This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier’s archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright
Applying the improved Chen and Han's algorithm to different versions of shortest path problems on a polyhedral surface

Shi-Qing Xin, Guo-Jin Wang *
Department of Mathematics and State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou 310027, PR China

A R T I C L E   I N F O
Article history:
Received 9 July 2009
Accepted 20 May 2010

Keywords:
Computational geometry
Discrete geodesic problem
Voronoi diagram

A B S T R A C T
The computation of shortest paths on a polyhedral surface is a common operation in many computer graphics applications. There are two best known exact algorithms for the "single source, any destination" shortest path problem. One is proposed by Mitchell et al. (1987) [1]. The other is by Chen and Han (1990) [11]. Recently, Xin and Wang (2009) [9] improved the CH algorithm by exploiting a filtering theorem and achieved a practical method that outperforms both the CH algorithm and the MMP algorithm whether in time or in space.

In this paper, we apply the improved CH algorithm to different versions of shortest path problems. The contributions of this paper include: (1) For a surface point \( p \in \Delta v_1v_2v_3 \), we present an unfolding technique for estimating the distance value at \( p \) using the distances at \( v_1, v_2 \) and \( v_3 \). (2) We show that the improved CH algorithm can be naturally extended to the "multiple sources, any destination" version. Also, introducing a well-chosen heuristic factor into the improved CH algorithm will induce an exact solution to the "single source, single destination" version. (3) At the conclusion of multi-source shortest path algorithms, we can use the distance values at vertices to approximately compute the geodesic-distance-based offsets, the Voronoi diagram and the Delaunay triangulation in \( O(n) \) time. (4) By importing a precision parameter \( \lambda \), we obtain a precision controlled approximant which varies from the improved CH algorithm to Dijkstra’s algorithm as \( \lambda \) increases from 0 to 1. Thus, an interesting relationship between them can be naturally established.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

The discrete geodesic problem [1], i.e., finding shortest paths between points on a polyhedral surface, arises in natural applications such as robotics, motion planning, geographic information systems and navigation [2,3] and has been widely studied in computational geometry [4–6]. The discrete geodesic problem is also central to computer graphics because the computation of shortest paths is a common operation in many graphics problems [7].

According to different application requirements, the discrete geodesic problem may have at least the following four versions: “single source, any destination”, “multiple sources, any destination”, “single source, single destination”, and “any source, any destination”. The most important of these versions is the “single source, any destination” shortest path problem since it can be theoretically viewed as a kernel of the other problems if the goal is to find the exact solutions [8,9]. Several algorithms [1,8–11] have been proposed for this problem. In 1987, Mitchell, Mount and Papadimitriou [1] (MMP) gave an \( O(n^2 \log n) \)-time and \( O(n^2) \)-space algorithm that inherits the paradigm of Dijkstra’s algorithm [12], where \( n \) is the number of faces. Later in 1990, Chen and Han [11] (CH) provided an \( O(n^2) \)-time and \( O(n) \)-space algorithm based on a key observation of “one angle one split”. Interestingly, all the available experimental results [13,7,9] show that the MMP algorithm, of a higher time complexity, runs more efficiently in practice but requires higher space costs. Recently, Xin and Wang [9] (XW) improved the CH algorithm by filtering out those discrete events that will never contribute to any shortest path, achieving a new algorithm that outperforms both the CH algorithm and the MMP algorithm whether in time or in space: see Section 3.

In fact, the MMP algorithm [1], the CH algorithm [11] and the improved CH algorithm [9] all consist of two phases. In the first phase, the shortest path from the source to each vertex is computed, along with a set of windows encoding the shortest paths from the source to points on the edges. In the second phase, the windows are used to compute a decomposition of the polyhedral surface which permits a shortest path to any destination can be reported in \( O(\log n) \) time [8,1,11]. However, computing a decomposition of the surface is complicated and unnecessary. Surazhsky et al. [7] suggested that for a surface point \( p \in \Delta v_1v_2v_3 \), one can consider all windows on the three edges bounding the face, and choose the window that provides the shortest distance to point \( p \).
This technique also applies to the original CH algorithm [11] and the improved CH algorithm [9], but increases the processing complexity from $O(n)$ to $O(n^2)$. In this paper, we suggest using the distances at $v_1$, $v_2$ and $v_3$ to estimate the distance value at $p \in \Delta v_1 v_2 v_3$. The technique is taking the geodesic distance as the straight line distance and thereby estimating the geodesic distance of $p$ on the unfolded plane. Numerous experimental results show that the estimation is very close to the exact value, and the relative error is generally not more than 0.1%. Section 4 gives the details.

As in Dijkstra's algorithm, finding the exact solution to "single source, single destination" is not easier than "single source, any destination". In order to make a tradeoff between precision and performance when computing the geodesic distance between two points $s, t$ on the surface, researchers presented numerous approximation algorithms [3,6,14–23]. In fact, we can define a heuristic factor $\mathcal{H}(p) = D_{\text{geodesic}}(s, p) + D_{\text{Euclidean}}(p, t)$ for a point $p$, where $D_{\text{geodesic}}(\cdot)$ is the geodesic distance measure and $D_{\text{Euclidean}}(\cdot)$ is the Euclidean distance measure. This optimizes the processing order of the discrete events, greatly narrows the search space and therefore speeds up the improved CH algorithm [9]. Since the $\mathcal{H}(p)$ is always less than or equal to $D_{\text{geodesic}}(s, t)$, we can conclude that the resulting solution is exact; see Section 5.

Leibon and Letscher [24] proved that the geodesic-distance-based Voronoi diagram (GVD), as well as the geodesic-distance-based Delaunay triangulation, are well defined on a manifold if the sampling points are sufficiently dense. However, the computation of an exact GVD is difficult and known approximate algorithms for GVD computation are time consuming [25,26]. This motivates us to seek for an approximation method. In this paper, we approximately compute the geodesic based offsets, the geodesic based Voronoi diagram and the geodesic based Delaunay triangulation by estimating the locus in a face $f$ using the distance values at the three vertices of $f$. This enables us to quickly report these geometric structures in $O(n)$ time after running the improved CH algorithm [9]. Fig. 1 exhibits the results by our algorithm. Section 6 describes this idea in detail.

Although Xin and Wang [9] improved Chen and Han's algorithm [11], finding exact solutions still requires a high time cost for very large models. Since the input mesh is just an approximant of the real surface, approximation algorithms [16,19,20,3,22,7], especially the Fast Marching Method [2] and Dijkstra's algorithm [12], are preferred in many occasions. Observing that the key idea of the improved CH algorithm [9] is filtering out useless discrete events, we import a parameter $\lambda \in [0, 1]$ into the filtering theorem [9] to achieve an over-filtering rule. This induces a precision controlled approximant which varies from the improved CH algorithm to Dijkstra's algorithm as $\lambda$ increases from 0 to 1. Thus, an interesting relationship between them can be naturally established; see Section 7.

Note that all the experiments are made on a computer with the following configuration:

- Windows XP Professional x64 Edition,
- Intel(R) Xeon(R) CPU,
- W3520 @ 2.67 GHz,
- 12.0 GB of RAM.

The experimental results are organized into different sections. Section 8 gives the final conclusion.

2. Geometric preliminaries

Let $\delta$ be a (triangulated) polyhedral surface in $\mathbb{R}^3$, defined by a set of faces, edges and vertices. Without loss of generality, we assume that $\delta$ has a complexity of $n$ since Euler's formula affirms that the number of vertices, edges and faces of a polyhedral surface are linearly related [27].

Suppose $\Pi$ is a path restricted on the surface $\delta$ and this path goes through a sequence of vertices and edges. We call this sequence $\Gamma$. The following properties are known [8,11,11]:

P1: If $\Pi$ is a shortest path only if the sequence $\Gamma$ is simple, i.e., no edge or vertex appears more than once in $\Gamma$.

P2: If $\Pi$ is a shortest path, then its intersection with each face in $\Gamma$ is a line segment.

P3: A shortest path $\Pi$ cannot pass more than $n$ faces, where $n$ is the total number of faces of the polyhedral surface $\delta$.

P4: Both of the angles of a shortest path $\Pi$ passing through a vertex $v$ are greater than or equal to $\pi$. So $v$ must be a saddle vertex, where "saddle" means that the sum of the incident angles to $v$ is greater than $2\pi$.

P5: Let $\Pi_1, \Pi_2$ be two shortest paths between the source point $s$ and the destination point $t$. $\Pi_1, \Pi_2$ cannot intersect unless at or at a saddle vertex.

P6: If $\Pi$ is a shortest path, it can be unfolded into a polyline when $\Gamma$ is unfolded onto a plane, where a saddle vertex $v$ on $\Pi$ contributes to a corner $v'$ of the polyline. We call $v'$ the unfolded image of $v$.

In studying shortest path problem, we need to define a well-designed data structure called a window. A window encodes a set of shortest paths that goes through the same sequence $\Gamma$ of vertices and faces [11,9]. Since a typical edge–vertex sequence $\Gamma$ ends with a vertex or an edge, we classify windows into two types: (1) pseudo-source windows and (2) interval windows; see Fig. 2. Technically, we use a pair $(d, v)$ to denote the key information encoded in a pseudo-source window at vertex $v$, where $d$ is the shortest distance from the source to $v$ restricted on the sequence $\Gamma$. If $\Gamma$ ends with an edge $e$, we call the last vertex $r$ in $\Gamma$ a root.

Sharir and Schorr [8] proved that the point set

$$\{p | p \in e, \text{ the shortest path from } r \text{ to } p \}$$

can be unfolded into a straight line segment.
3. The original CH algorithm and the improved version

The CH algorithm [11] builds a tree, level by level, to arrange windows that possibly contribute to the determination of a shortest path. The basic rule of creating children of window \( w \) is summarized below.

- **If** \( w \) is a pseudo-source window at vertex \( v \), i.e., \( w := (d, v) \)
- **If** \( v \) is saddle vertex (see Fig. 3(a))
  - For each vertex \( v' \) adjacent to \( v \), compute a pseudo-source window;
  - For each edge opposite to \( v \), compute an interval window;
  - Else \("the sum of angles incident to vertex v is equal to or less than 2\pi\")
    - No children need to be computed;
  - Else \("let w := (d, I, e, [a, b])\) and \( v \) be opposite to edge \( \overline{te} \)
    - **If** the line segment \( \overline{tv} \) is right to the interval \([a,b]\) (see Fig. 3(b))
      - Compute the only interval-window child on edge \( \overline{te} \);
    - Elseif the line segment \( \overline{tv} \) is left to the interval \([a,b]\) (see Fig. 3(c))
      - Compute the only interval-window child on edge \( \overline{te} \);
    - Else \("\overline{tv}\) intersects the interval \([a,b]\) at some point (see Fig. 3(d))."
      - Compute two interval windows, one on edge \( \overline{te} \) and the other on edge \( \overline{et} \);
      - Compute a pseudo-source window at vertex \( v \).

Note that Property (P3) ensures that the tree cannot have more than \( n \) levels and Property (P4) means that we do not need to compute children for a convex vertex. Chen and Han [11] made a clever observation of "one angle one split"; see Lemma 1. Their algorithm thereby avoids the explosion of the tree, improving the discrete geodesic problem to be \( O(n^2) \)-time and \( O(n) \)-space. However, Surazhsky et al. [7] provided experimental results to show that the MMP algorithm [1], of an \( O(n^2 \log n) \) time complexity, runs many times faster in practice. Recently, Xin and Wang [9] observed that the overwhelming majority of the windows created by the CH algorithm [11] never contribute to a real shortest path. So they [9] improved the CH algorithm by two separate techniques. One filters out useless windows using the current estimates of the distances to the vertices. The other maintains a priority queue like that achieved in Dijkstra’s algorithm. The improved CH algorithm [9] outperforms the CH algorithm and the MMP algorithm.

In the next section, we will apply the improved CH algorithm to other versions of shortest path problems.

**Lemma 1** ([11]). Let \( w_1 \) and \( w_2 \) be two windows on the same directed edge \( \overline{tv} \) of \( \Delta v_1v_2v_3 \), shown in Fig. 4(a). If both of the windows cover the vertex \( v \), then at most one of them can have two children which could be used to define a shortest sequence.

**Lemma 2** ([9]). Let \( w \) be a window that enters \( \Delta v_1v_2v_3 \) through edge \( \overline{tv} \), shown in Fig. 4(b). Assume that \( d_1, d_2, d_3 \) are respectively the minimum-so-far distance at the three vertices \( v_1, v_2, v_3 \). \( w \) cannot define a shortest sequence if

\[
d + \|IB\| > d_1 + \|v_1B\|
\]

or

\[
d + \|IA\| > d_2 + \|v_2A\|
\]

or

\[
d + \|IA\| > d_3 + \|v_3A\|
\]
4. Estimating the geodesic distance to a surface point

All the exact algorithms [8,4,11,9] for the "single source, any destination" shortest path problem consist of two phases. In the first phase, the shortest path from the source to each vertex is computed, along with a set of windows encoding information about the shortest paths from the source to points on the edges. In the second phase, the windows are used to compute a decomposition of the polyhedral surface, with which a shortest path to any destination can be reported in $O(\log n)$ time. However, computing the decomposition is complicated to implement and is generally unnecessary. For a surface point $p$, Surazhsky et al. [7] suggested considering all windows on the three edges bounding the face $f$ and choosing the window that provides the shortest distance to $p$. The technique also applies to the improved CH algorithm [9]. However, the space complexity increases accordingly from $O(n)$ to $O(n^2)$. In this section, we propose a rather simple idea, according to the distance values at the three vertices of $f$, for estimating the distance to a surface point $p$ in the face $f$.

As Fig. 5 shows, the exact geodesic distances to the vertices $v_1, v_2, v_3$ are known and we herein approximate the geodesic distance to the surface point $p \in \triangle v_1 v_2 v_3$. If the source point $s$ and the surface point $p$ are located in the same face, we just need to return the Euclidean distance between them. Otherwise, we consider approximately "unfolding" the geodesic path $\Pi(s, p)$ respectively along the edges $v_1v_2, v_2v_3, v_3v_1$. Taking Fig. 5 for an example, we first "unfold" $\Pi(s, p)$, along $v_1v_2$, onto the plane of
5. The “single source, single destination” problem

In this section we discuss how to heuristically compute the exact shortest path between two points \( s \) and \( t \). Generally speaking, heuristic algorithms are used to find an approximate solution close to the best one among the search space. However, if the heuristic factor satisfies the following two conditions, then the heuristic algorithm results in an exact solution:

- \( \mathcal{H}(a, b) \leq D(a, b) \) for any \( a, b \) in the definition domain;
- \( \mathcal{H}(a, a) = 0 \) for any \( a \) in the definition domain,

where \( \mathcal{H}(\cdot, \cdot) \) is the heuristic factor for estimating the exact measure \( D(\cdot, \cdot) \) of interest. Here, we construct a heuristic factor \( \mathcal{H}_e(s, t) = D_{\text{geodesic}}(s, p) + D_{\text{Euclidean}}(p, t) \) regarding the point \( p \) for estimating the shortest distance between the source point \( s \) and the destination point \( t \), where \( D_{\text{geodesic}}(\cdot, \cdot) \) is the geodesic distance measure and \( D_{\text{Euclidean}}(\cdot, \cdot) \) is the Euclidean distance measure. Obviously, \( \mathcal{H}_e(\cdot, \cdot) \) satisfies the above two conditions and therefore gives the exact shortest path between \( s \) and \( t \). Fig. 7 shows a variant of such a definition:

\[
\mathcal{H}(w) = \begin{cases} 
  d + \min_{p \in [s, b]} (||p|| + ||pt||), & w = (d, e, [a, b]), \\
  d + ||pt||, & w = (d, v).
\end{cases}
\]

Like that achieved in the improved CH algorithm [9], we maintain a priority queue throughout the algorithm such that the window with the minimum \( \mathcal{H}(\cdot) \) is first handled. If the minimum \( \mathcal{H}(\cdot) \) is larger than the current distance at the destination \( t \), the algorithm terminates and returns the shortest distance between \( s \) and \( t \).

Experimental results in Fig. 8 and Table 2 show that if the source and the destination are already known, the heuristic technique will greatly narrow the search space and thus reduce the time cost.
The approximate offsets at $d_1$ and $d_2$ and the given $d$ is between $d_1$ and $d_2$, then points $p_1$, $p_2$ on the offset of geodesic distance $d$ are approximately computed by $\parallel p_1d_1 \parallel = \frac{d_1-d}{k}$ and $\parallel p_2d_2 \parallel = \frac{d_2-d}{k}$; (b) If $d_1$, $d_2$, $d_3$ are given by 2 different sources $s_1$, $s_2$, then the Voronoi edge $\parallel p_1p_2 \parallel$ is determined by $d_1 + \parallel p_1s_1 \parallel = d_2 + \parallel p_2s_2 \parallel$ and $d_1 + \parallel p_1s_1 \parallel = d_3 + \parallel p_3s_3 \parallel$; (c) If $d_1$, $d_2$, $d_3$ are given by 3 sources, then 3 Voronoi edges are determined by $d_1 + \parallel p_1s_1 \parallel = d_2 + \parallel p_2s_2 \parallel, d_1 + \parallel p_1s_1 \parallel = d_3 + \parallel p_3s_3 \parallel$ and $d_2 + \parallel p_2s_2 \parallel = d_3 + \parallel p_3s_3 \parallel$; (d) The Delaunay edge is defined by minimizing $D_{\text{geodesic}}(s_i) + D_{\text{geodesic}}(s_j) + D_{\text{euclidean}}(v_i, v_j)$, where $\parallel p_1p_2 \parallel$ is an edge.

### Table 1

<table>
<thead>
<tr>
<th>Path number</th>
<th>$\parallel (%)$</th>
<th>$\parallel (%)$</th>
<th>$\parallel (%)$</th>
<th>$\parallel (%)$</th>
<th>$\parallel (%)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra’s method</td>
<td>4.684</td>
<td>4.724</td>
<td>6.545</td>
<td>5.281</td>
<td>4.358</td>
</tr>
<tr>
<td>Our method</td>
<td>0.024</td>
<td>0.0</td>
<td>0.070</td>
<td>0.008</td>
<td>0.029</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Model</th>
<th>Faces</th>
<th>The improved CH algorithm</th>
<th>The heuristic algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time (ms)</td>
<td>Space (Mb)</td>
</tr>
<tr>
<td>Gargoyle</td>
<td>20,000</td>
<td>390</td>
<td>0.31</td>
</tr>
<tr>
<td>Feline</td>
<td>99,732</td>
<td>2,687</td>
<td>1.52</td>
</tr>
</tbody>
</table>

6. The geodesic based offsets, the Voronoi diagram and the Delaunay triangulation

As is shown in Fig. 9(a), it is trivial to extend the improved CH algorithm [9] to the problem of “multiple sources, any destination”, since both Lemmas 1 and 2 still make sense. Based on this, we give a set of approximation algorithms for computing the geodesic based offsets, the Voronoi diagram and the Delaunay triangulation. The common technique is to estimate the trajectory of these geometric elements using the distance values at the vertices.

6.1. The geodesic based offsets

Here we discuss how to approximately compute the offsets. Suppose that $d_1$, $d_2$ are respectively the shortest distances to the endpoints of edge $\parallel p_1p_2 \parallel$. For any $d \in [\min(d_1, d_2), \max(d_1, d_2)]$, there is always one or more points $p \in \parallel p_1p_2 \parallel$ such that $p$ has the shortest distance $d$. For refined meshes that are evenly triangulated, it is reasonable to assume: (1) the distance to $p$ linearly changes between $d_1$ and $d_2$ when $p$ varies from $v_1$ to $v_2$, and (2) an offset crosses a triangle $f$ by a straight line segment. We can approximately compute the offsets by traversing all the triangles.

In most cases we hope to backtrace a family of the offsets at distance $D$ divided by $k$, $D$ divided by $k$, $\ldots$, $D$ divided by $k$.

where $D = \max |d_i|$, $d_i$ is the shortest distance to $v_j$ and $k$ is a prescribed constant number. For $\Delta v_1v_2v_3$ shown in Fig. 10(a), without loss of generality, we assume $d_1 < d_2 < d_3$, where $d_i$ is the shortest distance to $v_i$, $\Delta v_1v_2v_3$ must contribute to the offsets at distance $d = j \frac{D}{k}$, $j \in \mathbb{N}$ and $k \frac{d_1}{D} < j \leq k \frac{d_1}{D}$.

For every $k \frac{d_1}{D} < j \leq k \frac{d_1}{D}$ we need to compute a straight line segment $p_1p_2$ to represent the offset at distance $d = j \frac{D}{k}$ lying in $\Delta v_1v_2v_3$. If $d_1 < d \leq d_2$, we compute the two points $p_1 \in \parallel p_1p_2 \parallel, p_2 \in \parallel p_1p_2 \parallel$ by solving

$$
\begin{align*}
\parallel p_1d_1 \parallel &= \parallel p_1d_2 \parallel \\
\parallel d - d_1 \parallel &= \parallel d_2 - d \parallel \\
\parallel d - d_1 \parallel &= \parallel d_3 - d \parallel,
\end{align*}
$$

and if $d_2 < d \leq d_3$, we compute $p_1, p_2$ by

$$
\begin{align*}
\parallel p_1d_1 \parallel &= \parallel p_1d_2 \parallel \\
\parallel d_1 - d_2 \parallel &= \parallel p_2d_3 \parallel \\
\parallel d_1 - d_2 \parallel &= \parallel d_3 - d \parallel.
\end{align*}
$$

Fig. 9(a) shows an example of the approximate geodesic based offsets. We can prove the following theorem:

**Theorem 3.** The approximate offsets at $d_1$ and the approximate offsets at $d_2$ do not intersect each other if $d_1 \neq d_2$. The approximate offsets at distance $d$ on a closed polyhedral surface are closed.

6.2. The Voronoi diagram

Leibon and Letscher [24] discussed the existence of the geodesic-distance-based Voronoi diagram (GVD) on a manifold. If
we take a source point as a site, then each site occupies an area, called a cell. Different cells do not overlap and the union of all the cells is just the whole surface. Exact algorithms for this problem are not available yet. Here we give a very simple approximation algorithm that runs in \(O(n)\) time. Without loss of generality, we assume that every face contains at most one source point. Our idea comes from the fact that if the shortest distances to the endpoints of edge \(Pv\) are provided by different sites \(s_1, s_2\), then \(Pv\) belongs to at least two cells. With the assumption that the input model is refined and evenly triangulated, we can divide the edge \(Pv\), at the point \(p\), into two parts, one in the cell of \(s_1\) and the other in the cell of \(s_2\). In other words, \(p\) is regarded to be located on the common boundary of \(s_1\)’s cell and \(s_2\)’s cell. We can approximately compute \(p\) by solving the equation

\[ d_1 + ||Pv|| = d_2 + ||Pv||, \]

where \(d_1, d_2\) are respectively the shortest distances to \(v_1, v_2\). It is easy to prove that the equation has one and only one solution \(p^*\) \([8]\). If the three distances at \(v_1, v_2, v_3\) of \(\Delta v_1v_2v_3\) are given by the same site \(s\), then we can assume that the face belongs to the cell of \(s\) and no cell boundary crosses the face. If the three distances are given by two different sites \(s_1, s_2\) as Fig. 10(b) shows. Then we can compute \(p_1 \in \theta Pv\) and \(p_2 \in \|Pv\|\) according to the above equation and take \(\theta Pv\) as the common boundary of \(s_1\)’s cell and \(s_2\)’s cell. Otherwise, the three distances are given by three different sites \(s_1, s_2, s_3\). As Fig. 10(c) shows, we compute \(p_1 \in \theta P_{v_1}, p_2 \in \theta P_{v_2}, p_3 \in \theta P_{v_3}\) according to the above equation, and the barycenter \(p\) of \(p_1, p_2, p_3, \theta P_{v_1}, \theta P_{v_2}, \theta P_{v_3}\) can be viewed as an approximant of the Voronoi diagram in the face \(\Delta v_1v_2v_3\). The whole approximate Voronoi diagram on the surface can be computed in \(O(n)\) time. Fig. 9(c) shows an example and we have the following observations:

1. For each site \(s\), the site itself belongs to its cell.
2. Each point on the surface is associated with a site; see Fig. 10(b) and (c).
3. The boundaries of the approximate Voronoi cells intersect an edge \(e\) at no more than one point, so it is impossible that an edge intersects a Voronoi cell at an entry point and an exit point. If both the endpoints of \(e\) belong to the approximate Voronoi cell of \(s\), then the whole edge belongs to the cell.
4. If a simple closed curve \(C\) lying on the surface does not enclose a vertex inside, then the curve \(C\) must be restricted on an edge sequence \(E\) and crosses each edge in \(E\) at least twice.

Sometimes when the input mesh is unevenly triangulated, there may be some “long” edges. Here we give a definition to describe these long edges.

**Definition 4.** Suppose that one endpoint of the edge \(e\) achieves the shortest distance from the site \(s_1\) and the other achieves the shortest distance from the site \(s_2\) (possibly \(s_1 = s_2\)). The edge \(e\) is called an XL edge if there is a point \(p \in e\), whose shortest distance is given by the site \(s\), such that \(s \neq s_1\) and \(s \neq s_2\).

Having Definition 4, we will prove the following theorem.

**Theorem 5.** Given a set of sites on a polyhedral surface that is a manifold and contains no XL edges, our approximation algorithm computes a simply connected cell for each site. The union of all the cells covers the surface, and different cells do not overlap.

**Proof.** First, if the cell of the site \(s\) is not simply connected, then its cell is composed of at least two simply connected areas \(A_1, A_2\); see Fig. 11(a). Without loss of generality, we assume that the area enclosing the site \(s\) is \(A_1\) and the other is \(A_2\). Suppose that \(p_2 \in A_2\) provides the shortest path \(\Pi (s, A_2)\) from \(s\) to \(A_2\). \(p_2\) must be located on the boundary of \(A_2\). Let \(p_1\) be the (farthest) intersection point where \(\Pi (s, A_2)\) intersects the boundary of \(A_1\). The section of \(\Pi (s, A_2)\) between \(p_1 \) and \(p_2\) denoted by \(\Pi (p_1, p_2)\), is outside \(A_1\) and \(A_2\). We assume that \(\Pi (p_1, p_2)\) crosses the face sequence \(f_1, f_2, \ldots, f_k\) or alternatively the edge sequence \(e_1, e_2, \ldots, e_k\). One of the three vertices of \(f_1\) must be inside \(A_1\). Otherwise, Observation 3 ensures that \(p_1\) is inside \(A_1\) rather than on the boundary. Without loss of generality, we assume that the lower endpoint of \(e_1\) is inside \(A_1\) while the upper endpoint is outside. This means that the shortest distance to the upper endpoint of \(e_1\) cannot be contributed by the site \(s\). Considering that no XL edges exist, we conclude that the lower endpoint \(v_2\) of the edge \(e_2\) gets the shortest distance from the site \(s\). Similarly we know that \(s\) provides the shortest distances to the lower endpoints \(v_3, v_4, \ldots, v_h\) of the edge sequence \(e_3, e_4, \ldots, e_{h+1}\). According to Observation (3), we have that all the edges \(\theta v_{i+1}\) (1 \(\leq i \leq h - 1\)) lie inside approximate Voronoi cell of \(s\) and therefore \(A_1\) and \(A_2\) cannot be two separate approximate Voronoi cells.

Second, if two approximate Voronoi cells \(A_1\) and \(A_2\) overlap, then the intersection points must be vertices, as Fig. 11(b) shows. We assume that two of the intersection points are \(v_1, v_2\). If there is a vertex \(v_3\) inside \(A_1 \cap A_2\), then it can be concluded that \(v_3\) must be located on the common boundary of \(A_1\) and \(A_2\). Therefore, the boundary of \(A_1 \cup A_2\) lies on an edge sequence \(E\) and cross each edge in \(E\) at least twice. According to Observation (4), this contradicts our approximation algorithm for computing Voronoi cells. □

6.3. The Delaunay triangulation

In order to generate an approximate Delaunay triangulation on a polyhedral surface, we introduce a definition here.

**Definition 6.** If there is an edge \(\theta v\) such that the site \(s_2\) provides the shortest distance to \(v_1\) and the site \(s_3\) \((s_1 \neq s_2)\) provides the shortest distance to \(v_2\), there is a candidate Delaunay edge that is composed of 3 sections — the shortest path from \(s_1\) to \(v_1\), the edge \(\theta v\) and the shortest path from \(v_2\) to \(s_2\). The approximate Voronoi edge is chosen by minimizing the candidates’ lengths \(D_{\text{geodesic}}(S_1, v_1) + D_{\text{Ludicmean}}(v_1, v_2) + D_{\text{geodesic}}(v_2, S_2)\).

Fig. 9(c) exhibits an example of the approximate Delaunay triangulation according to the above definition. We further have the following theorem.

**Theorem 7.** Given a set of sites on a polyhedral surface that is a manifold and contains no XL edges, no two Delaunay edges cross each other in the resulting approximate Delaunay triangulation. The site \(s_1\) and the site \(s_2\) are joined by a Delaunay edge if and only if they have a section of common boundary.

**Proof.** As Fig. 12(a) shows, we assume that two approximate Delaunay edges intersect at a point \(p \in \theta P_{v_1}\). We have \(p \neq \theta P_{v_2}\). \(s_1 \neq s_3\) and \(s_2 \neq s_3\). Since the shortest distance to \(v_3\) is given by the site \(s_3\), the shortest distance to \(p\) must also be given by \(s_3\). This means that the edge \(\theta P_{v_2}\) is an XL edge, contradicting the given conditions. Another case we need to consider is shown in Fig. 12(b). According to the Property (P5) in Section 2, the intersection point \(p\) is a saddle vertex and \(s_2, s_3\) provide the shortest distance to \(p\) at the same time. Recall that in the improved CH algorithm we just keep one of the vertices that provide the shortest distance to \(p\) at the same time. Without loss of generality, we assume that \(s_2\) gives the shortest distance to \(p\) as well as the shortest distance to \(v_3\). \(s_2\) cannot be an approximate Delaunay edge according to Definition 6. The latter half is obvious from our approximation algorithms. □
6.4. Removing Delaunay edges that pass through XL edges

From Theorems 5 and 7, we know that the approximation algorithms for computing the Voronoi diagram and the Delaunay triangulation do not work if there exist some XL edges. As Fig. 13(a) shows, the edge \( \overrightarrow{v_1v_2} \) is an XL edge since it is crossed by the Delaunay edge \( C_{v_1v_3E} \). However, the Delaunay edge between \( D \) and \( F \) passes through the XL edge \( \overrightarrow{v_1v_2} \). If we remove the Delaunay edge \( Dv_1v_2F \), then the surface is partitioned into Delaunay cells. Every cell is bounded by three Delaunay edges. For example, the red area in Fig. 13(b) is bounded by the Delaunay edges \( C_{v_1v_3v_4} \), \( C_{v_3v_2A} \) and \( A_{v_3v_4}F \). Generally, the approximation algorithm can be patched by (a) determining XL edges and (b) removing Delaunay edges that pass through XL edges. In practice, our experimental results illustrate that there are very few XL edges on refined and well-triangulated models.

7. A precision controlled approximant of the improved CH algorithm

Recall that the improved CH algorithm [9] exploits Lemma 2 to filter out useless interval windows. So we can easily import a parameter to get an over-filtering rule as follows. As Fig. 4(b) shows, if \( w = (d, f, e, AB) \) satisfies

\[
d + \|IA\| + \epsilon \geq d_1 + \|v_1B\|
\]

or

\[
d + \|IA\| + \epsilon \geq d_2 + \|v_2A\|
\]

we stop \( w \) from having children. We have the following theorem.

**Theorem 8.** When \( \epsilon \) exceeds the maximum edge length \( E \), the over-filtering rule will filter out all the interval windows, making the improved CH algorithm to be precisely Dijkstra’s algorithm [12].

**Proof.** As Fig. 15 shows, every interval window encodes a composite sequence of vertices and edges. Suppose the first interval window \( w \), rooted at the vertex \( f \), is on the edge \( \overrightarrow{AB} \). In fact, it suffices to prove that our over-filtering rule filters \( w \).

According to our assumption, the process of the improved CH algorithm is the same as Dijkstra’s algorithm before the window \( w \) is created. Using mathematical induction, we can conclude that for any edge \( \overrightarrow{v_i} \), the distance estimates at \( v_i, v_j \) satisfy \([d_{ij} - d_{ij}] \leq \|v_i\| \). We have \( d_4 + \|IA\| \geq d_3 \). Therefore \( d_4 + \|IA\| + E \geq d_4 + E \geq d_6 + \|AB\| \) and \( w \) is filtered, where \( E \) is the maximum edge length. \( \square \)

From Fig. 14 and Table 3, we can see that the precision parameter \( \lambda \) between 0 and 1 enables us to make a tradeoff between precision and performance. In practice, we suggest using \( \lambda E \) as \( \epsilon \) in the over-filtering rule, where \( 0 \leq \lambda \leq 1 \) and \( E \) is the average length of the edges.
Fig. 14. The improved CH algorithm, with a precision parameter $0 \leq \lambda \leq 1$, is precision controlled. When $\lambda = 0$, it is exactly the improved CH algorithm, and when $\lambda = 1$, it degenerates into Dijkstra’s algorithm.

Fig. 15. Proof to Theorem 8.

### Table 3

Testing the precision controlled improved CH algorithm: $\lambda \in [0, 1]$ is the precision parameter. “Windows” are used to take down the TOTAL number of interval windows (also including filtered windows) and “Levels” represent the maximum depth of the resulting sequence tree.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Improved CH</th>
<th>$\lambda = 5%$</th>
<th>$\lambda = 10%$</th>
<th>$\lambda = 100%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ≤ $\lambda$ ≤ 1</td>
<td>6.21</td>
<td>0.32</td>
<td>0.21</td>
<td>0.17</td>
</tr>
<tr>
<td>Time (s)</td>
<td>5536 127</td>
<td>237</td>
<td>183</td>
<td>173</td>
</tr>
<tr>
<td>Windows</td>
<td>127 496</td>
<td>10713</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Levels</td>
<td>361</td>
<td>237</td>
<td>183</td>
<td>173</td>
</tr>
</tbody>
</table>

8. Conclusions

In this paper, we have applied the improved CH algorithm to several shortest path problems. We extend the improved CH algorithm to the “single source, single destination” version and the “multiple sources, any destination” version. We also give a practical approach to estimate the geodesic distance to a surface point. Our main contribution in this paper is a set of approximation algorithms for computing geodesic based offsets, the geodesic based Voronoi diagram and the geodesic based Delaunay triangulation. Finally, we suggest importing a precision parameter $\lambda$, between 0 and 1, into the improved CH algorithm to make a tradeoff between precision and performance.

For sparse models with long and thin triangles, the Delaunay edges by the approximation algorithm are not always “straight”. One possible solution is to evolve the approximate Delaunay edges into exact shortest paths. However, the time cost will increase to $O(kn^2 \log n)$, where $k$ is the total number of sites and $n$ is the complexity of the input model. Unfortunately, the resulting Delaunay edges may intersect with each other.

### Acknowledgement

This work was supported by the NNSF of China (No. 60873111, No. 60933007).

### References


