Fast exact and approximate geodesics on meshes

Fast Exact and Approximate Geodesics on Meshes

Vitaly Surazhsky, University of Oslo
Tatiana Surazhsky, University of Oslo
Danil Kirsanov, Harvard University
Steven J. Gortler, Harvard University
Hugues Hoppe, Microsoft Research

Abstract

The computation of geodesic paths and distances on triangle meshes is a common operation in many computer graphics applications. We present several practical algorithms for computing such geodesics from a source point to one or all other points efficiently. First, we describe an implementation of the exact "single source, all destination" algorithm presented by Mitchell, Mount, and Papadimitriou (MMP). We show that the algorithm runs much faster in practice than suggested by worst case analysis. Next, we extend the algorithm with a merging operation to obtain computationally efficient and accurate approximations with bounded error. Finally, to compute the shortest path between two given points, we use a lower-bound property of our approximate geodesic algorithm to efficiently prune the frontier of the MMP algorithm, thereby obtaining an exact solution even more quickly.

Keywords: shortest path, geodesic distance.

1 Introduction

In this paper we present practical methods for computing both exact and approximate shortest (i.e. geodesic) paths on a triangle mesh. These geodesic paths typically cut across faces in the mesh and are therefore not found by the traditional graph-based Dijkstra algorithm for shortest paths.

The computation of geodesic paths is a common operation in many computer graphics applications. For example, parameterizing a mesh often involves cutting the mesh into one or more charts (e.g. [Krishnamurthy and Levoy 1996; Sander et al. 2003]), and the result generally has less distortion and better packing efficiency if the cuts are geodesic. Geodesic paths are used in segmenting a mesh into subparts, as done in [Katz and Tal 2003; Funkhouser et al. 2004]. Mesh editing systems such as [Kobbelt et al. 1998] also use geodesics to delineate the extents of editing operations. Simulating fire on a mesh [Lee et al. 2001] also benefits from geodesics.

In addition, geodesic paths establish a surface distance metric, which is an essential building block for many other techniques. For example, radial-basis interpolation over a mesh requires calculation of geodesic distances, and is used in numerous applications such as skinning [Sloan et al. 2001], mesh watermarking [Praun et al. 1999], and the definition of surface vector fields [Praun et al. 2000]. Shape classification algorithms such as [Hilaga et al. 2001] use Morse analysis of a geodesic distance field. Parameterization metrics based on isomaps [Zigelman et al. 2002; Zhou et al. 2004; Peyré and Cohen 2005] are also driven by geodesic distances.

In this paper we explore the problem of producing both exact and approximate solutions for geodesic paths (and hence distances) on triangle meshes (Figure 1). We present three contributions:

Exact algorithm We first present an efficient implementation of the exact geodesic algorithm by Mitchell, Mount, and Papadimitriou (MMP) [1987]. Using a simple parameterization of the distance function over the edges, the implementation is actually practical even though, to our knowledge, it has never been done previously. We demonstrate that the algorithm's worst case running time of $O(n^2 \log n)$ is pessimistic, and that in practice, the algorithm runs in sub-quadratic time. For instance, we can compute the exact geodesic distance from a source point to all vertices of a 400K-triangle mesh in about one minute.

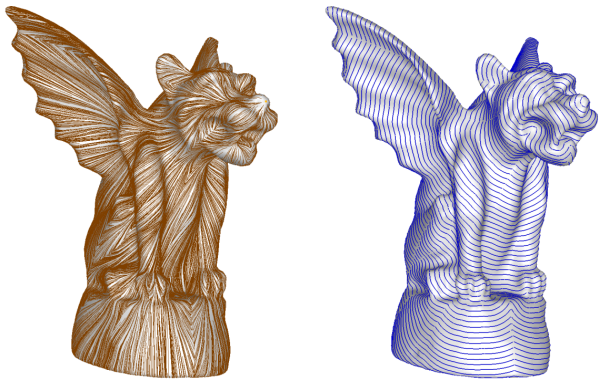


Figure 1: Geodesic paths from a source vertex, and isolines of the geodesic distance function.

Approximation algorithm We extend the algorithm with a merging operation to obtain computationally efficient and accurate approximations with *bounded* error. In practice, the algorithm runs in $O(n \log n)$ time even for small error thresholds.

Exact geodesic path between two points We show how to efficiently obtain the exact solution to the "single source, single destination" problem, by using a lower-bound property of our approximation algorithm to prune the frontier of the MMP algorithm. In practice, we compute the shortest path between two points on a 1M-triangle mesh in just a few seconds.

2 Related work

The MMP algorithm [Mitchell et al. 1987] provides an exact solution for the "single source, all destination" shortest path problem on a triangle mesh. Their algorithm partitions each mesh edge into a set of intervals (*windows*) over which the exact distance computation can be performed atomically. These windows are propagated in a "continuous Dijkstra"-like manner. They prove a worst case running time of $O(n^2 \log n)$. Unfortunately, as far as we know the MMP algorithm has not been implemented previously and thus has not made its way into practice.

An exact geodesic algorithm with worst case time complexity of $O(n^2)$ was described by Chen and Han [1996] and partially implemented by Kaneva and O'Rourke [2000]. We show that our MMP implementation runs many times faster than that implementation.

Kapoor [1999] describes an algorithm for the "single source, single destination" geodesic path between two given mesh vertices, in $O(n \log^2 n)$ time. This is a complicated method which calls as subroutines many other complicated computational geometry algorithms; it is unclear if this algorithm will ever be realized.

Approximate geodesics with guaranteed error bounds can be obtained by adding extra edges into the mesh and running Dijkstra on the one-skeleton of this augmented mesh [Lanthier et al. 1997].

Many extra edges are required to obtain accurate geodesics. Algorithms such as [Kanai and Suzuki 2001; Martinez et al. 2004] rely on iterative optimization, and the results therefore depend significantly on the initial approximate path. If this approximation is poor, the method might converge to an (incorrect) locally shortest path, or might require a large number of iterations. Mitchell [2000] presents a broad survey of approximate algorithms for graph and geodesic search.

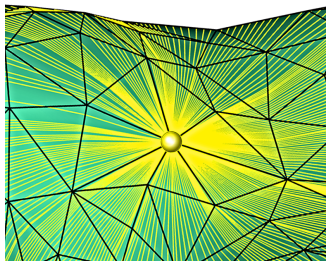
Kimmel and Sethian [1998] employ a variant of the fast-marching method to compute approximate geodesics on meshes in $O(n \log n)$ time. Several methods [Novotni and Klein 2002; Kirsanov 2004; Reimers 2004] explore an improved update rule for geodesic computation. Many of these methods require special processing of triangles with obtuse angles. We show that our MMP-based approximation algorithm yields more accurate solutions than the fast-marching method when applied to meshes. Moreover in Appendix B, we also demonstrate benefits of our algorithm when applied to meshes that approximate smooth manifolds.

Polthier and Schmies [1998] explore a different definition of geodesic path on meshes using a notion of "straightest" instead of "shortest". Because these straightest geodesics are not always defined between pairs of points on a mesh, this notion may be inappropriate for many applications. Pham-Trong et al. [2001] explore geodesic paths over smooth parametric surfaces.

3 Exact algorithm

> *A window in the wall admits light into the room and its borders define the illuminated regions on the other walls of the room.*
>
> *Common knowledge*

Given a piecewise planar surface $\mathcal{S}$ defined by a triangle mesh, and a source vertex $v_s \in \mathcal{S}$, the MMP algorithm computes an explicit representation of the *geodesic distance* function $D : \mathcal{S} \to \mathcal{R}$. For any point $p \in \mathcal{S}$, this function $D(p)$ returns the length of the geodesic path from p back to the source v_s. Once a complete representation for D has been computed, one can quickly apply a "backtracing" algorithm to compute the shortest path from any query point to the source. This distance function can also be used to calculate *isolines* of constant distance (Figure 1).



Shortest paths can be visualized as rays emanating from the source vertex v_s in all tangent directions. These shortest paths are governed by the following three properties. Interior to a triangle, a shortest path must be a straight line. When crossing over an edge, a shortest path must correspond to a straight line if the two adjacent faces are *unfolded* into a common plane. As proven in [Mitchell et al. 1987], the only vertices that a shortest path can pass through (besides the source and destination) are (1) boundary vertices, (2) *saddle vertices* which are vertices with total angle greater than 2π (i.e. also called hyperbolic vertices), and (3) *parabolic vertices* whose total angle equals 2π. Parabolic vertices typically do not require special treatment since their neighborhoods unfold isometrically.

Algorithm overview The basic idea of the MMP algorithm is to track together groups of shortest paths that can be parameterized atomically. This is achieved by partitioning each mesh edge into a set of intervals that we call *windows*. We will show that all the shortest paths within a window can be encoded locally using a 6-tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$. The windows are then propagated across faces of the mesh in Dijkstra-like sweep.

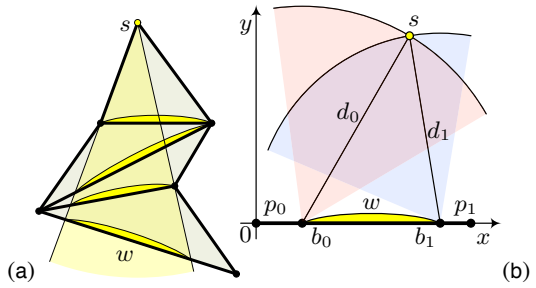
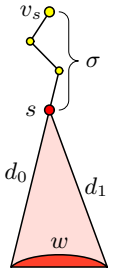


Figure 2: (a) Pencil of rays from the source vertex s passing through window w over the strip of unfolded triangles. (b) The position of the source v_s (or more generally pseudosource s) is parameterized relative to the window w that lies on edge $\overrightarrow{p_0 p_1}$.

3.1 Distance field along a window

Consider a specific shortest path from the source vertex v_s to some point p on an edge e, and let us assume that this path does not pass through any saddle or boundary vertices. In this case, when all the faces intersecting the path are unfolded in a single common plane, the path forms a straight line. Consider the set of neighboring points on e whose shortest paths back to the source pass through the same sequence of faces. These paths are also straight lines in the same unfolding. In particular, the paths form a *pencil* of lines emanating from the unfolded source vertex. We represent this group of shortest paths atomically over a window w of the edge e (see Figure 2(a)).

The distance field $D(p)$ over the window w is represented compactly as follows. We first store the endpoints of the window using two scalar valued parametric coordinates $b_0, b_1 \in [0, \|e\|]$ measuring distance along the edge. Next, we encode the position of the source vertex (relative to the window in the planar unfolding) using its distances d_0, d_1 to the window endpoints. Finally, we record a binary direction τ specifying the side of the edge on which the source lies. From this tuple $(b_0, b_1, d_0, d_1, \tau)$, it is easy to position the source in the planar unfolding adjacent to the edge, by intersecting two circles as shown in Figure 2(b), and thereby recover the distance field within this window.



Suppose now that the shortest path passes through one or more saddle/boundary vertices on its way to the source, and let s be the nearest such vertex to w. Again, consider the set of neighboring points on e whose shortest paths go through the same strip of faces back to s. In the unfolding of the strip between e and s, these shortest paths will form a pencil of lines emanating from s as seen in Figure 3. Because this set of shortest paths share the same path from s back to the source vertex v_s, the distance field is now characterized by (1) the position of this *pseudosource* vertex s relative to the edge, and (2) the length $\sigma = D(s)$ of the path from s back to the source v_s. We refer to σ as the *pseudosource distance*.

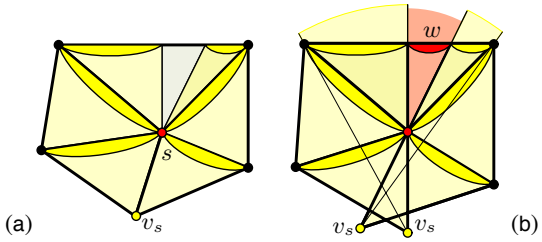


Figure 3: (a) Orthogonal projection of a 3D mesh near a red saddle vertex. Part of the edge in the upper triangle is not visible by rays from the source vertex v_s. (b) Unfolding the triangles into the plane of the upper triangle reveals that the total angle is greater than 2π, resulting in two different "images" of v_s in the unfolding. All shortest paths from v_s to the red window w pass through the red pseudosource vertex s.

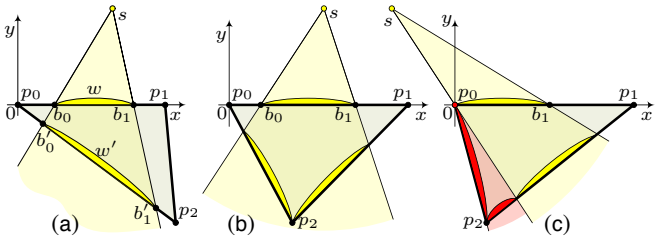


Figure 4: (a) Window propagation resulting in one window. (b) Window propagation resulting in two windows. (c) Special case of window propagation (saddle vertex at p_0); two additional (red) windows are added to the left of the ray (s, p_0).

To summarize, the distance field $D(p)$ over the window is expressed as a tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$, where b_0, b_1 define the endpoints of w, d_0, d_1 are the corresponding distances to the pseudosource, σ is the geodesic distance from s to the source v_s, and τ encodes the direction of s from the directed edge e.

3.2 Window propagation

Given a window w on an edge e_1, we propagate its distance field across an adjacent face f to define new potential windows on the two opposing edges e_2, e_3. Essentially, we compute how the pencil of straight lines would extend across one more unfolded face in the strip. However, the edges e_2 and e_3 may already contain previously propagated windows, so we must "intersect" these previous windows with the new potential windows, to capture their combined minimum distance field. In other words, we only keep the part of a newly computed potential window that has a smaller geodesic distance than that already associated with points on e_2 or e_3.

Let w' be the new potential window on one of the two opposing edges (Figure 4(a,b)). To define the distance field over w', we extend the rays from the pseudosource s through the endpoints of w and intersect them with the new edge, to obtain the new interval $[b_0', b_1']$. We then compute the new distances d_0', d_1' from these new endpoints to the pseudosource s. The pseudosource distance $\sigma' = \sigma$ is unchanged, and the direction τ is assigned to point into face f.

We have a special case when w is adjacent to a saddle/boundary vertex v, since shortest paths may pass *through* v, i.e. v may act as a *new* pseudosource. Suppose v lies at the left endpoint $q_0 = p_0$ of w. In this case we add extra windows on the edges $\overrightarrow{p_1 p_2}$ and $\overrightarrow{p_2 p_0}$. These additional windows cover the parts of the edges that lie to the left of the ray (s, p_0) and are not already "illuminated" by s through w (Figure 4(c)). These new additional windows will have a pseudosource at v with $\sigma = D(v)$. The case with v at $q_1 = p_1$ is treated in a symmetrical manner.

3.3 Intersection of overlapping windows

Suppose that the newly created window w_0 and at least one existing window w_1 on edge e have a nonempty intersection region $\delta = w_0 \cap w_1$. We must decide which of the windows defines the *minimal* distance function for each point in δ, and update the windows along e appropriately.

In particular, if one of the windows defines a larger distance everywhere over δ, then we simply cut δ away from its interval. The more interesting case is when w_0 is minimal on part of δ, while w_1 is minimal on the remaining part of δ. To correctly partition δ, we must then find the point $p \in \delta$ where the distance functions defined by w_0 and w_1 are equal, i.e. $\|s_0 - p\| + \sigma_0 = \|s_1 - p\| + \sigma_1$. If we define our planar coordinate system to align e with the x axis as shown in Figure 5(a), this can be expressed as:

$$\sqrt{(p_x - s_{0x})^2 + s_{0y}^2} + \sigma_0 = \sqrt{(p_x - s_{1x})^2 + s_{1y}^2} + \sigma_1.$$

This equation can be reduced to a quadratic with a single solution

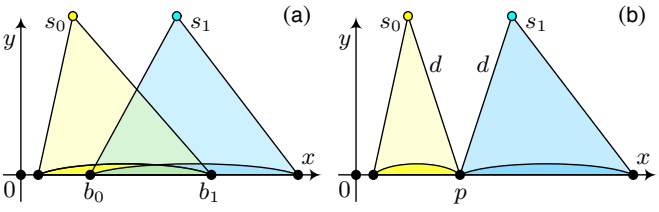


Figure 5: (a) Two overlapping windows w_0 and w_1 (with unfolded pseudosources s_0 and s_1) and their intersection $\delta = [b_0, b_1]$. (b) The formation of two disjoint windows; in the special case that $\sigma_0 = \sigma_1$, then $\|s_0 - p\| = \|s_1 - p\| = d$ as illustrated.

in the required range, i.e. $p_x \in \delta$ such that

$$Ap_x^2 + Bp_x + C = 0, \quad \text{where} \quad A = (\alpha^2 - \beta^2),$$
$$B = \gamma\alpha + 2s_{1x}\beta^2, \quad C = 1/4\gamma^2 - \|s_1\|^2\beta^2,$$
$$\text{with } \alpha = s_{1x} - s_{0x}, \ \beta = \sigma_1 - \sigma_0, \ \gamma = \|s_0\|^2 - \|s_1\|^2 - \beta^2.$$

Finally, we adjust the boundary b_1 of the left window and b_0 of the right window to the location of p_x as shown in Figure 5(b).

3.4 Continuous Dijkstra

The MMP algorithm propagates distance information out from the source in a Dijkstra-like fashion. When windows are created, they are placed in a priority queue sorted by distance back to the source. When a window is popped off the queue, it is propagated outward across a face. We next present this algorithm in more detail.

The priority queue is initialized with a window for each edge adjacent to the source v_s. The distance fields for these initial edges are trivial, e.g. vertices adjacent to v_s are assigned distances D equal to the corresponding edge lengths.

The basic step is to select (and remove) a window from the queue and propagate it as in Section 3.2. Note that the propagation step can add, modify, or remove existing windows, and the queue is updated accordingly. We repeat the process until the queue is empty.

Interestingly, the algorithm will generate the correct solution regardless of the order in which windows are removed from the queue. However, selecting windows in arbitrary order leads to an extremely slow result. The optimal approach is to propagate the windows as a wavefront, by ordering them in the queue according to minimal distance from the source vertex, i.e. $\min_{p \in w} D(p)$ for window w. In practice, other reasonable criteria are acceptable. For example, the minimal distance at the window endpoints is faster to compute and the overall algorithm performance is nearly the same.

Because of the limited precision of the computations, small gaps or overlaps can be generated between windows on an edge. We check and fix such problems by either extending or reducing the window extents. Even for our largest models (700K faces) we have not encountered numerical difficulties. Note that the total path lengths are simple sums of distances across finitely many triangle faces ($O(\sqrt{n})$ in practice), so numerical errors do not accumulate exponentially as they might with ODE integration algorithms.

3.5 Geodesic path construction: Backtracing

Once all edges are covered by windows representing geodesic distance, it is easy to trace a shortest path from any surface point p back to the source.

First, in the general case that p lies in a face interior, we consider all windows on the three edges bounding the face, and minimize $\|p - p'\| + D(p')$ over all points p' within these windows.

Having jumped to a first window, we can then iteratively hop to previous windows all the way back to the source. That is, given location p on a current window, we find the adjacent face f according to the direction τ. We reconstruct the location of the pseudosource

s in the plane of the face using the window parameters, and intersect the line $\overrightarrow{p\,s}$ with the two opposite edges of face f. This intersection point gives us a new point on a new window, from which we repeat the process.

When reaching a pseudosource s itself (i.e. a saddle or boundary vertex), we iteratively move to windows adjacent to s, circumnavigating either clockwise or counter-clockwise, until finding a window with a new pseudosource.

3.6 Performance analysis

Let n be the number of the mesh edges. When the windows are propagated as a wavefront, it is shown in [Mitchell et al. 1987] that each edge may have $O(n)$ windows and therefore the total number of windows can be $O(n^2)$. This results in a worst case complexity of $O(n^2)$ space and $O(n^2 \log n)$ time. The $\log n$ factor is due to the need for a priority queue and for the binary searching required to find overlapping windows during window propagation.

However, for more typical meshes, we observe that edges have an average of $O(\sqrt{n})$ windows. Intuitively, given a mesh with uniformly distributed vertices, the number of edges can be thought of as being proportional to area, while the number of edges crossed by a shortest path is proportional to diameter. This intuitive reasoning gives us the expected $O(\sqrt{n})$ window-per-edge complexity.

To confirm this intuition, we constructed a series of subdivision meshes for several simple surfaces using the Loop subdivision scheme [1987], and ran our implementation. Let W_i and W_{i+1} be the total number of resulting windows on two subsequent levels of mesh subdivision. We estimate the exponent p in the $O(n^p)$ window complexity by evaluating $\log_4(W_{i+1}/W_i)$, since the number of mesh edges increases by 4 at each level. From Figure 6 we see that the window complexity grows as approximately $O(n^{1.5})$.

	Time	Faces	Windows	WPE	Exp
Eight (1)	0.055	1536	11859	5.15	
Eight (2)	0.453	6144	90986	9.87	**1.47**
Eight (3)	4.256	24576	705032	19.13	**1.48**
Eight (4)	56.103	98304	5383796	36.51	**1.47**
Sphere (1)	0.000	32	54	1.13	
Sphere (2)	0.002	128	310	1.62	**1.26**
Sphere (3)	0.011	512	2362	3.08	**1.46**
Sphere (4)	0.087	2048	17418	5.67	**1.44**
Sphere (5)	0.677	8192	132116	10.75	**1.46**
Sphere (6)	5.650	32768	997608	20.30	**1.46**
Sphere (7)	77.320	131072	8022292	40.80	**1.50**
Convex (1)	0.000	16	28	1.00	
Convex (2)	0.000	64	116	1.12	**1.03**
Convex (3)	0.003	256	692	1.73	**1.29**
Convex (4)	0.023	1024	5094	3.25	**1.44**
Convex (5)	0.199	4096	39996	6.44	**1.49**
Convex (6)	1.778	16384	318113	12.88	**1.50**
Convex (7)	19.971	65536	2564209	26.02	**1.51**
Saddle (1)	0.000	16	36	1.29	
Saddle (2)	0.001	64	216	2.08	**1.29**
Saddle (3)	0.007	256	1520	3.80	**1.41**
Saddle (4)	0.054	1024	11448	7.30	**1.46**
Saddle (5)	0.453	4096	87194	14.05	**1.46**
Saddle (6)	4.209	16384	663782	26.87	**1.46**
Saddle (7)	51.007	65536	5062828	51.37	**1.47**

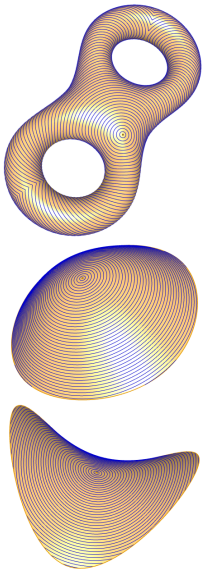


Figure 6: These various subdivision meshes show how the resulting number of windows increases as the mesh is subdivided. WPE is the average number of windows per edge; 'Exp' is the exponent p when estimating $O(n^p)$.

Knot	Time	WPE
Smooth	5.16	20.6
+10% noise	3.67	15.1
+20% noise	2.52	10.7
+40% noise	1.48	6.4
+80% noise	0.81	3.5

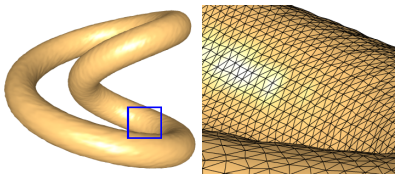


Figure 7: Performance of the exact algorithm when adding random geometrical noise to the smooth knot surface. The noise moves each vertex in a random direction by a uniform random distance from 0 to the indicated percentage of average edge length. The model and zoom-in shown have "20%" noise.

Surprisingly, the window complexity is even less when the mesh surface has a rough texture, as shown in the experiments of Figure 7. Intuitively, the bumpy features in the surface cause adjacent windows to overlap and thereby annihilate each other.

4 Approximation algorithm

In this section we introduce a method to compute an approximation $\mathcal{D}$ to the geodesic distance function D, which requires less time and space. The method works just like the exact algorithm, except for one key difference — before propagating a window, we try to *merge* it with an adjacent window on the same edge, as illustrated in Figure 8(a). The merging replaces two windows w_0, w_1 in the queue with a new window w' that covers $w_0 \cup w_1$. This involves choosing a new pseudosource s' that effectively replaces the previous approximate distance function with a new one. There are constraints as described in the next section, and merging is only performed if these constraints can be satisfied. The windows w_0 and w_1 are then deleted from the queue, and w' is inserted into the queue.

In addition, we prove in Appendix A that our approximation $\mathcal{D}$ is a *lower bound*, namely $\mathcal{D}(p) \leq D(p), \forall p \in \mathcal{S}$. This property is employed later in Section 5.

4.1 Constraints on window merging

Some conditions must be satisfied before we merge two windows.

Directionality: The two windows w_0, w_1 must have direction values τ in agreement.

Visibility: We define the *visibility region* of a window to be the area spanned by the rays exiting the window from the direction of the pseudosource, as illustrated in Figure 8(a). To guarantee that $\mathcal{D}$ is defined without gaps, the visibility region of the new window w' must at least cover the union of those of the merged windows.

Continuity: To maintain distance field continuity along the edges and at the vertices of the mesh, we must preserve distances at the endpoints of the merged window w'.

Accuracy: If window merging is performed indiscriminately, it usually results in slightly more than one window per edge on average, and is thus fast and consumes little memory. However, in practice it is desirable to bound the error

$$\max_{p=(x,0)\in w'} \Delta\mathcal{D}(p) = |\mathcal{D}'(p) - \mathcal{D}(p)|,$$

where $\mathcal{D}$ is the original and $\mathcal{D}'$ the merged approximate distance function. The difference $\Delta\mathcal{D}$ reaches its maximum either at the endpoints of w_i, $i=0,1$, or where $\frac{\partial}{\partial x}\Delta\mathcal{D} = 0$, which is equivalent to the following quadratic equation for each of the two w_i:

$$({s'_y}^2 - {s_{iy}}^2)x^2 + 2({s'_x}{s_{iy}}^2 - s_{ix}{s'_y}^2)x + ({s_{ix}}^2{s'_y}^2 - {s'_x}^2{s_{iy}}^2) = 0.$$

To measure the error $\mathcal{D} - D$ between our approximation and the exact distance function, we accumulate error by storing a scalar value ξ on each window, assigning $\xi' = \max(\xi_0, \xi_1) + \Delta\mathcal{D}(p)$. This is rather conservative but still works well in practice. Then, one possible test is to disallow the merge unless $\xi' \leq \varepsilon_{abs}$ where ε_{abs} is an *absolute error* threshold. Instead, we prefer to bound the *relative error* (the error measured as a fraction of geodesic distance) at the same point p, by testing if $\xi'/\mathcal{D}(p) \leq \varepsilon_{rel}$.

However, if we only bound the *global* (accumulated) error, many merges will occur near the source vertex v_s (until the error threshold is reached), leaving little opportunity for later merges farther from the source, and thus possibly resulting in an excessive number of windows. Our solution is to additionally bound the *local error*

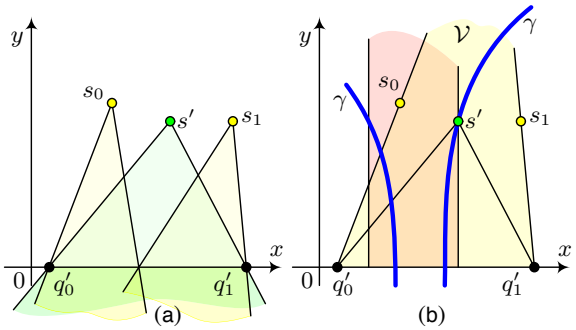


Figure 8: (a) Merging two windows into one such that the visibility region is not reduced. (b) Selecting the pseudosource for the new window. The visibility constraint $\mathcal{V}$ corresponds to the yellow region while the constraints of (4.3) define the pink region.

$\Delta\mathcal{D}(p)$ (actually, the relative local error $\Delta\mathcal{D}(p)/\mathcal{D}(p)$) by a fraction of the global error tolerance. We set this fraction to be 10% in all our experiments, although ideally this value should depend on the size of the mesh. Thanks to this heuristic, we are able to satisfy the global error bound using significantly fewer windows overall.

Monotonicity: We must be careful that the distance function over the edge is not smaller than the distance function over the parent windows from which it was propagated, where correspondence is defined by the propagation method. However, it is difficult to analytically check this property in practice. When we do not check this, loops in window propagation are possible. Because we maintain consistency of the source direction τ, our algorithm cannot produce any "bouncing" infinite loops, propagating back and forth between two adjacent edges. It is conceivable, though, that more exotic infinite loops could occur. Of course, we could explicitly maintain a directed graph representing the propagation evolution, explicitly check for loops in this graph, and disallow any attempted propagation steps that would complete a loop. In practice, we have never encountered such loops in any of our experiments, and so we do not explicitly perform these checks.

4.2 Finding the merged window pseudosource

We attempt to find a pseudosource position for the merged window that satisfies all the constraints of the previous section.

Denote the two adjacent windows w_i, with pseudosources s_i, for $i = 0, 1$. Using the same local coordinate frame used previously, the merged window w' will have two endpoints $q'_i \equiv (b'_i, 0)$. Our goal is to find a new pseudosource $s' = (s'_x, s'_y)$ and pseudosource distance σ' for the merged window that satisfy the following constraints.

To maintain continuity, we require that the geodesic distances at its endpoints $\mathcal{D}_i \equiv \mathcal{D}(q'_i) = \|s_i - q'_i\| + \sigma_i$ are preserved by the merge. This can be expressed as

$$(s'_x - b'_i)^2 + {s'_y}^2 = (\mathcal{D}_i - \sigma')^2.$$

It follows that

$$\sigma' = \alpha s'_x + \beta; \tag{4.1}$$

$${s'_y}^2 = A{s'_x}^2 + Bs'_x + C, \tag{4.2}$$

with $A = \alpha^2 - 1$, $B = 2\alpha(\beta - \mathcal{D}_0) + 2b'_0$, $C = (\mathcal{D}_0 - \beta)^2 - b'^2_0$, $\alpha = (b'_1 - b'_0)/(\mathcal{D}_1 - \mathcal{D}_0)$, $\beta = (b'^2_0 - b'^2_1 - \mathcal{D}^2_0 + \mathcal{D}^2_1)/(2(\mathcal{D}_1 - \mathcal{D}_0))$. This constrains s' to lie on a conic curve γ.

To maintain directionality, we must impose the inequality $s'_y \geq 0$. To satisfy the visibility constraint we require our solution to lie inside the sector between the two lines $\mathcal{L}_i, i = 0, 1$ that pass through the q'_i to s_i (Figure 8(b)). (If the intersection point of $\mathcal{L}_i$ has positive y-coordinate, then the allowed region $\mathcal{V}$ is a triangle. Otherwise $\mathcal{V}$ is open.)

To constrain σ' and the d'_i to be non-negative we add the inequalities $\sigma' \geq 0, \|q'_i - s'\| = \mathcal{D}_i - \sigma' \geq 0$. It follows from (4.1) that these inequalities are equivalent to:

$$\begin{cases} -\frac{\beta}{\alpha} \leq s'_x \leq \frac{\mathcal{D}_0 - \beta}{\alpha}, & \text{if } \alpha > 0, \\ \frac{\mathcal{D}_0 - \beta}{\alpha} \leq s'_x \leq -\frac{\beta}{\alpha}, & \text{if } \alpha < 0. \end{cases} \tag{4.3}$$

If all the above constraints are not simultaneously satisfiable we disallow the merge. Otherwise we pick the s' with minimal σ' value. This must occur when one of our inequality constraints is "tight".

4.3 Backtracing

The geodesic path for our approximation algorithm is traced similarly to the algorithm in Section 3.5 but with one essential difference. When window w is the result of merging two original windows, its pseudosource position is different from the pseudosource positions of those original windows. If we were to trace back in the direction of the merged pseudosource, the resulting path would be different from that represented in the forward propagation, and its overall length might exceed the computed error bound.

Our approach is to obtain the original pre-merge pseudosource by maintaining a list of references to the windows that were successively merged into w (together with their endpoints). The average length of these lists is only about 2 in all our experiments. Another benefit of using the correct pseudosource is that we can trivially guarantee that the source will be reached without any loops.

5 Exact geodesic between two vertices

Our goal is to find the geodesic path between a source vertex v_s and a target vertex v_t on the mesh. Note that it is inefficient to run our exact algorithm on the entire mesh (or even until reaching v_t). In this section we present an algorithm that performs a sequence of pruned searches, exploiting progressively tighter lower and upper bounds on geodesic distance, so that the final, exact algorithm need only be run on a "thin" region surrounding the solution.

Our approach can be seen as a "continuous A* search", in that it adapts the traditional edge-based A* algorithm [Pohl 1971] to meshes. A similar pruning approach is also explored in [Floater et al. 2002] although their scheme lacks true distance bounds.

Denote by $\mathcal{P}_{st}$ the geodesic path between v_s and v_t. Let $D_s(p)$ and $D_t(p)$ be the geodesic distances from point p to v_s and v_t respectively, and let $D_{st} = D_s(t) = D_t(s)$ be the length of $\mathcal{P}_{st}$. Then, it is obvious that any point p on $\mathcal{P}_{st}$ satisfies $D_s(p) + D_t(p) = D_{st}$.

If $L_s(p)$ and $L_t(p)$ are *lower-bound* functions of $D_s(p)$ and $D_t(p)$ respectively, and U_{st} is an *upper-bound* value for D_{st}, then any point p on $\mathcal{P}_{st}$ also satisfies

$$L_s(p) + L_t(p) \leq U_{st}. \tag{5.1}$$

This inequality is the core of the following algorithm:

Step 1: Using Dijkstra search on edges only, compute an upper-bound distance U_{st}(Dijkstra) by searching from v_s until v_t is reached. This step is made almost twice as fast using a bidirectional search, which runs two *simultaneous* Dijkstra searches from v_s and v_t until they both retire a common vertex.

Step 2: Start our *approximation* search (Section 4) from v_t until v_s is reached, which computes a lower-bound distance function $L_t(\cdot)$. During the search, we use the inequality (5.1) to prune the search by only propagating windows that have at least one point p satisfying

$$L_t(p) + \|p, v_s\| \leq U_{st}(\text{Dijkstra}),$$

where $\|\cdot,\cdot\|$ measures Euclidean distance (not on the mesh) and is therefore a trivial lower bound on $D_s(p)$.

Step 3: Using the windows provided by the previous step, apply backtracing (Section 4.3) to form a path from v_s back to v_t.

The length of this path defines a tighter upper-bound distance $U_{st}(\text{backtrace})$.

Step 4: Start our *exact* search (Section 3) from v_s until v_t is reached, which computes exact distance $D(v_s, \cdot)$. During the search, we again use the inequality (5.1) to propagate only windows that have at least one point p satisfying

$$D_s(p) + L_t(p) \leq U_{st}(\text{backtrace}).$$

Step 5: The geodesic distance between v_s and v_t has now been computed as $D_{st} = D_s(v_t)$. To obtain the geodesic path v_t back to v_s, apply backtracing (Section 3.5) using the windows provided by the previous step.

As future work, it would be useful to obtain even tighter A* bounds by precomputing distances to a set of landmark points on the mesh, as explored in [Goldberg and Harrelson 2005] for graph search.

6 Experimental results

We tested the algorithms on a Pentium M 1.6GHz PC with 1GB RAM. As shown in Table 1, our exact algorithm is useful even for large models. For instance, the exact geodesic distance from a source point to all vertices of the 400K-triangle David model is computed in 75 seconds. By comparison, the implementation of Chen and Han [1996] by Kaneva and O'Rourke [2000] runs successfully only on the 30K-triangle Buddha S model from our dataset, with a computation time of over 28 hours. In practice the main bottleneck of our exact algorithm is the memory space required to store all the windows. For 1GB memory, we are able to process a mesh of up to 700K faces. This space complexity constraint provides strong motivation for our approximate algorithm.

With a 0.1% relative error bound, our approximate geodesic algorithm runs significantly faster and uses much less memory than the exact algorithm. Table 1 reports both (1) the maximal absolute difference $|\mathcal{D}(v) - D(v)|$ between the approximate and exact distances, and (2) the average relative difference $|\mathcal{D}(v) - D(v)|/D(v)$. Absolute errors are reported as percentages of the object diameter.

We compare the accuracy of our approximate algorithm with the fast marching (FM) algorithm of [Kimmel and Sethian 1998]. Specifically, we used the FM implementation of Peyré and Cohen [2003; 2005], and verified that two other recent FM implementations [Reimers 2004; Sifri et al. 2003] produced identical distance results and had similar speed. Table 1 shows that our approximation algorithm has similar running times to the FM algorithm, but more importantly it has significantly better accuracy. The error distribution graph of Figure 11 shows that our algorithm also has much better accuracy than the improved update rule of [Kirsanov 2004].

Path results Figure 10 shows our point-to-point exact shortest path algorithm of Section 5 applied to a 1M-triangle Buddha XL model. It takes about 4 seconds to compute the path crossing half the model. Shortest paths between relatively closer vertices can be computed at interactive rates. For small models, paths between two arbitrary vertices can be computed in a matter of milliseconds.

		Exact		Aprroximate (0.1% rel)				Fast marching		
Model	Faces	time	WPE	time	WPE	max abs	ave rel	time	max abs	ave rel
Rockerarm	80,354	18.24	19.74	1.92	1.32	0.06%	0.04%	3.35	0.63%	0.84%
Horse	96,956	18.43	18.41	2.44	1.40	0.08%	0.05%	3.45	1.44%	0.75%
Dragon	100,000	6.45	7.18	3.53	1.85	0.09%	0.07%	5.81	1.09%	1.20%
Buddha S	30,000	1.03	4.57	0.97	1.95	0.14%	0.08%	1.23	2.15%	2.59%
Buddha M	199,272	24.43	11.83	6.17	1.57	0.08%	0.06%	11.03	0.56%	0.79%
David	399,710	75.13	16.72	11.13	1.48	0.11%	0.05%	18.15	0.49%	0.55%
Fandisk S	1,000	0.05	6.77	0.03	2.07	0.74%	0.07%	0.04	10.03%	5.09%
Fandisk U	9,926	0.78	10.72	0.14	1.01	0.05%	0.03%	0.21	1.21%	1.41%

Table 1: Comparison of our exact and approximation algorithms with fast-marching for the models of Figure 9. Times are in seconds; WPE indicates average number of windows per edge.

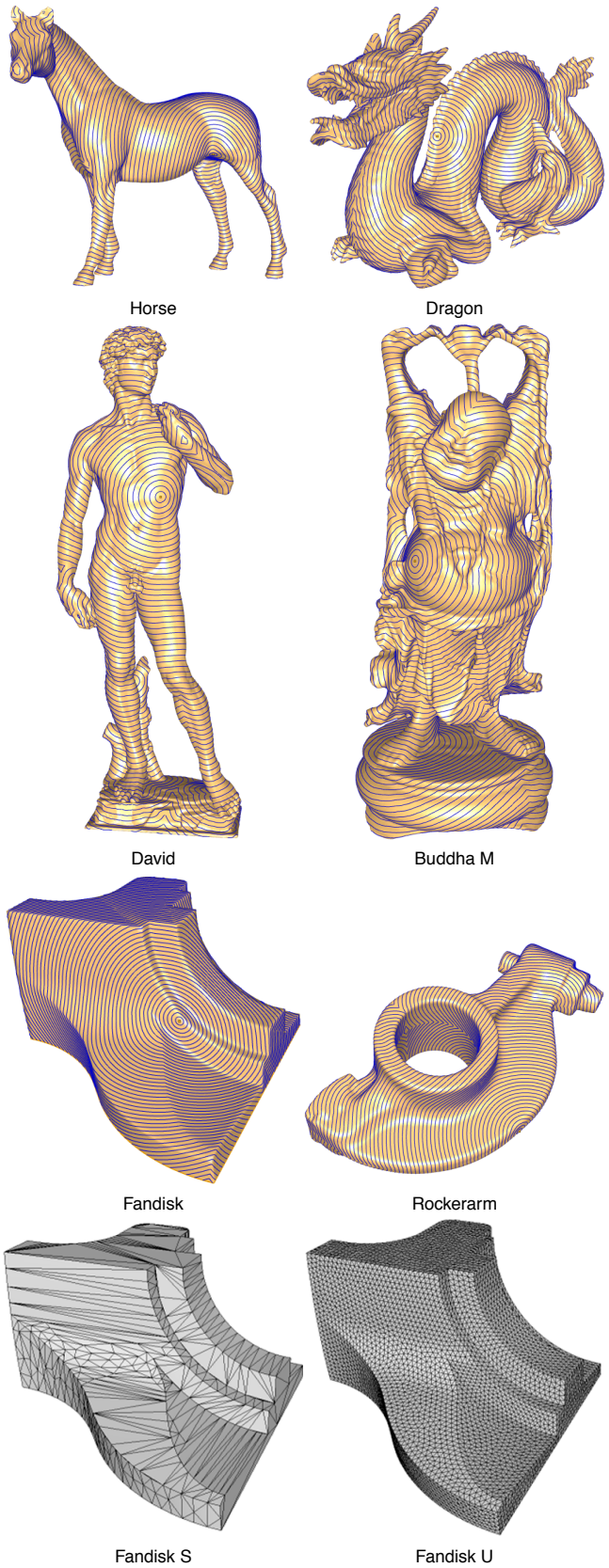
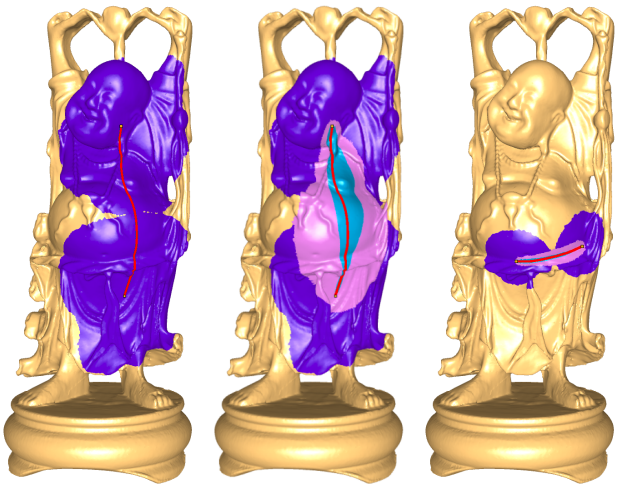


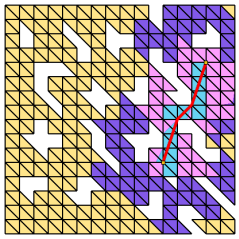
Figure 9: Models used for the tests in Table 1.



Time (sec)	Dijkstra	Approximation	Exact	Total
Buddha XL: Left two	0.466	1.327	2.561	4.354
Buddha XL: Right	0.098	0.188	0.229	0.515
Buddha M: Left two	0.032	0.228	0.228	0.519
Buddha M: Right	0.017	0.040	0.031	0.088

Figure 10: Illustration of our exact algorithm for finding the shortest path between pairs of vertices on the Buddha. The colored regions correspond to the pruned searches of successive search steps, and the final exact paths are shown in red. Timings in seconds are listed for both the 1M-face Buddha XL and 200K-face Buddha M meshes.

Our mesh geodesic algorithm also provides a practical solution to the problem of path planning in the presence of obstacles [Hershberger and Suri 1999]. In this setting, the mesh is a triangulation of a planar region, with holes in the mesh encoding the geometry of the obstacles. A simple example is shown in the inset figure.



7 Conclusions and future work

We have presented both exact and approximate algorithms for computing geodesic paths and distances on triangle meshes, and showed that these algorithms are fast enough to be practical on complicated meshes. For point-to-point geodesic computation, we developed a technique that uses a sequence of pruned searches with narrowing distance bounds to efficiently home in on the exact shortest path.

The bottleneck of the exact geodesic algorithm is the memory requirement due to the large number of windows. One idea to remedy this problem is to instead store a "pseudosource direction vector" at each mesh vertex. This direction vector is sufficient to repeatedly trace the geodesic path back (through successive face unfoldings) to preceding pseudosources. With this new data structure, windows can be deleted once they are retired from the queue.

Other future directions are to consider weighted or anisotropic distance metrics, and to generalize to higher-dimensional or smooth manifolds.

8 Acknowledgments

We thank Ron Kimmel, Martin Reimers, Michael Floater and Leonard McMillan for fruitful discussions. We also thank Gabriel Peyré, Martin Reimers and Oren Sifri for providing us with their implementations of the fast marching method. We are very grateful to the anonymous reviewers for their detailed constructive comments. This research was partially supported by the Research Council of Norway (BeMatA project).

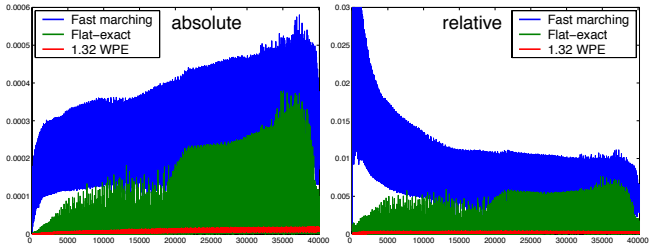


Figure 11: Error distribution on the rockerarm model (40K vertices). Vertices are sorted on the horizontal axis by their exact geodesic distance from the source. Flat-exact is the method of [Kirsanov 2004].

References

CHEN, J., AND HAN, Y. 1996. Shortest paths on a polyhedron; part I: computing shortest paths. *Int. J. Comp. Geom. & Appl. 6 (2)*, 127–144.

FLOATER, M. S., HORMANN, K., AND REIMERS, M. 2002. *Parameterization of manifold triangulations.*

FLOATER, M. S. 2005. Chordal cubic spline interpolation is fourth order accurate. *IMA J. Numer. Analysis.* to appear.

FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., , AND DOBKIN, D. 2004. Modeling by example. In *Proc. of SIGGRAPH 2004*, 652–663.

GOLDBERG, A. V., AND HARRELSON, C. 2005. Computing the shortest path: A* search meets graph theory. In *Proc. of the 16th Annual ACM-SIAM Symp. on Discrete Algorithms*, 156–165.

HERSHBERGER, J. E., AND SURI, S. 1999. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Computing 28 (6)* (Aug).

HILAGA, M., SHINAGAWA, Y., KOHMURA, T., AND KUNII, T. L. 2001. Topology matching for fully automatic similarity estimation of 3d shapes. In *Proc. of ACM SIGGRAPH 2001*, 203–212.

KANAI, T., AND SUZUKI, H. 2001. Approximate shortest path on a polyhedral surface and its applications. *Comp.-Aided Design 33, 11*, 801–811.

KANEVA, B., AND O'ROURKE, J. 2000. An implementation of Chen & Han's shortest paths algorithm. In *Proc. of the 12th Canadian Conf. on Comput. Geom.*, 139–146.

KAPOOR, S. 1999. Efficient computation of geodesic shortest paths. In *Proc. 32nd Annual ACM Symp. Theory Comput.*, 770–779.

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. on Graphics 22, 3* (Jul), 954–961.

KIMMEL, R., AND SETHIAN, J. A. 1998. Computing geodesic paths on manifolds. *Proc. of National Academy of Sci. 95(15)* (July), 8431–8435.

KIRSANOV, D. 2004. *Minimal Discrete Curves and Surfaces.* PhD thesis, Applied Math., Harvard University.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multiresolution modeling on arbitrary meshes. In *Proc. of SIGGRAPH 98*, 105–114.

KRISHNAMURTHY, V., AND LEVOY, M. 1996. Fitting smooth surfaces to dense polygon meshes. In *Proc. of SIGGRAPH 96*, 313–324.

LANTHIER, M., MAHESHWARI, A., AND SACK, J.-R. 1997. Approximating weighted shortest paths on polyhedral surfaces. In *Proc. 13th Annu. ACM Symp. Comput. Geom.*, 274–283.

LEE, H., KIM, L., MEYER, M., AND DESBRUN, M. 2001. Meshes on fire. In *Eurographics Workshop on Comp. Animation and Simulation*, 75–84.

LOOP, C. T. 1987. *Smooth subdivision surfaces based on triangles.* Master's thesis, Mathematics, Univ. of Utah.

MARTINEZ, D., VELHO, L., AND CARVALHO, P. C. 2004. Geodesic paths on triangular meshes. In *Proc. of SIBGRAPI/SIACG.*, 210–217.

MITCHELL, J. S. B., MOUNT, D. M., AND PAPADIMITRIOU, C. H. 1987. The discrete geodesic problem. *SIAM J. of Computing 16(4)*, 647–668.

MITCHELL, J. 2000. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Elsevier Science, 633–702.

NOVOTNI, M., AND KLEIN, R. 2002. Computing geodesic distances on triangular meshes. In *Proc. of WSCG'2002.*

PEYRÉ, G., AND COHEN, L. 2003. Geodesic re-meshing and parameterization using front propagation. In *Proc. of VLSM'03.*

PEYRÉ, G., AND COHEN, L. 2005. Geodesic computations for fast and accurate surface remeshing and parameterization. In *Progress in nonlinear differential equations and their applications*, vol. 63, 157–171.

PHAM-TRONG, V., SZAFRAN, N., AND BIARD, L. 2001. Pseudo-geodesics on three-dimensional surfaces and pseudo-geodesic meshes. *Numerical Algorithms 26*, 305–315.

POHL, I. 1971. Bi-directional search. In *Machine Intelligence*, vol. 6, 124–140.

POLTHIER, K., AND SCHMIES, M. 1998. Straightest geodesics on polyhedral surfaces. *Mathematical Visualization, Ed: H.C. Hege, K. Polthier, Springer Verlag*, 391–409.

PRAUN, E., HOPPE, H., AND FINKELSTEIN, A. 1999. Robust mesh watermarking. In *Proc. of SIGGRAPH 99*, 49–56.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proc. of SIGGRAPH 2000*, 465–470.

REIMERS, M. 2004. *Topics in Mesh based Modeling*. PhD thesis, Mathematics, Univ. of Oslo.

SANDER, P., WOOD, Z., GORTLER, S., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. *Symp. on Geometry Processing*.

SIFRI, O., SHEFFER, A., AND GOTSMAN, C. 2003. Geodesic-based surface remeshing. In *Proc. 12th Intnl. Meshing Roundtable*, 189–199.

SLOAN, P.-P. J., ROSE, C. F., AND COHEN, M. F. 2001. Shape by example. In *ACM Symp. on Interactive 3D Graphics*, 135–144.

ZHOU, K., SNYDER, J., GUO, B., AND SHUM, H.-Y. 2004. Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In *Symposium on Geometry Processing*.

ZIGELMAN, G., KIMMEL, R., AND KIRYATI, N. 2002. Texture mapping using surface flattening via multidimensional scaling. *IEEE Transactions on Visualization and Computer Graphics 8, 2*, 198–207.

A Lower bound for the exact geodesic distance

Suppose we merge the windows w_0 and w_1 to form a single window w'. We prove that the distance function defined by w' is *no greater* than the distance function defined by the two original windows w_0 and w_1, i.e. $\forall p \in w',\ \|p - s'\| + \sigma' \leq \|p - s_i\| + \sigma_i, i = 0, 1.$

Consider the bisector between s_0 and s', i.e. the locus of points p such that

$$\sigma_0 + \|s_0 - p\| = \sigma' + \|s' - p\|.$$

The bisector curve is a hyperbola branch, since the difference between the distances to s_0 and s' is constant. This hyperbola divides the plane into two (Voronoi) regions, one closer to s_0 and another closer to s'. The bisector intersects the x-axis in at most two points, at q'_0 and perhaps somewhere else. See Figure 8(a).

By the visibility constraint, s' is to the right of the line defined by s_0 and q'_0, and hence the x-axis immediately to the right of q'_0 is in the region closer to s'. Since s_1 is closer to q'_1 than s_0: $\sigma_0 + \|s_0 - q'_1\| > \sigma_1 + \|s_1 - q'_1\|$, and $\sigma_1 + \|s_1 - q'_1\| = \sigma' + \|s' - q'_1\|$ by construction, q'_1 is on a segment of the x-axis that is closer to s' than to s_0:

$$\sigma_0 + \|s_0 - q'_1\| > \sigma' + \|s' - q'_1\|$$

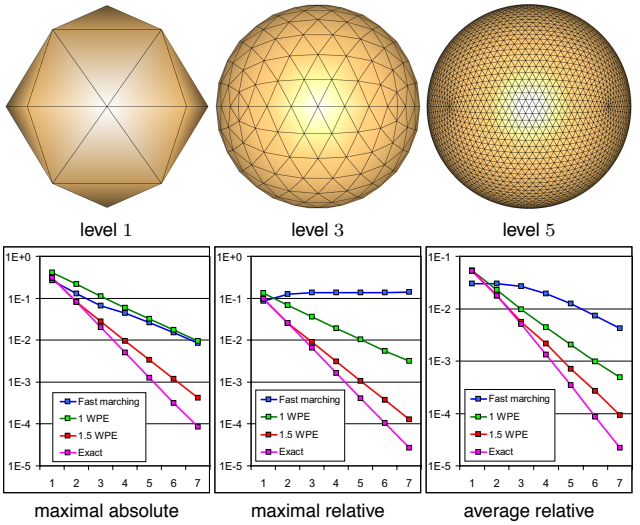
and hence the whole w' is closer to s' than to s_0.

Analogously, it can be proven that w' is closer to s' than to s_1, by considering the bisector between s_1 and s'.

B Experiments with smooth surfaces

In this paper we have focused on the problem of computing geodesic distances over meshes. However, in computer graphics, meshes frequently represent an underlying smooth surface and thus, it is informative to examine the difference between the analytic geodesic distances on the smooth surface and the distances computed along the mesh.

For numerical comparison, we have chosen a spherical surface because it has nonzero curvature and a simple analytic distance function $\mathcal{G}(\cdot)$. We consider seven approximation levels of the sphere obtained by regularly subdividing an octahedron using Loop's scheme, with vertices projected onto the sphere; the coarsest mesh has 18 vertices and the finest one has $64K$ vertices.



	maximal absolute						average relative					
level	Exact	R	1.5 WPE	R	FM	R	Exact	R	1.5 WPE	R	FM	R
1	0.30170		0.30170		0.266		0.05197		0.05197		0.030	
2	0.08052	**1.91**	0.08052	**1.91**	0.127	**1.07**	0.01760	**1.56**	0.01760	**1.56**	0.030	**0.00**
3	0.02020	**1.99**	0.02796	**1.53**	0.067	**0.93**	0.00506	**1.80**	0.00552	**1.67**	0.027	**0.17**
4	0.00499	**2.02**	0.00959	**1.54**	0.044	**0.61**	0.00133	**1.92**	0.00213	**1.38**	0.019	**0.47**
5	0.00125	**2.00**	0.00333	**1.52**	0.027	**0.72**	0.00034	**1.97**	0.00071	**1.58**	0.012	**0.65**
6	0.00031	**2.00**	0.00117	**1.51**	0.015	**0.80**	0.00009	**1.98**	0.00027	**1.40**	0.007	**0.75**
7	0.00008	**1.99**	0.00041	**1.50**	0.008	**0.85**	0.00002	**1.98**	0.00009	**1.53**	0.004	**0.82**

Figure 12: The first row shows three of the approximating meshes. The second row shows errors for the meshes on seven subdivision levels. The table shows the errors and convergence rates.

	maximal absolute					average relative				
level	1 WPE	1.5 WPE	2 WPE	2.5 WPE	Exact	1 WPE	1.5 WPE	2 WPE	2.5 WPE	Exact
2	0.94				1.91	1.25				1.56
3	0.94	1.53			1.99	1.22	1.67			1.80
4	0.90	1.54	1.65		2.02	1.15	1.38	1.81		1.92
5	0.89	1.52	1.87	1.85	2.00	1.10	1.58	1.82	1.88	1.97
6	0.89	1.51	1.84	1.90	2.00	1.07	1.40	1.64	1.86	1.98
7	0.89	1.50	1.81	1.90	1.99	1.00	1.53	1.79	1.85	1.98

Table 2: Convergence rates of our approximation and exact algorithms for the sphere model. See also Figure 12. When the average number of windows per edge (WPE) increases, the convergence rate approaches the exact algorithm rate. (Missing cells denote values identical to the exact algorithm.)

To analyze the algorithms we measure errors using the maximal absolute difference between $\mathcal{G}(v)$ and the distances $D(v)$ provided by the algorithms, as well as the maximal and average relative differences $|D(v) - \mathcal{G}(v)|/\mathcal{G}(v)$ between the distances. In addition, we numerically compute the convergence rate $R = \log_2(E_{i-1}/E_i)$ where E_i are errors at successive subdivision levels. Figure 12 compares our exact and approximate algorithms with the FM method. The results indicate that the exact algorithm has second-order convergence, which is in agreement with the results of [Floater 2005] for approximation of curves.

For the results in Figure 12, we maintained an average of 1.5 windows per edge (WPE) by manually adjusting the relative error bound ε_{rel} across subdivision levels. Note that our approximation algorithm is adaptive with respect to the number of windows per edge, since it allocates more windows to "difficult" mesh regions. Table 2 demonstrates that when WPE increases, the convergence rate grows and approaches that of the exact algorithm. These results also indicate that setting WPE to a value as small as 1.5 already increases convergence rate substantially.