e)(f) Both generators are line segments.*

$(a,b) \in e$ and $w_2 = (b,c) \in e$, their *co-illuminated region*, denoted by $c(w_1,w_2)$, is defined as follows:
(1) if $s(w_1)$ and $s(w_2)$ are on the same side of e, $c(w_1,w_2)$ is the intersection of their illuminated regions, i.e., $c(w_1,w_2) = l(w_1) \cap l(w_2)$.
(2) otherwise, assume $s(w_2)$ is a generator on the other side of e and w_3 is the parent window of w_2. Then $c(w_1,w_2)$ is the intersection of w_1 and w_3's illuminated regions, i.e., $c(w_1,w_2) = l(w_1) \cap l(w_3)$. See the yellow regions in Figure 5.

Property 4.2 Upon the termination of the MMP algorithm, two adjacent windows w_i and w_j have a *non-empty* co-illuminated region. Moreover, bisector $\beta(w_1,w_2)$ is in the co-illuminated region $c(w_1,w_2)$.

Proof See the Supplemental Material. □

4.2. Local Voronoi Diagrams (LVDs)

Consider a triangle $t = (v_1,v_2,v_3) \in F$. Upon the termination of the MMP algorithm, each edge of t is covered by a set of non-overlapping windows. Let $\mathcal{P}(t) = \{s(w) | \forall w \in e(v_i,v_j), i,j = 1,2,3, i \neq j\}$ denote the set of pseudo-sources and line-sources for all windows on t's edges.

Definition A local Voronoi diagram on a triangle t, denoted by $\mathcal{L}(t)$, is the combination of additively weighted Voronoi diagram and line-segment Voronoi diagram restricted on t with $\mathcal{P}(t)$ as generators. The weight of a window w is the distance from its pseudo-source to the source if w is a point-source window, and 0 otherwise.

Property 4.3 Each LVD edge bisects two windows, and it does not intersect their borders.

Proof See the Supplemental Material. □

Property 4.3 tells us that the window's sources or pseudo-sources can fully determine the bisector and there is no need to trim an LVD edge with window's borders. This observation can simplify the LVD construction significantly.

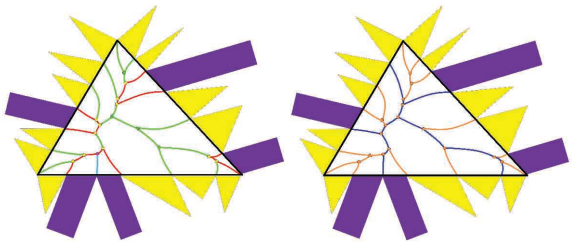


Figure 6: *The relationship between GVD and LVD. The point-source windows and line-source windows are drawn in yellow and purple respectively. (a) The LVD edges consist of hyperbolic segments (green), parabolic segments (red) and line segments (cyan). (b) The GVD edges (blue) are a subset of the LVD edges.*

Let us denote by $\mathcal{G}(t)$ the GVD restricted on triangle t. The following property reveals the relationship between LVD and GVD.

Property 4.4 The GVD restricted on a triangle t is a subset of the LVD on t, i.e., $\mathcal{G}(t) \subseteq \mathcal{L}(t)$.

Proof See the Supplemental Material. □

Remark Note that the converse of Property 4.4 is not true in general. For example, consider two point-source windows $w_1 \in e$ and $w_2 \in e$, which share the *same* generator but have *different* pseudo-sources $s(w_1) \neq s(w_2)$. Obviously, $\beta(w_1,w_2) \in \mathcal{L}(t)$ and $\beta(w_1,w_2) \notin \mathcal{G}(t)$. Figure 6 shows an example of the LVD and the GVD on a triangle.

4.3. Triangles Containing GVD Edges

Since the GVD restricted on a triangle is just a subset of the LVD on the same triangle, one can adopt a simple approach for constructing the GVD: first, construct the LVD on each mesh face and then find the LVD edges which belong to the GVD. However, this naïve method is not efficient at all, since

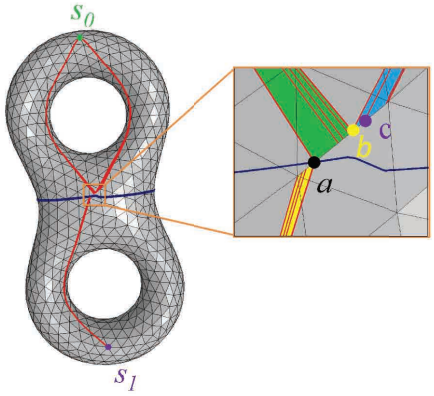


Figure 7: *The blue curve is the bisector of sources s_0 and s_1, and the red curves are geodesic paths. In the inset, the regions with the same color indicate the geodesic paths coming from the same pseudo-source. Point a, the common point of a yellow window and a green window, is a Category 1 point, since the two adjacent windows have distinct sources. Point b, the common point of a green window and a blue window, is a Category 2 point, since both windows are lit from the same source s_0, but their pseudo-sources are different. Point c is in Category 3, since the corresponding windows share the same source and pseudo-source.*

only a small number of mesh triangles contain GVD edges. Therefore, to develop an efficient algorithm for constructing GVD, it is important to know which triangles contain the GVD edges, instead of a brute force search of all triangles.

We add the generator's identifier to the 6-tuple representation of a window structure. As each edge is covered by windows without any gap or overlap, we can classify the common point of two adjacent windows into three categories: 1) the two sources are distinct; 2) the two sources are identical, but their pseudo-sources are different; 3) both sources and pseudo-sources are the same. Figure 7 illustrates the common points in the three categories. We call the common points in Category 1 the *key points*. Obviously, for any key point there exists some GVD edge passing through it.

Property 4.5 Only two types of triangles, namely, the ones having at least one key point on its side, or the ones having a source inside, can contain GVD edges.

Proof See the Supplemental Material. □

Triangles that can contain GVD edges are called *candidate* triangles.

5. Practical Algorithm

The input of the algorithm is a triangle mesh $M = (V,E,F)$ and a set of generators $\mathcal{G}$. The user also specifies a parameter c, which controls the algorithm's performance (will be explained later). The result of the algorithm is the undiscretized geodesic Voronoi diagram on M. As Algorithm 1 shows, our algorithm also uses a window propagation framework as the MMP algorithm does. It organizes windows in two data structures, namely, a priority queue $\mathcal{Q}$, which represents the wavefronts, and an array $windows[1..|E|]$, where $windows[e]$ stores the windows on edge e.

Algorithm 1 Constructing polyline-sourced GVD

Input: $M = (V,E,F)$ the triangle mesh; $\mathcal{G} = \{g_j\}_{j=1}^{m}$: the set of generators defined on M; c: the performance control parameter;

Output: the geodesic Voronoi diagram on M.

// n: the number of triangles in M;
// $\mathcal{Q}$: the priority queue containing windows on the wavefront;
// $windows[1..|E|]$: $windows[e]$ is a list storing windows on edge e;
// $marked[1..|F|]$: $marked[i]$ is a boolean value to indicate the status of face i;

```
for each generator g_i do
    Add the windows that are directly illuminated by g_i into Q;
end for
Initialize each face's marked label to false;
i ← 0;
while Q is not empty do
    i ++;
    Pop a window w from Q;
    Propagate w across its adjacent triangle to produce children window(s) {ŵ_i};
    for each ŵ_i do
        Push ŵ_i into Q and update windows for the corresponding edge;
    end for
    if i ≡ 0 mod [cn] or Q is empty then
        for each triangle t with marked[t] = false do
            if all edges of t have distances smaller than w's distance then
                if # key points > 0 or ∃g_j ∈ G, g_j ∈ t then
                    Compute the LVD and the GVD on t;
                end if
                marked[t] = true;
                Free all windows stored on t's edges;
            end if
        end for
    end if
end while
```

Initially, $\mathcal{Q}$ contains the windows that are directly illuminated by the generators. The algorithm then iteratively computes both the geodesic distance and the GVD by performing the following steps in each iteration:

1. It takes the top window w from $\mathcal{Q}$, and propagates w across its adjacent triangle to produce child window(s) $\widehat{w}_i$.

Figure 8: *Some examples of the geodesic Voronoi diagram (GVD) with polyline generators on triangle meshes. The generators, the bisectors and the iso-distance contours are drawn in red, pink and black, respectively. The background color also indicates the distance to the generators.*

2. For each child window $\widehat{w}_i$, it updates the geodesic distance of the vertex covered by $\widehat{w}$, inserts $\widehat{w}$ to $\mathcal{Q}$ and updates the window list of the corresponding edge.
3. If the current iteration number is a multiple of $[cn]$ or the priority queue is empty (i.e., when MMP algorithm terminates), check the candidate triangles and compute the LVD and GVD on them.

The algorithm repeats the above steps until the priority queue $\mathcal{Q}$ is empty. Upon termination, each edge is covered by a set of non-overlapping windows, from which one can compute the geodesic distance to arbitrary point.

Thank to the properties in Section 4, we can determine candidate triangles and compute GVD during the window propagation process in the MMP algorithm. Observe that in each iteration, the MMP algorithm propagates the window with the least geodesic distance. Thus, the distance for the top window in the priority queue $\mathcal{Q}$ is non-decreasing. Consider a triangle t. If each edge of t has a distance shorter than the distance of the current top window, we can guarantee that the triangle t has been fully covered by the geodesic wavefronts. In other words, the windows on t's sides are final, and we are ready to check whether or not t contains the GVD. Since each edge contain $O(n)$ windows, computing the exact distance from the source to the edge may be expensive. In our implementation, we measure the upper bound of the distance to edge $e = (a,b)$ by $(d(a) + d(b) + \|e\|)/2$, where $d(v)$ is the geodesic distance to vertex v and $\|e\|$ is the length of e. Our algorithm inherits the high performance of the MMP algorithm.

Property 5.1 On an n-face mesh with m generators, Algorithm 1 has an $O(Nn\log N)$ time complexity and an $O(Nn)$ space complexity, where $N = \max\{m,n\}$.

Proof See the Supplemental Material. □

There are two key issues in a practical implementation of geometric algorithms: numerical errors and degenerate cases. Since we use floating-point computation, there are two sources of numerical errors and degenerate cases: one is from the MMP algorithm itself and the other is from the GVD in each candidate triangle. For the MMP algorithm, we use the method in [LZH07] to handle degenerate cases that is also robust to numerical errors. To robustly compute the LVD $\mathcal{L}(t)$ and the GVD $\mathcal{G}(t)$ on a triangle t, note that the LVD is a combination of two types of Euclidean Voronoi diagrams: additively weighted Voronoi diagram and line-segment Voronoi diagram. We use the plane sweep algorithm in [For87] (also know as Fortune's algorithm) that can efficiently construct both types of Voronoi diagrams in a unified way. By Property 4.1, a bisector is one of three different conic curves. So we choose to use the rational quadratic Bézier curve for representing bisectors [KHP95] and use the cocktail algorithm in [KLS98] to compute the planar Bézier curve intersections. To handle the degenerate cases in constructing Voronoi diagram, we use the symbolic perturbation technique in [EM90, Yap88]. The computed LVD $\mathcal{L}(t)$ is represented as a graph. Then we compute the GVD by traversing $\mathcal{L}(t)$ using depth-first-search (DFS). Specifically, for every LVD edge (i.e., a bisector), check whether the two associated windows come from different sources; if so, put the edge and its two endpoints into the GVD edge and node sets. Upon the termination of the DFS, the GVD structure is available. More implementation details are presented in Supplementary Material.

6. Experimental Results

We implemented our algorithm in C++ and tested it on real-world models. Some examples are shown in Figure 8. The source code is available upon request and the executable program can be downloaded on the internet[†]. Timing was measured a PC with an Intel Core i7-2600 3.40GHz CPU and 8GB memory. Our algorithm adopts the parameter c to balance the performance and memory consumption. In Algorithm 1, we look for the candidate triangles every $[cn]$ iterations (see lines 13-23). A small c means that we do the checking more frequently, and thus, can identify and discard the useless windows at the early stage (see line 20). As a result, the memory consumption is low. However, frequent *in-progress* checking obviously slows down the performance.

[†] http://cg.cs.tsinghua.edu.cn/people/∼Yongjin/yongjin.htm

c	4-kid		Gargoyle		Bunny		Armadillo		Buddha	
	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.	Time	Mem.
0.5	30.18	1,013	42.25	1,561	8.50	363	17.11	751	38.66	1,597
1.0	29.65	1,034	38.43	1,593	8.09	385	16.05	782	37.45	1,678
1.5	27.98	1,115	38.39	1,721	7.91	407	15.50	844	33.36	1,743
2.0	27.02	1,153	35.59	1,783	7.73	418	15.47	876	31.01	1,798
2.5	26.94	1,186	32.73	1,824	7.59	425	14.77	887	30.22	1,870
3.0	26.69	1,217	33.79	1,882	7.21	435	14.12	889	29.66	1,904
3.5	25.91	1,245	31.87	1,910	7.13	445	14.16	914	29.39	1,964
4.0	25.53	1,276	32.29	1,919	6.94	448	13.85	943	28.82	1,979
4.5	25.02	1,305	31.52	1,971	6.84	456	13.63	948	28.71	2,006
5.0	24.85	1,311	30.44	1,984	6.80	461	13.58	954	28.42	2,024
5.5	24.72	1,345	30.34	2,001	6.73	474	13.33	961	28.04	2,055
6.0	23.91	1,353	30.62	2,005	6.71	482	13.16	969	27.57	2,060

Table 1: *The parameter c balances the memory consumption and the performance. Time is measured in seconds and memory is measured in MB.*

Model	n	Memory (MB)		Time (s)	
		$c=\infty$	$c=1$	$c=\infty$	$c=1$
Botijo	23K	67	52	0.90	1.04
Kitten	33K	98	78	1.27	1.39
Bone	40K	108	86	1.46	1.70
Horse	97K	310	230	3.66	4.76
Bunny	144K	490	385	6.36	8.20
Armadillo	346K	1,022	782	12.26	16.49
4-kid	400K	1,437	1,034	23.13	30.61
Gargoyle	700K	2,131	1,593	30.28	39.96
Buddha	800K	2,163	1,678	28.94	38.22

Table 2: *Setting $c=\infty$ leads to the best performance and the most memory consumed.*

On the other hand, a large c means less-frequent checking. An extreme case is that a sufficiently large c (e.g., $c > n$) results in a *delayed* checking, i.e., when the MMP algorithm is done. So one can achieve the best performance but with the most memory consumed. See Tables 1 and 2.

Table 3 reveals the relationship between the number of generators m and the performance/consumed memory. Increasing m to a reasonable extent usually reduces the computational cost, since each geodesic wavefront only need to cover a small portion of the surface. On the other hand, a very large m means that the GVD and the LVD are more complicated, which take more time to compute.

Below we present the comparisons of our polyline-sourced GVD algorithm with three closedly related work (i.e., the geodesic offset structure of polyline generators [BK07], the accurate GVD with point generators [LCT11] and the fast approximate GVD [XYH12]).

m	Bunny		Armadillo	
	Time (s)	Mem. (MB)	Time (s)	Mem. (MB)
2	8.81	925	15.76	1,744
8	8.93	860	16.10	1,725
32	7.20	699	16.56	1,639
128	5.72	561	15.76	1,452
512	4.95	474	13.24	1,229
2,048	5.11	427	12.11	1,072
8,192	6.76	480	14.75	1,091
32,768	8.77	692	18.02	1,203

Table 3: *The performance and memory consumption vs the number of generators m.*

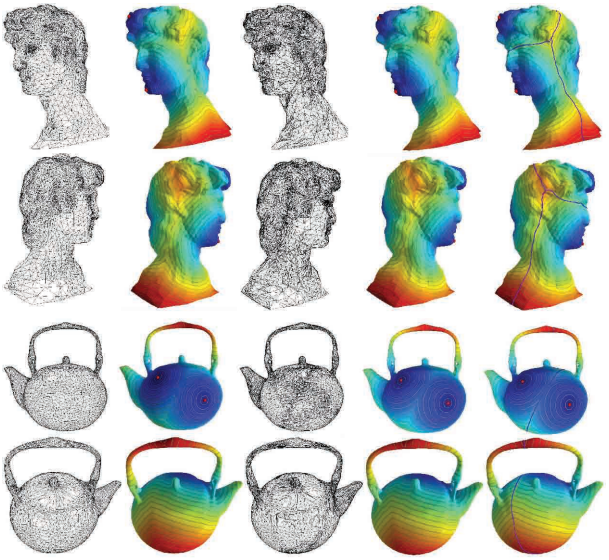


Figure 9: *Leftmost: original mesh. Middle left: original offsets. Middle: refined mesh. Middle right: refined offsets. Rightmost: GVD. Mesh refinement does not help with the construction of GVD.*

6.1. Comparison with [BK07]

As another important structure, geodesic offsets are usually created by using only distance information at vertices, and approximating the geodesic distance inside the triangles by interpolating. This method is only suitable for regular and high resolution meshes. In [BK07] an adaptive refinement scheme is proposed that re-meshes the model for the region where the interpolating generates an obvious error and smoothes the geodesic offsets.

It is quite convenient to construct the structure of Voronoi diagram from offsets in 2D with only point generators by connecting the *ridges* on the offsets. However, this is not the case in GVD and geodesic offsets. First, the complex structure of GVD, including line segments, hyperbolic segments and parabolic segments, makes the building of accurate GVD very difficult. Second, the method that creates GVD from geodesic offsets itself has an accuracy problem too: an arbitrarily accurate GVD cannot be obtained with a fixed number of geodesic offsets. These problems cannot be solved by any re-meshing method (including [BK07]) because it is an intrinsic defect of the method itself. For example, Figure 9 gives the results generated by the refinement in [BK07]. We can see that although the refinement indeed

Model	n	Our algorithm	Liu's algorithm
Kitten	33K	1.27	3.51
Horse	97K	3.66	12.74
Bunny	144K	6.36	22.47
Armadillo	346K	12.26	45.64

Table 4: *Running time comparison (in seconds) between our algorithm ($c = \infty$) and Liu's algorithm [LCT11].*

improves the appearance of the offsets, constructing accurate GVD structure from offsets cannot be achieved with the current accuracy of offsets.

6.2. Comparison with [LCT11]

Liu et al. [LCT11] proposed a practical algorithm (called Liu's algorithm in the following) to construct GVDs with point sources. Our algorithm differs from Liu's algorithm in two aspects.

First, Liu's algorithm does not apply to the case with sources of line segments. In details, a necessary preprocessing in Liu's algorithm is to subdivide triangles of mesh until the bisectors cross each face at most one time. This condition is judged by the windows on each edge (or in this paper, key points number provides this information). However, when line segments exist as generators, there are cases when the preprocessing procedure will fail.

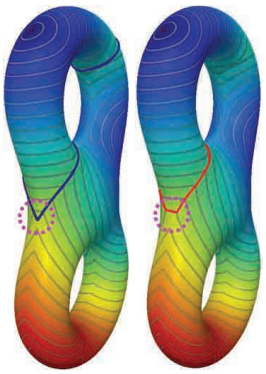
Second, our algorithm reveals the deep relationship between the structure of the MMP algorithm itself and the GVD structure. Accordingly our algorithm has a better performance than Liu's algorithm. See Table 4.

6.3. Comparison with [XYH12]

Xin et al. [XYH12] proposed an *approximate* method for constructing GVDs with point sources, which performs as follows:

1. Taking all generators as sources, compute the multi-source geodesic distance field on the subdivided mesh. Upon the termination, each vertex is labeled its nearest source (generator).
2. Identify the edges whose end points have different labels, since these edges intersect the bisectors. Use linear interpolation to approximate the intersection.
3. Construct the bisectors by checking all triangles containing at least two intersection points. If there are two intersection points, use a line segment to connect them. For the case with three intersection points, find a point inside the triangle and then connect it to all three intersections.
4. Finally, trace the bisectors to form the Voronoi cells.

This approximate algorithm is efficient, easy to implement and works fairly well for high resolution mesh with regular triangulation and a large amount of uniformly distributed sites. However, it produces very poor results on meshes with irregular triangulations or when the number of sites is small. The right inset shows a case that the exact GVD bisector has a sharp corner (see (a)), whereas the approximate bisector simply follows iso-distance contours, leading to a wrong result (see (b)). Simply improving the triangulation quality and/or increasing mesh resolution can only partially solve the problem, since there is no guarantee that tracing the cut locus can produce the correct GVD bisectors. Moreover, the overhead of remeshing and computing the geodesic would be very high. Therefore, it is highly desirable to use our proposed algorithm for computing accurate GVD on arbitrary triangle meshes.



(a) Exact bisectors (b) Approximate bisectors

6.4. Limitations

The GVD with polyline generators computed by our method is exact if numerical operations are exact. However, in our current implementation, we use floating-point computation due to its high efficiency. Although our method uses the robust implementation [KLS98] for intersection of rational quadratic Bézier curves representing bisectors [KHP95] and handle degenerate cases using the symbolic perturbation technique [EM90, Yap88], our method cannot guarantee theoretically that for the LVD retricted in each candidate triangle, the numerical results are topologically consistent with theoretically correct solutions. We will address this issue using some topology-oriented techniques in a future work.

7. Conclusion

This paper investigates the GVDs in a general setting, where the generators are polylines. We show that a typical bisector contains line segments, hyperbolic segments, and parabolic segments, therefore, computing GVD with polyline generators is more challenging than the GVD with point sources as well as the 2D Euclidean Voronoi diagrams. We introduce a new concept, called *local Voronoi diagram*, or LVD, which is a combination of weighted and line-segment Euclidean Voronoi diagrams. We show that when restricting on a mesh triangle, the GVD is a subset of the LVD, which can be computed by using the existing 2D Euclidean techniques. Moreover, only two types of mesh faces can contain GVD edges. Guided by these results, we propose an efficient algorithm for constructing the exact GVD with polyline generators. Our algorithm distinguishes with the existing GVD work in

that it is integrated into the MMP framework seamlessly: once the MMP algorithm terminates, both the geodesic distance and the GVD are readily available. We demonstrate the efficacy of our method on real-world models.

Acknowledgement

This work was supported by the Natural Science Foundation of China (61322206, 61272228), the National High Technology Research and Development Program of China (2012AA011802), Tsinghua University Initiative Scientific Research Program (20131089252) and Beijing Higher Institution Engineering Research Center of Visual Media Intelligent Processing and Security. Ying He is supported by Singapore MOE Grants RG40/12 and MOE2013-T2-2-011.

References

[AP85] AUGENBAUM J. M., PESKIN C. S.: On the construction of the Voronoi mesh on a sphere. *Journal of Computational Physics 59*, 2 (1985), 177–192.

[BDG13] BOISSONNAT J.-D., DYER R., GHOSH A.: Constructing intrinsic Delaunay triangulations of submanifolds. *Research Report RR-8273, INRIA arXiv: 1303.6493* (2013).

[BK07] BOMMES D., KOBBELT L.: Accurate computation of geodesic distance fields for polygonal curves on triangle meshes. In *Proceedings of the Vision, Modeling, and Visualization Conference (VMV '07)* (2007), pp. 151–160.

[CH90] CHEN J., HAN Y.: Shortest paths on a polyhedron. In *Proceedings of the Sixth Annual Symposium on Computational Geometry* (1990), pp. 360–369.

[CJL13] CHAMBERS E. W., JU T., LETSCHER D.: Medial residues of piecewise linear manifolds. In *Proc. Canadian Conference on Computational Geometry (CCCG'13)* (2013), p. Session 1A.

[CWW13] CRANE K., WEISCHEDEL C., WARDETZKY M.: Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics 32*, 5 (2013), 152.

[EM90] EDELSBRUNNER H., MÜCKE E. P.: Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph. 9*, 1 (Jan. 1990), 66–104.

[For87] FORTUNE S.: A sweepline algorithm for Voronoi diagrams. *Algorithmica 2*, 1-4 (1987), 153–174.

[FS07] FORT M., SELLARES J. A.: Generalized source shortest paths on polyhedral surfaces. In *Proceedings of the 23rd European Workshop on Computational Geometry* (2007), pp. 186–189.

[KHP95] KIM D.-S., HWANG I.-K., PARK B.-J.: Representing the Voronoi diagram of a simple polygon using rational quadratic Bézier curves. *Computer-Aided Design 27*, 8 (1995), 605–614.

[KKB98] KIMMEL R., KIRYATI N., BRUCKSTEIN A. M.: Multivalued distance maps for motion planning on surfaces with moving obstacles. *IEEE Transactions on Robotics and Automation 14*, 3 (1998), 427–436.

[KLS98] KIM D.-S., LEE S.-W., SHIN H.: A cocktail algorithm for planar Bézier curve intersections. *Computer-Aided Design 30*, 13 (1998), 1047–1051.

[KS98] KIMMEL R., SETHIAN J.: Computing geodesic paths on manifolds. *Proceedings of National Academy of Sciences 95* (1998), 8431–8435.

[KS99] KIMMEL R., SETHIAN J. A.: Fast Voronoi diagrams and offsets on triangulated surfaces. In *Proc. of AFA Conf. on Curves and Surfaces* (1999).

[KWR97] KUNZE R., WOLTER F.-E., RAUSCH T.: Geodesic Voronoi diagrams on parametric surfaces. In *Computer Graphics International* (1997), vol. 97, pp. 230–237.

[LCT11] LIU Y.-J., CHEN Z., TANG K.: Construction of iso-contours, bisectors, and Voronoi diagrams on triangulated surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*, 8 (2011), 1502–1517.

[Liu13] LIU Y.-J.: Exact geodesic metric in 2-manifold triangle meshes using edge-based data structures. *Computer-Aided Design 45*, 3 (2013), 695–704.

[LLW12] LU L., LÉVY B., WANG W.: Centroidal Voronoi tessellation of line segments and graphs. *Comp. Graph. Forum 31*, 2pt4 (May 2012), 775–784.

[LT13] LIU Y.-J., TANG K.: The complexity of geodesic Voronoi diagrams on triangulated 2-manifold surfaces. *Information Processing Letters 113*, 4 (2013), 132–136.

[LWL*09] LIU Y., WANG W., LEVY B., SUN F., YAN D.-M., LU L., YANG C.: On centroidal Voronoi tessellation – energy smoothness and fast computation. *ACM Transactions on Graphics 28*, 4 (2009), Article No. 101.

[LXHK14] LIU Y.-J., XU C.-X., HE Y., KIM D.-S.: The duality of geodesic Voronoi/Delaunay diagrams for an intrinsic discrete laplace-beltrami operator on simplicial surfaces. In *Proc. Canadian Conference on Computational Geometry (CCCG'14)* (2014), pp. Session 4A, paper 19.

[LZH07] LIU Y.-J., ZHOU Q.-Y., HU S.-M.: Handling degenerate cases in exact geodesic computation on triangle meshes. *The Visual Computer 23*, 9-11 (2007), 661–668.

[MMP87] MITCHELL J. S., MOUNT D. M., PAPADIMITRIOU C. H.: The discrete geodesic problem. *SIAM Journal on Computing 16*, 4 (1987), 647–668.

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

[OBSC00] OKABE A., BOOTS B., SUGIHARA K., CHIU S. N.: *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, 2000.

[OT96] ONISHI K., TAKAYAMA N.: Construction of Voronoi diagram on the upper half-plane. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 79*, 4 (1996), 533–539.

[SSK*05] SURAZHSKY V., SURAZHSKY T., KIRSANOV D., GORTLER S. J., HOPPE H.: Fast exact and approximate geodesics on meshes. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, pp. 553–560.

[XYH11] XIN S.-Q., YING X., HE Y.: Efficiently computing geodesic offsets on triangle meshes by the extended xin-wang algorithm. *Comput. Aided Des. 43*, 11 (2011), 1468–1476.

[XYH12] XIN S.-Q., YING X., HE Y.: Constant-time all-pairs geodesic distance query on triangle meshes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '12)* (2012), pp. 31–38.

[Yap88] YAP C. K.: A geometric consistency theorem for a symbolic perturbation scheme. In *Proceedings of the Fourth Annual Symposium on Computational Geometry* (1988), SCG '88, pp. 134–142.

[YWH13] YING X., WANG X., HE Y.: Saddle vertex graph (SVG): a novel solution to the discrete geodesic problem. *ACM Transactions on Graphics (SIGGRAPH ASIA 2013) 32*, 6 (2013), 170:1–170:12.