Fast Exact and Approximate Geodesic Paths on Meshes

Danil Kirsanov
Steven J. Gortler
and
Hugues Hoppe

TR-10-04

Computer Science Group
Harvard University
Cambridge, Massachusetts

Fast Exact and Approximate Geodesic Paths on Meshes

Danil Kirsanov* Steven J. Gortler* Hugues Hoppe†

Abstract

In this paper, we develop simple and numerically stable family of algorithms for computing geodesic paths on meshes. The exact version of the algorithm is based on the interval propagation idea introduced by Mitchell, Mount, and Papadimitriou, and has the same $O(n^2 \log n)$ worst case time complexity. The fastest approximate version works in roughly $O(n \log n)$ time and still guarantees computing exact geodesics on any subdivision of a plane. The desired tradeoff between the time complexity and the error of the approximation can be achieved by setting the interval simplification threshold. The algorithms were evaluated on meshes with up to 100,000 vertices.

1 Introduction

In this paper we present practical methods for computing both exact and approximate geodesic paths on triangular meshes. These paths typically cut across faces in the mesh and so are not directly found by graph algorithms (such as Dijkstra's shortest path algorithm).

Many mesh-processing algorithms require a subroutine that efficiently computes geodesic paths between vertices on a triangular mesh. When parameterizing a complicated mesh, one often breaks it up into a set of charts; (See, for example, [KL96] for a manual chartification system, and [SWG*03] for an automatic one). In these cases, the parameterization can be done more efficiently (with less distortion and better packing efficiency) when the charts are bounded by geodesic paths. Geodesic paths also are needed when segmenting a mesh into subparts, such as done in [KT03]. In mesh editing systems, such as [KCVS98], one desires straight boundaries to define the bounding extents of editing operations.

Many mesh processing algorithms require geodesic distances between vertices in a triangular mesh. For example, radial-basis interpolation over a mesh requires point to point distances. This type of interpolation is used in numerical applications, such as skinning [SIC01], and in mesh watermarking [PHF99]. Shape analysis algorithms such as [HSKK01] use Morse analysis of a geodesic distance field. Mesh parameterization methods [ZKK02] based on isomap type algorithms [TdSL00] are also driven by point to point geodesic distances.

In this paper, we first give a simplified interpretation of, and describe a simple way to implement, the "MMP exact geodesic algorithm" [MMP87]. The MMP algorithm has worst case running time of $O(n^2 \log n)$ but runs much faster on most meshes. The original description of the MMP algorithm is quite complicated, and has never been implemented as far as we can tell. Our simplified interpretation makes an implementation very straightforward. In particular it involves

*Harvard University, {kirsanov, sjg}@deas.harvard.edu

†Microsoft Research, hhoppe@microsoft.com

no special handling of saddle vertices, and guarantees no gaps in the distance field due to numerical errors (a saddle vertex is a vertex for which the sum of the angles of the adjacent mesh faces is more than 2π).

The MMP algorithm splits up each edge on the mesh into a set of smaller intervals over which exact distance computation can be dealt with atomically. In the second part of this paper, we explain how approximate geodesics can be computed by running a version of MMP that skips these edge-splits. This approximate approach is (essentially) of comparable complexity to Dijskstra. Our approximate algorithm also has the property that it computes the exact geodesics if given a flat (planar) mesh.

1.1 Related Work

An exact geodesic algorithm with worst case time complexity of $O(n^2)$ was described by Chen and Han [CH96]. This algorithm was partially implemented by Kaneva and O'Rourke [KO00]. We show in our results section that our exact implementation runs many times faster than that method.

An exact geodesic algorithm with worst case time complexity of $O(n \log^2 n)$ was described by Kapoor [Kap99]. This is a very complicated algorithm which calls as "subroutines" many other complicated computational geometry algorithms. It is not clear if this algorithm will ever be implemented.

Approximate geodesics (with guaranteed error bounds) can be obtained by adding extra edges into the mesh and running Dijsktra on the one-skeleton of this augmented mesh [KS01], [LMS97]. These algorithm requires the addition of numerous extra edges to obtain accurate geodesics. The algorithms described in [KS01] relies on the selective refinement, and therefore significantly depends on the first approximation path found. If this approximation is far from the actual solution, the subdivision method might converge to the local minimal path (instead of the global geodesic one), or it might take a very large number of iterations until the refinement area moves to the vicinity of the actual geodesic and the process converge.

Approximate geodesics are computed by Kimmel and Sethian [KS98] using an algorithm that runs in $O(n \log n)$ time. The approximate geodesics found by this method can be quite inaccurate, even for planar meshes. We show in our results section that our approximate geodesic computation is much more accurate than that method.

Polthier and Schmies [PS98] describe a different definition of a geodesic path on meshes using a notion of "straightest" instead of "shortest". This notion may be inappropriate for some applications of geodesics.

2 Exact Algorithm

Our implementation of the exact algorithm is based on a very simple idea. Let us fix the source vertex on the mesh. The minimal distance from the source is a scalar function that is defined for every point on the surface of the mesh. In particular, for every edge e_i, that is parameterized by the parameter x, the *minimal distance function* $D_{e_i}(x)$ is well defined.

Let us consider two edges e_1 and e_2 that belong to the same face. If the shortest path from some point $x_2 \in e_2$ goes through the point $x_1 \in e_1$, then the distance function in the point x_2

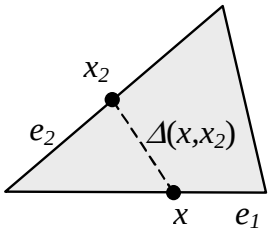

Figure 1: If the shortest path from the point x_2 of the edge e_2 crosses the edge e_1, the distance function in this point can be computed by theorem (2.1).

can be simply computed as $D_{e_2}(x_2) = D_{e_1}(x_1) + \Delta(x_1, x_2)$, where the $\Delta(x_1, x_2)$ is Euclidean distance between these points across the face. Because this path is the shortest possible one, we can generalize this observation.

Theorem 2.1. *If the shortest path from the point $x_2 \in e_2$ goes through the edge e_1 that belongs to the same face (fig. 1), then*

$$D_{e_2}(x_2) = \min_{x \in e_1} \left(D_{e_1}(x) + \Delta(x, x_2) \right) \tag{2.1}$$

Based on this theorem, we can write a simple algorithm (table 1) that is guaranteed to find all the distance functions on the edges correctly. This algorithm works for any convex or non-convex triangulated mesh with or without boundary. If the queue is sorted by the minimal value of the edge distance function, we have a Dijkstra-style propagation of the distance function across the surface.

The algorithm described above is a close relative of many algorithms developed for this problem before. On the one hand, it can be considered to be a continuous version of the discrete subdivision algorithm described in [LMS97]. On the other hand, this interval-based algorithm is also in the heart of [MMP87].

2.1 Back-tracing the shortest path

When all the distance functions on the edges are computed, tracing the shortest path is very simple. Any geodesic path is piece-wise linear curve with nodes on the edges of the initial mesh. Geodesics can be traced backward step-by-step by finding the next appropriate nodes. Let us take some initial node q that belongs to face f with edges e_1, e_2 and e_3. If q does not belong to any of the edges, the next node of the shortest path is point $\hat{x}$ that minimizes the distance function

$$\min_{x \in \{e_1, e_2, e_3\}} \left(D(x) + \Delta(x, q) \right) \tag{2.2}$$

Otherwise, if $q \in e_1$, the next node of the shortest path $\hat{x}$ minimizes

$$\min_{x \in \{e_2, e_3, e_4, e_5\}} \left(D(x) + \Delta(x, q) \right), \tag{2.3}$$

where e_4, e_5 are the edges of the other face adjacent to e_1. We can keep tracing the shortest path until we hit the origin.

algorithm FindMinimalDistanceFunctions

1. Compute distance functions for all edges that belong to the same faces as source. Put these edges in the queue. Define distance functions to be infinite for the rest of the edges.
2. **until** the queue is empty
3. Remove the next edge e from the queue.
4. **for** all edges e_i that belong to same faces as e.
5. Compute $\hat{D}_{e_i}(x)$ according to (2.1) for all $x \in e_i$
6. Update distance function at e_i as $D_{e_i}(x) = \min\left(D_{e_i}(x), \hat{D}_{e_i}(x)\right)$
7. If the function $D_{e_i}(x)$ is changed, put the edge e_i on the queue.
8. **end for**
9. **end until**

Table 1: Simple algorithm for computing minimal distance functions on the edges.

2.2 Intervals of optimality

To derive the general representation of the distance functions $D_{e_i}(x)$, we briefly review the interval theory developed in [MMP87].

First, let us consider convex polygons without boundary. The shortest path from some point on the edge to the source crosses some set of faces $F = \{f_i\}$. It can be shown that on a convex polygon with no boundary the shortest path never goes through the vertices of the polygon, except for the source vertex and, possibly, geodesic endpoint. We can *unfold* this sequence of faces to lay them flat on a plane. In this unfolding, the shortest path must be a straight line. In fact, the sets of points on an edge whose shortest pathes go through the same sets of faces, form the continuous *intervals of optimality* on the edge (fig. 2).

Using the cosine theorem, the distance function on such an interval p can be represented as

$$D_{p \subset e_i}(x) = \sqrt{x^2 - 2xc\cos\theta + c^2}, \tag{2.4}$$

where x is a distance from the origin of the edge, c is a distance from the source to the edge origin and θ is an angle is the angle between the edge and vector $\overrightarrow{c}$.

In the presence of a boundary, or when the polygon is non-convex, shortest passes can go through the vertices or follow edges of the polygon. In this case, other vertices can act as *pseudo-sources* for the intervals of optimality and one more parameter is needed to represent the distance function on the interval.

$$D_{p \subset e_i}(x) = d + \sqrt{x^2 - 2xc\cos\theta + c^2}, \tag{2.5}$$

where d is a distance from the source to the pseudo-source, c is now a distance from the pseudo-source to the origin of the edge. The distance function on the interval can therefore be uniquely

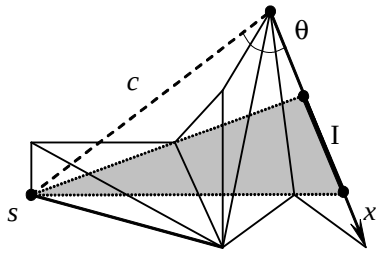

Figure 2: When the faces are unfolded on the plane, the shortest path from the the source (or pseudo-source) s to any point on the interval of optimality p is represented as a straight line. c is the distance from the (pseudo-)source to the edge origin, x is a parameter along the edge.

defined by three parameters (d, c, θ).

In general, the distance function on the edge is a continuous piece-wise smooth function that consists of the interval functions of the type (2.5).

2.3 Interval propagation

Given the distance function D_{e_1} on the edge e_1, the propagated distance function on the adjacent edge e_2 is defined by formula (2.1). It is possible to solve this problem algebraically, using the distance function representation (2.5), but it is much simpler to employ geometrical reasoning. We can propagate intervals independently and denote as $D_{e_2}|_{p \subset e_1}(x_2)$ the distance function that is created on the edge e_2 by the interval $p \subset e_1$,

$$D_{e_2}|_{p \subset e_1}(x_2) = \min_{x \in p \subset e_1} \left(D_{e_1}(x) + \Delta(x, x_2) \right). \tag{2.6}$$

The equation (2.1) can be restated as minimum over the interval propagated functions $D_{e_2,p}$,

$$D_{e_2}(x_2) = \min_{p \subset e_1} D_{e_2}|_{p \subset e_1}(x_2). \tag{2.7}$$

The interval propagation function $D_{e_2}|_{p \subset e_1}(x_2)$ can be computed from the geometrical reasons as follows. Part of the edge e_2 can "see" the pseudo-source s_p of the interval p directly through the interval p. Obviously, for the points in this region, the shortest path to the pseudo-source is a straight line. Therefore, this whole region can be represented as an interval with the pseudo-source s_p (fig. 3).

The shortest path of the points to the "left" side of this region have to pass through the "left" end of the interval p. That is why these points can be represented as an interval with new pseudo-source $s_{p,l}$. Similarly, the points on the "right" side of the edge can be represented as an interval with new pseudo-source $s_{p,r}$.

Summarizing, the interval propagation function $D_{e_2}|_{p \subset e_1}(x_2)$ consist of up to three intervals with different pseudo-sources. In theory, the shortest path cannot bend on the edge (it can bend only at the saddle or boundary vertex) and construction of the "left" and "right" intervals might seem unnecessary in many cases. However, by introducing these additional intervals, we achieve two goals simultaneously. First, we do not have to come up with a special treatment of the saddle

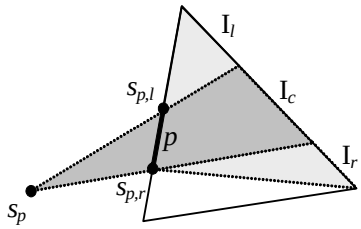

Figure 3: When interval p propagates onto the adjacent edge, it can create up to three new intervals. Central intervals I_c has its (pseudo-)source located in the same point s_p; left and right intervals I_l and I_r have their (pseudo-)sources located in the points s_l and s_r respectively.

and boundary vertices. Second, and more important, this trick provides numerical stability of the algorithm and covers small gaps in the "wavefront".

2.4 Interval intersection

In order to compute the minimum of two interval functions in formula (2.7) and algorithm (1), we need to find the points where two interval functions are equal to each other. Unlike the interval propagation, this operation is easier to do algebraically. Let us say that two different functions $D_1(x)$ and $D_2(x)$ are defined on the edge e. We want to find such points $\hat{x}$ that $D_1(\hat{x}) = D_2(\hat{x})$. When the pseudo-sources are the same distance from the initial source, $d_1 = d_2$, there is one intersection at most.

$$\hat{x} = \frac{c_2^2 - c_1^2}{2\left(c_1 \cos\theta_1 - c_2 \cos\theta_2\right)} \tag{2.8}$$

In general, when $d_1 \neq d_2$, we have to solve a quadratic equation

$$A\hat{x}^2 + B\hat{x} + C = 0, \tag{2.9}$$

where

$$\begin{aligned} A = P^2 - D^2, \;\; B = PQ - 2c_1 \cos\theta_1 D^2, \\ C = Q^2/4 - c_1^2 D^2, \;\; P = c_1 \cos\theta_1 - c_2 \cos\theta_2, \\ Q = D^2 + c_1^2 - c_2^2, \;\; D = d_1 - d_2. \end{aligned}$$

2.5 Interval-based algorithm

Putting it together, we can now derive final interval-based algorithm. In this algorithm, each edge distance function $D_{e_i}(x)$ is represented as a list of intervals L_{e_i}. Therefore, when we propagate an interval onto another edge, we create a new list of intervals that consist of up to three intervals (section 2.3). When we find the minimum of two distance function, we intersect two lists of intervals and take the smallest ones (section 2.4).

Obviously, all the edges that belong to the same faces as the source, can be seen from the source directly and therefore have only one interval each.

algorithm IntervalPropagation

1. Represent each edge that belongs to the same face as source vertex by one interval. Put these intervals in the priority queue.
2. **until** the queue is empty
3. Remove the next interval p from the queue. It belongs to an edge e.
4. **for** all edges e_i that belong to same faces as e.
5. Compute propagated interval list $\hat{L}_{e_i,p}$.
6. Update the interval list of the edge at e_i as $L_{e_i} = \min\left(L_{e_i}, \hat{L}_{e_i}\right)$.
7. Remove deleted intervals from the priority queue.
8. Add created intervals to the priority queue.
9. **end for**
10. **end until**

Table 2: Interval-based algorithm for computing minimal distance functions on the edges. When the interval list is updated, some intervals might be deleted and/or created.

The interval propagation algorithm can be seen in the table (2). We maintain the interval priority queue sorted by the minimum value of the distance function on the interval. This queue is described in [MMP87] in order to maintain a Dijkstra-style properties of the interval propagation. Computing the minimum value of the distance function on the interval is straightforward and omitted here for the sake of space.

In our implementation of the algorithm, each interval also has a bit that shows the direction of the interval propagation. Instead of propagating the interval into two directions, we propagate it only into one direction. In practice it makes the code faster, because the interval propagation procedure is computationally expensive.

2.6 Properties of the interval-based algorithm

Our interval-based algorithm is based on the algorithm developed in [MMP87], but has several crucial differences.

We do not have to worry about saddle vertices and boundary vertices, which otherwise require additional treatment. In the original algorithm, every saddle vertex became a pseudo-source of intervals. Similarly, it was reported by [KO00] about Chen&Han algorithm, that such vertices required significant additional processing time in their implementation of the shortest path algorithm.

Even more important property is robustness. When the intervals are propagating, some of them

become very small. The processing of the small intervals can require a lot of time and memory, and become a source of numerical instability. However, in MMP algorithm, it was not possible to clip small intervals, because it could result in large gaps when the intervals are propagated. By propagating side lobes of the intervals, we guarantee that all the gaps are covered in the very beginning.

This robustness property gives us an opportunity to construct a wide class of the approximate algorithms. We can simplify the intervals when they are small or when two neighboring intervals are similar. By construction of our algorithm it will never lead to gaps when the intervals are propagated.

Also, in contrast to [MMP87], we do not have to consider edge-face pairs and can treat each edge independently of where the signal comes from. This simplifies algorithm implementation.

3 Merge simplifications

Unfortunately, the exact algorithm is in the worst case quadratic in memory and $n^2 \log n$ in time [MMP87]. When the mesh has hundreds of thousands of vertices, this non-linear growth can become a bottleneck. The reason of such a growth is that far from the source intervals become smaller and smaller. In many cases, a set of small intervals can be approximated by a single interval with high precision.

In both MMP and Chen&Han algorithms, even the slightest modification of any interval could lead to a potential distance function gap somewhere on the mesh. On the contrary, our algorithm remains stable after interval approximation, which is demonstrated in the following lemma.

Lemma 3.1. *Let us take the results of the exact algorithm, pick one of the edges, e_k, and one interval p_m on this edge. Let us say that the distance function on this interval, $D_{p_m \subset e_k}(x)$ is defined by the constants d, c, θ. Let us also define a new distance function $\bar{D}_{p_m \subset e_k}(x)$ by changing these parameters arbitrarily and re-propagate the interval through the mesh. Then the resulting distance field in any point of the mesh will not change more than ε, where*

$$\varepsilon = \max_x |\bar{D}_{p_m \subset e_k}(x) - D_{p_m \subset e_k}(x)|.$$

To make a bridge to our flat-exact algorithm developed in the next chapter, here we describe a possible simplification algorithm that is based on interval merging. Because of the perfect results shown by the flat-exact algorithm, we did not implemented the merging algorithm described here.

The easiest merging strategy is to check the neighbors of the interval before its propagation, and if their distance function are approximated by its distance function well enough, merge them together.

There are still two issues remain. First, after a chain of simplifications the algorithm can come to the loop, i.e. some interval become modified by its own ”children” intervals. Second, simplifications can create local minimums in the distance field of the mesh and the tracing-back algorithm that computes geodesics can stuck in such a minimum.

It is possible that both issues can be handled by applying some restrictions on the simplification rules. We did not find the set of restrictions that is elegant and fast to compute; that is why we resolve both issues by the following method.

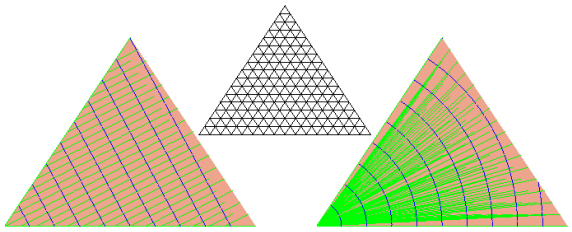

Figure 4: Comparison of the approximate geodesics algorithms for the planar mesh, which is subdivision of a triangle (top). The source is located in the leftmost corner of the mesh. Kimmel-Sethian algorithm is on the left, our flat-exact one is on the right. Geodesics are shown in green, iso-distant curves are shown in blue.

3.1 Dependence DAG

In our algorithm, every interval is a result of the propagation of another interval (for mathematical accuracy, we have to mention that the intervals adjacent to the source vertex are the results of the propagation of this vertex). We can consider a dependence graph. A vertex of this graph corresponds to an interval of the mesh, and the edges of the graph are directed from the parent intervals to their children. At every stage of our exact algorithm, this directed graph is acyclic (DAG). The idea is to maintain such a dependence DAG for approximation algorithm.

In the exact algorithm, an interval has only one parent by construction. The only difference of the approximation algorithm is that one interval can have many parents. Whenever the intervals are merged, the resulting interval inherit all their children and parents. Similarly, whenever we create new interval that covers the area of the edge that belonged to some other interval, it has only one parent but inherits all children of this previous interval.

We merge or propagate intervals only if the dependence graph remains acyclic. This rule solves the cycling problem automatically. By back-tracing the geodesics only from children to parents, we make sure that the tracing algorithm always converge to the origin.

Checking whether the resulting graph remains acyclic in theory requires breadth-searching the whole graph, and therefore ruins the complexity of our algorithm. However, in practice this check is extremely fast, because most of the time we do not have to consider the entire graph. The updated intervals are located on the frontier of the distance propagation wave, i.e. they are located very close to the leaves of the DAG. Therefore, if the graph remains acyclic, the search will stop after processing the few descendants of the updated interval.

4 Flat-exact algorithm

Though interval simplification improves both time and memory complexity of the algorithm, it is even more complicated to implement this algorithm from scratch than the original exact algorithm. That is why we focused on the approximation algorithm that is also very simple to implement. Though it can be considered as an extreme case of the interval simplification described in the previous chapter, this algorithm can also be related to algorithm described in [KS98].

The following section describes an approximation algorithm where each edge is represented

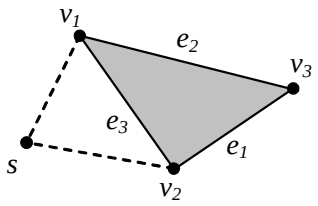

Figure 5: Edge e_3 with source s is used to compute the propagated distance function in the vertex v_3.

by a single interval. Now terms "edge" and "interval" are equivalent and we will call interval parameters to be edge parameters.

In the extreme case when the entire mesh is located in one plane, the resulting distance function will coincide with the exact one. Because of this reason, we call the simplification to be flat-exact. This property is extremely important in practice. For example, let us consider the planar mesh that was created as a regular subdivision of a triangle 4, and put the source in the leftmost corner of the mesh. For this simple mesh, the distance field computed by Kimmel-Sethian algorithm will be a linear function with the gradient equal to 1 everywhere. The resulting geodesics will be a set of parallel lines, that can be very far from the actual geodesics.

The main structure of the interval-based algorithm remains unchanged. Initially, one interval is created for all edges that belong to the same faces as the source. Then the intervals are propagated by a very simple rules across the mesh.

4.1 Simplified interval structure and propagation rules

Another simplification we make is that the distance field on the whole mesh can be represented by the distances values in the vertices of the mesh. In particular, the distance function on the edge is defined by its values in the vertices of the edge, $D(v_1)$ and $D(v_2)$. The distance function on the edge is still of the form (2.4), that has three parameters (d, c, θ). In order to compute them from the two vertex distance values, from now on we consider $d = 0$, i.e. we ignore the pseudo-sources. After this assumption, the one-to-one correspondence between the (c, θ) and $(D(v_1), D(v_2))$ is trivial

$$D(v_1) = c, \tag{4.1}$$

$$D(v_2)^2 = c^2 + L^2 - 2cL\cos\theta, \tag{4.2}$$

where L is the length of the edge. Our new construction also implies that the distance function is continuous on the 1-skeleton of the mesh.

The propagation rules change as follows. Given the edge e_3 to be propagated, we first define the distance in the opposite vertex of the triangle (fig. 5). If the quadrilateral (s, v_1, v_3, v_2) is convex, i.e. the opposite vertex can "see" the source s of the interval, the propagated distance $\hat{D}(v_3)$ is computed as Euclidean distance between two points, $\hat{D}(v_3) = d(s_e, v_3)$. Otherwise, the distance is computed along the edge, $\hat{D}(v_3) = \min(D(v_1) + |e_2|, D(v_1) + |e_2|)$.

If new distance in the vertex v_3 is smaller than the old one, we update all the edges adjacent to v_3 and put them in a priority queue. The simplified algorithm is represented in the table 3.

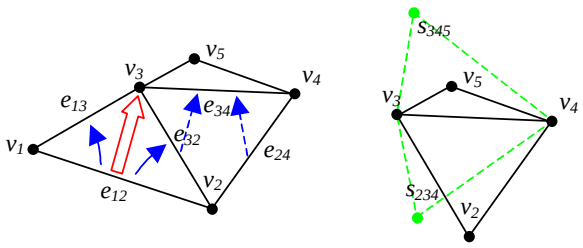

Figure 6: Construction of the dependance DAG. The vertex v_3 is updated from the edge e_{12} (red arrow), the blue arrows show the desirable parent-children links. On the right, two possible pseudo-sources for the edge e_{34} are located symmetrically to this edge.

4.2 Dependance DAG for the simplified algorithm

In order to avoid cycling, we use the same idea of the dependance DAG as before. Each edges of the is represented by a vertex in the DAG. The dependence information is now maintained as follows. Let us say that the vertex v_3 has its distance updated from the edge e_{12} that belong to the face f_{123} (fig. 6). The distance function over all the edges adjacent to v_3 have now been affected by e_{12}. So to avoid cycling in the distance propagation, we add a pointer from e_{12} to all of the edges adjacent to v_3.

During shortest path back-tracing, we will only trace paths that go towards ancestors edges, as recorded in the DAG. If an edge has no ancestor edges that it shares a face with, then we only trace the path back to one of its vertices. So, when v_3 is updated to allow as much edge to edge back-tracing when possible we would also like to other DAG edges.

We do this by "estimating" the direction to the source and adding extra dependencies into the DAG as follows. Consider edge e_{34}, and assume that the weights of both v_2 and v_5 are finite. To assess direction of the source we compute the parameters (c_{35}, θ_{35}) and find the positions of two possible pseudo-sources, s_{234} and s_{345}. Then we also assess how well these pseudo-sources approximate the distance function values in the vertices v_2 and v_5.

$$Q_{234} = |D(v_4) - \|s_{234} - v_4\|| + |D(v_3) - \|s_{234} - v_3\||\,,$$
$$Q_{345} = |D(v_4) - \|s_{345} - v_4\|| + |D(v_3) - \|s_{345} - v_3\||\,.$$

If $Q_{234} < Q_{345}$, we set e_{32} and e_{24} as parents of e_{34} otherwise we set e_{35} and e_{54} to be parents of e_{34}.

If both of the vertices v_2 and v_5 have infinite weights, then no extra DAG information is added, and back-tracing will have to go through edges in the graph.

Again, the initial vertex v_3 is updated only in case that after this update the resulting dependency graph remains acyclic.

Finally, we want to mention that the situations when the dependency graph may become cyclic are very rare and have only been seen in hand-crafted examples.

algorithm SimplifiedIntervalPropagation

1. Compute the distance for all vertices adjacent to the source. Put all edges that belong to the same faces as source in the priority queue.
2. **until** the queue is empty
3. Remove the next edge e from the queue.
4. **for** both faces f_i adjacent to e.
5. Consider the vertex v opposite to e.
6. Compute updated distance function $\hat{D}(v)$.
7. Update the dependence graph.
8. **if** $\hat{D}(v) < D(v)$ and the dependence graph is acyclic
9. $D(v) = \hat{D}(v)$
10. Put all edges adjacent to v in a queue.
11. **end if**
12. **end for**
13. **end until**

Table 3: Simplified algorithm for computing minimal distance function.

5 Results

In order to test the algorithms, we compared them with several existing ones. The fastest (and least precise) algorithm for approximate geodesic path computation is Dijkstra shortest path that runs on the edges of the mesh. We also implemented an acute version of the approximation algorithm proposed by Kimmel and Sethian [KS98] and used publicly available implementation of Chen and Han's algorithm by Kaneva and O'Rourke [KO00](due to unknown implementation reasons, we were unable to test this algorithm to all the meshes we have). The algorithms were tested on an Athlon 1.5GHz machine with 2Gb of memory.

To assess the relative error, we used the following test. Given the mesh and a randomly selected source vertex, we computed the shortest paths to all vertices of the mesh. The relative error was computed as a sum over all vertices

$$E = \frac{1}{N_p} \sum_i \left| \frac{|\bar{p}_i| - |p_i|}{|p_i|} \right|, \tag{5.1}$$

where $|p_i|$ is the length of the shortest path obtained by the exact algorithm, $|\bar{p}_i|$ is the length

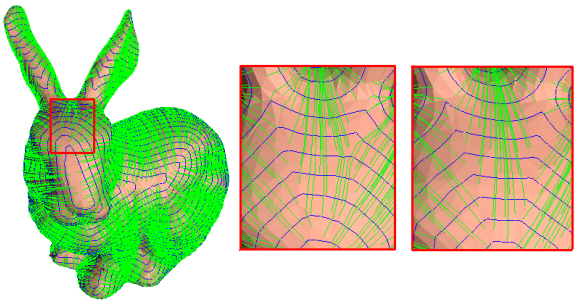

Figure 7: Limitations of our flat-exact algorithms. In the areas where several wavefronts meet together, the one-interval approximation of the edge distance function leads to geodesic distortion. Enlarged area of the bunny head is shown in the middle (flat-exact). Result of the exact algorithm is shown to the right. Geodesics are shown in green, iso-distance curves are shown in blue.

of the shortest path obtained by an approximation algorithm, and N_p is the total number of the computed paths.

The results of both exact and flat-exact algorithms are shown in figures 8, 9.

One of the interesting effects that we observed with our exact algorithms, is that it is usually works much faster for the models with a lot saddle vertices and gives the slowest performance on the saddle objects (compare *sphere2* vs. *parasaur* models). This happens because on the convex models the intervals do not eliminate each and become smaller and smaller while the ”wavefront” propagates further from the source.

The flat-exact algorithm runs a little bit slower than Kimmel-Sethian algorithm because of the necessity to maintain the DAG structure, but gives significantly better results.

The only limitation of the flat-exact algorithm that we could find can be observed when the two distance ”wavefronts” meet on the opposite sides of some obstacle. In this case our one-interval simplification rule oversimplifies the distance field and the geodesics are visibly distorted (fig. 7).

References

[CH96] CHEN J., HAN Y.: Shortest paths on a polyhedron; part i: computing shortest paths. *Int. J. Comput. Geom. & Appl. 6*, 2 (1996), 127–144.

[HSKK01] HILAGA M., SHINAGAWA Y., KOHMURA T., KUNII T. L.: Topology matching for fully automatic similarity estimation of 3d shapes. In *Proceedings of ACM SIGGRAPH 2001* (Aug. 2001), Computer Graphics Proceedings, Annual Conference Series, pp. 203–212.

[Kap99] KAPOOR S.: Efficient computation of geodesic shortest paths. In *Proc. 32nd Annu. ACM Sympos. Theory Comput.* (1999), pp. 770–779.

[KCVS98] KOBBELT L., CAMPAGNA S., VORSATZ J., SEIDEL H.-P.: Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of SIGGRAPH 98* (July 1998), Computer Graphics Proceedings, Annual Conference Series, pp. 105–114.

[KL96] KRISHNAMURTHY V., LEVOY M.: Fitting smooth surfaces to dense polygon meshes. In *Proceedings of SIGGRAPH 96* (Aug. 1996), Computer Graphics Proceedings, Annual Conference Series, pp. 313–324.

[KO00] KANEVA B., O'ROURKE J.: An implementation of chen & han's shortest paths algorithm. In *Proc. of the 12th Canadian Conf. on Comput. Geom.* (2000), pp. 139–146.

[KS98] KIMMEL R., SETHIAN J. A.: Computing geodesic paths on manifolds. *Proc. National. Academy of Sciences 95*, 15 (1998), 8431–8435.

[KS01] KANAI T., SUZUKI H.: Approximate shortest path on a polyhedral surface and its applications. *Computer-Aided Design 33*, 11 (2001), 801–811.

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics 22,* 3 (July 2003), 954–961.

[LMS97] LANTHIER M., MAHESHWARI A., SACK J.-R.: Approximating weighted shortest paths on polyhedral surfaces. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.* (1997), pp. 274–283.

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

[PHF99] PRAUN E., HOPPE H., FINKELSTEIN A.: Robust mesh watermarking. In *Proceedings of SIGGRAPH 99* (Aug. 1999), Computer Graphics Proceedings, Annual Conference Series, pp. 49–56.

[PS98] POLTHIER K., SCHMIES M.: Straightest geodesics on polyhedral surfaces. *Mathematical Visualization, Ed: H.C. Hege, K. Polthier, Springer Verlag* (1998), 391.

[SIC01] SLOAN P.-P. J., III C. F. R., COHEN M. F.: Shape by example. In *2001 ACM Symposium on Interactive 3D Graphics* (Mar. 2001), pp. 135–144.

[SWG*03] SANDER P., WOOD Z., GORTLER S., SNYDER J., HOPPE H.: Multi-chart geometry images. *ACM Symposium on Geometry Processing* (2003).

[TdSL00] TENENBAUM J. B., DE SILVA V., LANGFORD J. C.: A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000), 2319–2323.

[ZKK02] ZIGELMAN G., KIMMEL R., KIRYATI N.: Texture mapping using surface flattening via multidimensional scaling. *IEEE Transactions on Visualization and Computer Graphics 8*, 2 (apr - jun 2002), 198–207.

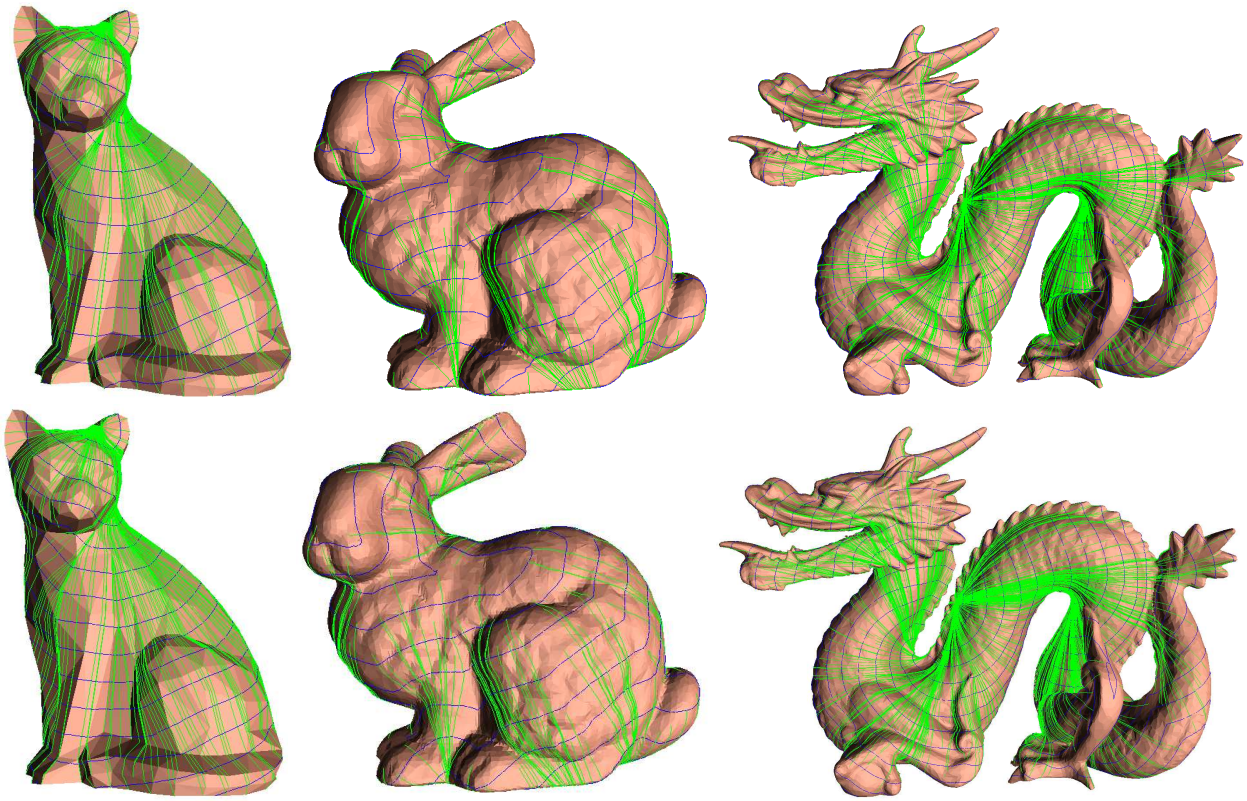

Figure 8: The results of the our exact (top row) and flat-exact (bottom row) algorithms are shown on the cat, bunny, and dragon models (366, 10000, and 50000 vertices respectively). Geodesics are shown in green, iso-distant curves are shown in blue. In the bunny model, the source is on the back side.

model	number of vertices	Dijkstra	Kimmel-Sethian	Chen-Han	Our Exact	Our Flat-exact
cat	366	0/6.1	0.011/2.2	5.0/0	0.12/0	0.015/0.2
plane triangulation	561	0/7.3	0.015/2.1	5.2/0	0.031/0	0.016/0
sphere1	1562	0.01/7.2	0.047/0.9	11.0/0	2.56/0	0.063/0.08
sphere2	6242	0.015/7.2	0.203/0.7	-	40.1/0	0.21/0.06
handmale	7609	0.023/5.7	0.26/1.7	-	11.5/0	0.34/0.22
bunny	10002	0.025/5.3	0.37/2.0	44160.0/0	12.5/0	0.41/0.25
buddhaf	14990	0.06/4.8	0.62/2.4	101500.0/0	10.9/0	0.85/0.36
parasaur	21935	0.082/6.3	0.78/1.9	-	45.5/0	0.86/0.32
rockerarm	40177	0.16/5.7	1.6/1.4	-	235.0/0	1.7/0.12
horse	48500	0.21/4.9	1.9/1.8	-	343.0/0	2.2/0.13
dragon	50000	0.23/6.4	2.3/2.3	-	64.0/0	2.7/0.42

Figure 9: Time/error statistics for the geodesics algorithms. Time is given in seconds, the relative error is computed by 5.1 and shown in percents.