Fast Wavefront Propagation (FWP) for Computing Exact Geodesic Distances on Meshes

Chunxu Xu, Tuanfeng Y. Wang, Yong-Jin Liu, *Member, IEEE,*
Ligang Liu, *Member, IEEE,* and Ying He, *Member, IEEE*

Abstract—Computing geodesic distances on triangle meshes is a fundamental problem in computational geometry and computer graphics. To date, two notable classes of algorithms, the Mitchell-Mount-Papadimitriou (MMP) algorithm and the Chen-Han (CH) algorithm, have been proposed. Although these algorithms can compute exact geodesic distances if numerical computation is exact, they are computationally expensive, which diminishes their usefulness for large-scale models and/or time-critical applications. In this paper, we propose the fast wavefront propagation (FWP) framework for improving the performance of both the MMP and CH algorithms. Unlike the original algorithms that propagate only a single window (a data structure locally encodes geodesic information) at each iteration, our method organizes windows with a bucket data structure so that it can process a large number of windows simultaneously without compromising wavefront quality. Thanks to its macro nature, the FWP method is less sensitive to mesh triangulation than the MMP and CH algorithms. We evaluate our FWP-based MMP and CH algorithms on a wide range of large-scale real-world models. Computational results show that our method can improve the speed by a factor of 3-10.

Index Terms—Discrete geodesic, fast wavefront propagation, algorithm complexities

1 Introduction

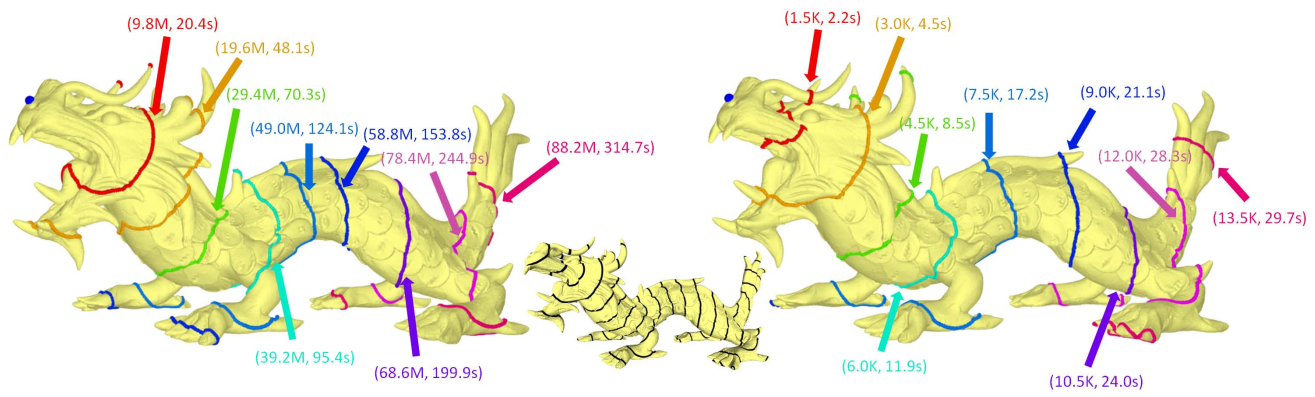
How to compute shortest paths on polyhedral surfaces is a fundamental problem in computational geometry and computer graphics, which has been studied for almost three decades [1]. To date, there are two notable classes of algorithms, namely, the Mitchell-Mount-Papadimitriou (MMP) algorithm [2] and the Chen-Han (CH) algorithm [3], which can compute exact geodesic distances on triangle meshes if numerical computation is exact. Although they are based on different domain subdivision strategies, these two algorithms adopt a similar data structure called window, which locally encodes the geodesic information.

It is known that both the MMP and CH algorithms produce $O(n^2)$ windows on an n-face triangle mesh and the upper bound is tight [4]. Therefore, any window-based discrete geodesic algorithm cannot run faster than $O(n^2)$ theoretically, which is known as the quadratic time barrier. Interestingly, as shown in this paper, the correctness of the MMP and CH algorithm is independent of the order of the windows being processed. However, propagating windows in an arbitrary order results in an extremely poor performance. The MMP algorithm keeps windows in a priority queue, where the window closest to the source is taken at each iteration. Since each window operation (i.e., choosing a window from the priority queue and propagating it) takes $O(\log n)$ time, the MMP algorithm has an $O(n^2 \log n)$ time complexity. The CH algorithm, in contrast, maintains windows in a hierarchical structure and processes them in a breadth-first-search order, resulting in an $O(n^2)$ time complexity. However, computational results in [5] [6] show that the MMP algorithm runs much faster than the CH algorithm.

Xin and Wang [7] observed that the slow performance of the CH algorithm is mainly due to the large amount of useless windows processed. They proposed a simple yet effective window filter to detect the useless windows, accompanied by a priority queue for window organization. The improved CH algorithm, called ICH, has a speed comparable to the MMP algorithm, however, its theoretical time complexity becomes $O(n^2 \log n)$ due to the priority queue. To date, developing an $O(n^2)$ exact discrete geodesic algorithm with good practical performance is still a great challenge.

This paper tackles this challenge by proposing a fast wavefront propagation (FWP) framework, which bridges the gap between theoretical time complexity and practical performance of discrete geodesic algorithms. Unlike the MMP and ICH algorithms that propagate only a single window (the one closest to

- *C. Xu and Y-J. Liu are with the TNList, Department of Computer Science and Technology, Tsinghua University, China.*
- *T. Wang and L. Liu are with School of Mathematical Sciences, University of Science and Technology of China.*
- *Y. He is with School of Computer Engineering, Nanyang Technological University, Singapore.*
Corresponding authors: Yong-Jin Liu & Ying He



MMP: $89,047,334$ iterations, 320.6s Dragon: 4M faces FWP-MMP: $13,748$ iterations, 31.7s

Fig. 1. Choose the source vertex at the nose of the 4M-face Dragon model. The existing undiscretized geodesic algorithms (e.g., MMP and ICH) have poor performance since they propagate the wavefronts very slowly. Our fast wavefront propagation technique can significantly improve their speed. Each colored curve is a *discrete* wavefront and the 2-tuple associated with each wavefront is the iteration number and the corresponding time, which was measured on a PC with an Intel Core i7-2600 CPU (3.40 GHz). We draw only a few representative wavefronts to avoid clutter. The small inset in the middle illustrates the computed geodesic distances using iso-distance contours.

the source) at each iteration, our method organizes windows in a bucket data structure so that it is able to process a large number of windows simultaneously without compromising wavefront quality. Although a window may enter and leave the bucket multiple times, our method guarantees that the window complexity is still $O(n^2)$. Since the practical overhead required for each iteration is very small, our FWP-based MMP and CH algorithms run faster than the original algorithms. Computational results on real-world models show that our method improves the performance by a factor of 3-10. See Figure 1 for an example. Intuitively speaking, the performance improvement by our method is due to its efficient data organization on a *macro* scale (i.e., focusing on wavefronts consisting of many windows), whereas the existing algorithms are on a *micro* scale (i.e., focusing on an individual window). Thanks to its macro nature, the FWP-based methods are also less sensitive to mesh resolution than the MMP and CH algorithms, i.e., increasing the mesh anisotropy may significantly slow down the existing algorithms, but it affects the FWP-based methods slightly.

In this paper our contributions are twofold:

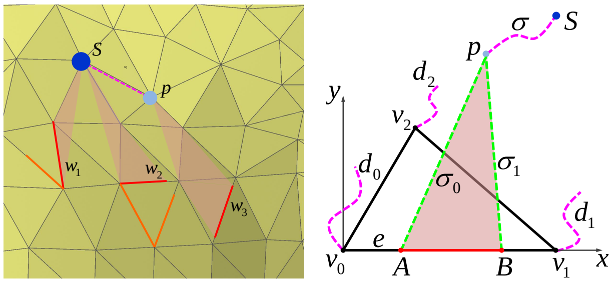
First, from a theoretical perspective, the FWP framework unifies the two classes of algorithms from the macro scale: the FWP-CH and FWP-MMP algorithms propagate the wavefronts at a similar pace and they converge in roughly the same number of iterations, although their window propagation schemes are very different. The FWP framework has provable time and space complexity: the FWP-CH and FWP-MMP algorithms have $O(n^2)$ and $O(n^2 \log n)$ time complexity, respectively.

Second, from a practical perspective, the FWP technique is easy to implement and it can speedup the MMP and CH algorithms significantly. It is worth noting that the FWP-MMP algorithm can improve the performance of the MMP algorithm by an order of magnitude on large-scale real-world models, making it comparable to the state-of-the-art GPU-based parallel Chen-Han algorithm [8]. As a macro algorithm, the FWP-CH and FWP-MMP algorithms are also less sensitive to mesh triangulation than the existing micro algorithms. We also demonstrate that the FWP-based algorithm can be applied to the pre-computation methods, such as Saddle Vertex Graph (SVG) [9] and Geodesic Triangle Unfolding (GTU) [10].

2 Related Work

Classic techniques for computing discrete geodesics on triangle meshes include the computational geometry approaches and the partial differential equation (PDE) approaches. The former includes the abovementioned MMP/CH algorithms and their many variants [5] [6] [7] [8] [11] [12]. The latter consists of the popular fast marching method (FMM) [13], [14] and the gradient-based approaches [15][16]. See [1] for a comprehensive survey of classic techniques.

Each type of technique has its own merits and limitations. The undiscretized methods such as MMP and CH in computational geometry approaches can obtain the exact geodesics on arbitrary triangle meshes if numerical operations are exact. As a comparison, the PDE approaches provide only the approximate solutions (e.g., the first-order approximation by the FMM), which may be poor on meshes with highly irregular tessellation. On the other hand, the PDE approaches are usually faster than the computational geometry approaches. But they assume that the triangle meshes are discrete samples of underlying smooth surfaces and proving the convergence of the discrete geodesic



(a) Covered regions of windows (b) 2D parameterization

Fig. 2. (a) A window w is an interval (drawn in red) on a mesh edge such that the geodesic paths from the source to any point in w have the same face sequence (colored in pink). Propagating a window across an edge produces one or more child windows. w_1 has one child and w_2 has two children. Both w_1 and w_2 are directly visible from the source s, but w_3 is not. Instead, w_3 is visible from the pseudo source p. (b) Parameterizing a window to $\mathbb{R}^2$ locally encodes the geodesic distance from the source s to any point on the window.

distance to its smooth counterpart is of important theoretical value; e.g., uniform convergence of geodesics is proved in [17] under the assumption of convergence of surfaces in Hausdorff distance.

Recently precomputation techniques have been proposed, which aim at balancing quality and performance for computing various types of discrete geodesics. Xin et al. [10] proposed the Geodesic Triangle Unfolding (GTU) method, which flattens the curved geodesic triangle onto $\mathbb{R}^2$ and then uses Euclidean distance to approximate geodesic distance. The heat method, proposed by Crane et al. [16], is an elegant gradient based approach, which recovers the geodesic distance from the normalized gradient of the heat flow. By pre-factoring the Laplacian matrix, both the heat flow and the distance computation can be done in near-linear time. The heat method is easy to implement. It is also flexible to support a wide range of geometric domains, including grids, triangle meshes and point clouds. Such a feature is not available in the computational geometry approaches that work only for triangle meshes. However, similar to the FMM, it provides only a first order approximation.

Observing that the discrete geodesic problem has a surprisingly strong local structure due to the existence of the saddle vertices, Ying et al. [9] proposed another precomputation technique called the saddle vertex graph (SVG), a sparse graph which encodes the geodesic information on triangle meshes. With the SVG, computing the polyhedral distance is equivalent to finding the shortest path on the graph. Note that the pre-computation of both the GTU and SVG methods heavily depends on the MMP or ICH algorithm. As the proposed FWP technique improves their performance significantly, we show that it can be adopted in the GTU and SVG methods to reduce their precomputation time.

3 Preliminary

Let $M = (V, E, F)$ be a triangle mesh, where V, E and F are the sets of vertices, edges and faces, respectively. Given a source point $s \in V$, Mitchell et al. [2] showed that a geodesic path from s to vertex v_j passes through a sequence of mesh faces. A window is an interval I defined on a mesh edge such that the geodesic paths from s to any point in I share the same face sequence. See Figure 2(a). Mitchell et al. also showed that a geodesic path cannot pass through any spherical vertex (a vertex at which the sum of surrounding angles is less than 2π) unless it is the destination, since perturbing the path a bit off the spherical vertex reduces its length. However, a geodesic path may pass through one or more saddle vertices (a vertex at which the sum of surrounding angles is greater than 2π). The saddle vertex nearest to the destination is called a *pseudo source*. A window associated with a half-edge e is a 6-tuple $(\sigma, A, B, \sigma_0, \sigma_1, e)$ [5] where

- σ is the distance from the pseudo source p to the source s;
- A and B are the left and right endpoints of the interval;
- σ_0 and σ_1 are the distances from I's endpoints to p.

With this window data structure, we can easily position the source or pseudo source in the unfolded face/edge sequences and compute the geodesic distance for any point inside the interval. See Figure 2(b). Note that the MMP algorithm stores oriented windows so that each side of a non-boundary edge contains windows, whereas the CH algorithm does not require edge orientation.

The MMP and CH algorithms maintain a vector $(d_1, \cdots, d_n)$, $n = |V|$, for the polyhedral distances defined on mesh vertices, and a set of windows $\mathcal{W}$. Initially, we have $d_s = 0$ and $d_i = \infty$ for $i \neq s$. Set $\mathcal{W}$ contains the windows covering the edges opposite to the source vertex s. The algorithms then iteratively propagate windows across the faces and update the polyhedral distances when a window covers a vertex or part of an edge, until the set $\mathcal{W}$ is empty. Upon termination, label d_i is the geodesic distance from the source s to vertex v_i. The computational framework of the MMP and CH algorithms is as follows:

```
while $\mathcal{W}$ is not empty do
    extract a window $w = (\sigma, A, B, \sigma_0, \sigma_1, e)$ from $\mathcal{W}$;
    if $e$ is not a boundary edge then
        propagate $w$ across $e$ to produce child windows $\widehat{w}$;
        update the distance of vertex/edge covered by $\widehat{w}$;
        add $\widehat{w}$ to $\mathcal{W}$;
    end if
end while
```

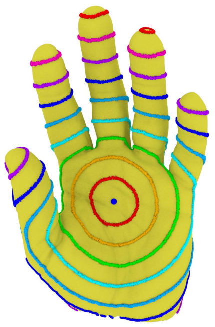
Both the MMP and CH algorithms have $O(n^2)$ window complexity. They are distinguished by the data structure for organizing the windows and the order of window processing. The MMP algorithm maintains a priority queue for the windows and takes the window *closest* to the source in each iteration. As a result, the MMP algorithm has an $O(n^2 \log n)$ time complexity. The CH algorithm records the parent-child relationship of windows in a hierarchical tree structure and processes the windows in a breadth-first-search order, which is implemented by a first-in-first-out (FIFO) queue. Therefore, the CH algorithm has an $O(n^2)$ time complexity. The ICH algorithm adopts a simple-yet-effective window filter, which can reduce many useless windows. Furthermore, it uses the priority queue to organize the windows according to their distance back to the source. The ICH algorithm, with a time complexity $O(n^2 \log n)$, significantly outperforms the CH algorithm in terms of speed.

4 Fast Wavefront Propagation

A wavefront in a continuous setting is the locus of points having the same distance to the source. The MMP/CH/ICH algorithms maintain a *discrete* wavefront, which can be formally defined as follows:

Definition 1. *The i-th wavefront, denoted by W_i, is the union of windows in $\mathcal{W}$ at the i-th iteration.*

A wavefront depends on the model, the location of the source point, as well as the time, and it may be unconnected. As the right inset shows, each color corresponds to one wavefront. At the beginning, the wavefront is connected and has a circular shape. Later, it evolves to several connected components. Finally, each connected component shrinks to a maximal point (where the geodesic distance reaches the local maximum) and then vanishes. For a window w, we define its label as the shortest distance from the source to w's associated mesh edge. The *size* of W_i, denoted by $|W_i|$, is determined by the number of windows it has. The standard deviation of all window labels in W_i, $std(W_i)$, is a good measure of wavefront quality. Intuitively speaking, the smaller the variance, the higher quality the wavefront has. Wavefront quality depends on the order of windows being processed. As shown in Figure 3, propagating windows in the smallest-label-first order leads to a high-quality wavefront (i.e., smooth and with small variance), whereas using the first-in-first-out order produces a low-quality wavefront (i.e., very rough and with large variance).



The MMP and ICH algorithms always take the window with the minimal label at each iteration, resulting in a high-quality wavefront. However, the overhead required for each iteration is expensive. Computational results show that more than 60% of the time is used for maintaining a priority queue in the MMP and ICH algorithms. See Figure 4. On the other hand, the CH algorithm organizes the windows in an FIFO queue, leading to a constant time overhead. However, as its wavefront is of poor quality, the CH algorithm produces many useless windows and converges very slowly.

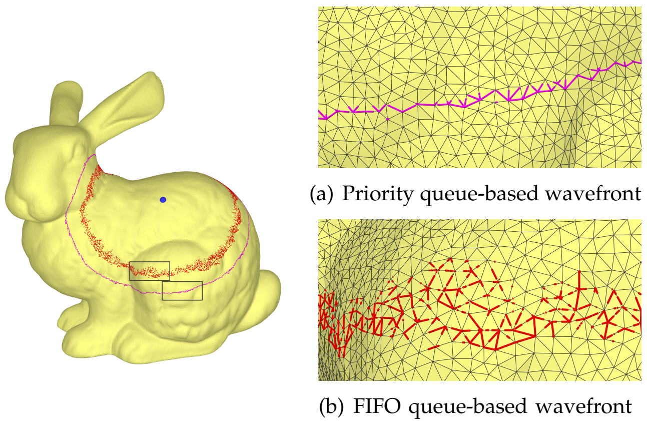

(a) Priority queue-based wavefront
(b) FIFO queue-based wavefront

Fig. 3. The MMP algorithm maintains a priority queue for the windows, leading to a high-quality wavefront with standard deviation $std = 0.00219$. Replacing the priority queue by an FIFO queue results in a very poor and slow-moving wavefront with $std = 0.02431$. The model has been scaled to a unit cube.

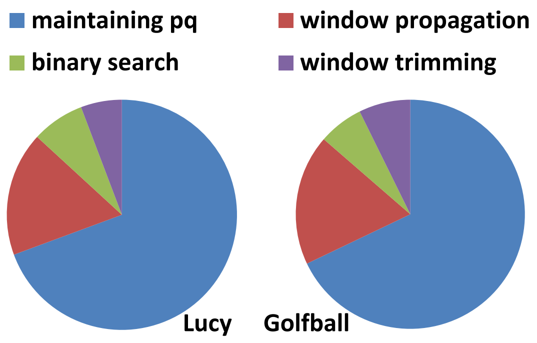


Fig. 4. Maintaining a priority queue (pq) takes roughly 70% of the runtime of the MMP algorithm. The ICH algorithm has a similar percentage.

Our idea is to balance the wavefront quality and the overhead for updating wavefronts. Unlike the MMP/ICH algorithms that propagate only the window with the smallest label in each iteration, our method propagates at least K smallest-label windows at the same time. We propose a bucket data structure to organize windows so that it takes only $O(1)$ time to process each window. In addition, K is adaptive to the wavefront size, and it has a constant upper bound, leading to an $O(n^2)$ worst-case window complexity. We call our method *fast wavefront propagation*, which distinguishes itself from the existing slow-propagating algorithms such as MMP and ICH.

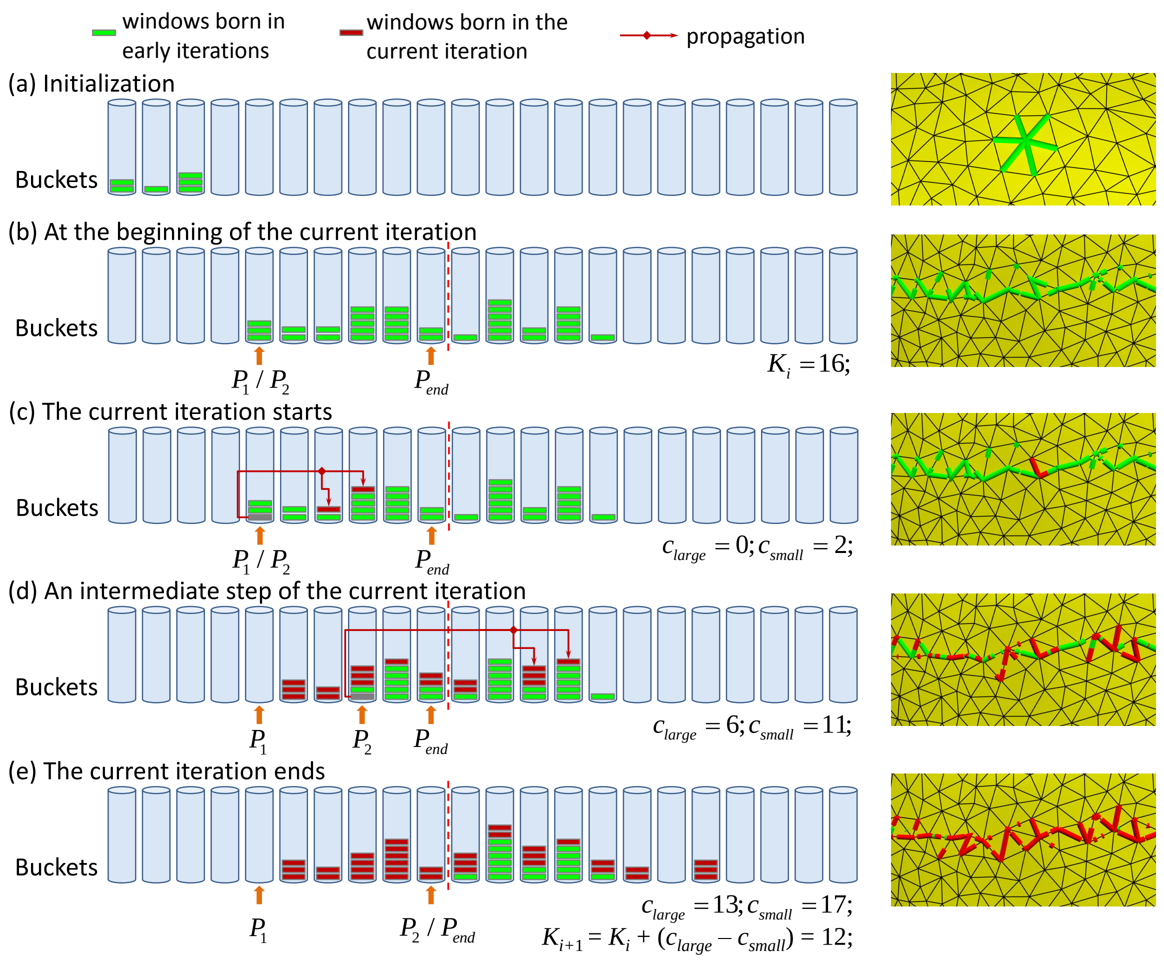

windows born in early iterations
windows born in the current iteration
propagation
(a) Initialization
Buckets
(b) At the beginning of the current iteration
Buckets
P_1 / P_2
P_{end}
$K_i = 16;$
(c) The current iteration starts
Buckets
P_1 / P_2
P_{end}
$c_{large} = 0; c_{small} = 2;$
(d) An intermediate step of the current iteration
Buckets
P_1
P_2
P_{end}
$c_{large} = 6; c_{small} = 11;$
(e) The current iteration ends
Buckets
P_1
P_2 / P_{end}
$c_{large} = 13; c_{small} = 17;$
$K_{i+1} = K_i + (c_{large} - c_{small}) = 12;$

Fig. 5. A typical iteration of the FWP algorithm. (a) Initialize by creating a window for each edge facing the source vertex and putting it into some bucket according to its label. (b) At the beginning of each iteration, determine the value of K and find the first non-empty bucket, labeled as P_1. In this example, $K = 16$. The pointer P_2 is used to track the to-be-processed bucket. At this moment, $P_2 = P_1$. Find the bucket buckets$[P_{end}]$ so that at least 16 windows will be propagated in this iteration. (c) The first window in buckets$[P_2]$ generates two child windows, both of which are placed before P_{end}. Therefore, increase c_{small} by 2. Here c_{small} (resp. c_{large}) is the number of windows in the next iteration whose distances are less (resp. greater) than the distance of the window pointed by P_{end}. The existing windows (born in early iterations) are colored in green and the new windows (born in the current iteration) are in red. (d) In an intermediate step of the current iteration, another green window is propagated. The two child windows are placed after P_{end}, so increase c_{large} by 2. Move on to the next bucket when all of the green windows in buckets$[P_2]$ have been processed. (e) The current iteration is over when all of the buckets before P_{end} are done. Observe that $c_{large} < c_{small}$, meaning that the majority of the child windows are not far. As a result, the wavefront quality is not good. So we decrease K and propagate fewer windows in the next iteration.

4.1 Algorithm

Let $s \in V$ be the source vertex and $p \in M$ a point (not necessarily a mesh vertex) on M. Denote by $d(p)$ the geodesic distance between s and p. Obviously, $d(\cdot)$ is a continuous function. We partition the polyhedral distance into equal-length intervals, $[0, l)$, $[l, 2l)$, $[2l, 3l)$, etc. Each interval is called a *bucket*, which is used to organize windows. Observe that the maximum range of geodesic distances in most real-world models[1] is $O(\sqrt{n}h)$, where h is the average edge length. There are $O(n^2)$ windows and each bucket contains roughly $O(n)$ windows, so we set the bucket size $l = h/\sqrt{n}$.

We assign each window w a birth time, that is, the iteration when w was created. We put w into the i-th bucket $[(i-1)l, il)$, denoted by B_i, if w's label is in this range. Windows in a bucket are organized by an FIFO queue. The initial wavefront consists of windows on the edges opposite to the source s. These windows are put into the corresponding buckets according to their labels. The FWP algorithm then iteratively propagates the wavefront. In the i-th iteration, the FWP algorithm selects at least K smallest-label windows on the wavefront and propagates them across their adjacent

1. In a few extreme pathological cases, the maximum geodesic range can reach $O(nh)$. However, our bucket strategy still works well in practice with $l = h/\sqrt{n}$.

triangles. The number K is adapted to the size and quality of the wavefront, and K can be determined automatically. See Section 4.2.

Three pointers P_1, P_2 and P_{end} are used. The pointer P_1 points to the first non-empty bucket and the pointer P_2 is used to track the to-be-processed bucket. For each window w in buckets[P_2] who were born in some iteration earlier than i, we propagate w across its adjacent triangle, and obtain one or more child windows. Since a child window always has a larger distance than its parent, it cannot be placed in a bucket before its parent. Note that some new windows (who are born in the current iteration) may be added to the bottom of the queue in buckets[P_2]. If so, we skip these windows (colored in red in Figure 5), and move P_2 to the next non-empty bucket.

The current iteration is over when it processes at least K smallest-label windows who were born early (i.e., reaching the bucket pointed to by P_{end} that is determined at the beginning of current iteration). The FWP algorithm terminates when all of the buckets are empty. Figure 5 illustrates a typical iteration of the FWP algorithm. See Algorithm 1 for the pseudocode.

The proposed FWP algorithm is a general framework for organizing and propagating windows so that it can be applied to both the MMP and ICH algorithms. In the following, we refer to the FWP-based MMP and ICH algorithms as FWP-MMP and FWP-CH, respectively.

Algorithm 1 Fast Wavefront Propagation Algorithm

```
Input: a mesh $M = (V, E, F)$ and a source point $s$;
Output: the undiscretized geodesic distance for each vertex;
for each edge $e$ facing $s$ do
    generate a window $w$ for $e$ with $w$.birth = $-1$;
    insert_window($w$);
end for
$K = P_1 = c_{small} = iter = 0$, $c_{large}$ = wavefront_size;
while wavefront_size $\neq 0$ do
    $K$ += $(c_{large} - c_{small})$;
    $K = \min(\max(K, 1)$, wavefront_size, $K_{max})$;
    // find the first non-empty bucket
    while buckets[$P_1$].is_empty() do
        $P_1$ ++;
    end while
    // find $P_{end}$ so that at least $K$ smallest-label
       windows will be propagated
    $P_{end} = P_1$, $wc = 0$;
    while $wc < K$ do
        $wc$ += buckets[$P_{end}$].Q.size();
        $P_{end}$ ++;
    end while
    $P_2 = P_1$;
    while $P_2 \leq P_{end}$ do
        //propagate the windows born in early iterations
        while buckets[$P_2$].top().birth $< iter$ do
            $w$ = extract_window($P_2$);
            propagate $w$ across its adjacent triangle;
            for each child window $\widehat{w}$ do
                if $\widehat{w}$.dist() $\geq P_{end} * l$ then
                    // $\widehat{w}$ is in a bucket after $P_{end}$
                    $c_{large}$ ++;
                else
                    $c_{small}$ ++;
                end if
                $\widehat{w}$.birth = $iter$;
                insert_window($\widehat{w}$);
            end for
        end while
        $P_2$ ++;
    end while
    $iter$ ++;
end while
```

4.2 Adaptive Adjustment of K

Setting $K = 1$ is too conservative, since only windows in the first non-empty bucket propagate in each iteration and the overhead required for each iteration is expensive, akin to the MMP/ICH algorithms. On the other hand, an extremely large K means that all the windows on the wavefront are propagated at once, that is, without taking their distances into account. As a result, the wavefronts are of low quality and the FWP algorithm becomes the highly inefficient FIFO-based algorithm. Thus, an extremely large K is too aggressive. We do expect a proper K for both high-speed wavefront propagation and the wavefronts of high quality. Since the time-dependent wavefronts may change dramatically throughout the iterative procedure, the K value should be adaptive to the wavefront's size and quality, and are updated at each iteration.

Consider the i-th wavefront W_i. Denote by K_i the windows propagated in i-th iteration. Since K_i is the number of windows in buckets whose pointers range from P_1 to P_{end}, K_i is always equal to or larger than K. Let τ denote the K_i-th smallest label of the windows in W_i. Then we partition W_i into two sets, W_i^1 and W_i^2, where W_i^1 consists of the windows with distances less than τ and $W_i^2 = W_i \backslash W_i^1$ contains the remaining windows. Note that the FWP algorithm propagates only the windows in W_i^1 and the generated child windows are denoted by C_i. Let c_{large} (resp. c_{small}) be the number of windows in C_i whose distance is larger (resp. smaller) than τ. We have the following two observations:

- If $c_{small} < c_{large}$, the majority of the child windows have distances more than τ, meaning that the propagated K_i windows in the i-th iteration have a high possibility to survive in the next iteration and consequently the quality of the current wavefront W_i is good. Thus, we can increase K and propagate *more* windows in the next iteration. To better understand this, one could consider the extreme case where $K = 1$ and $c_{small} = 0$: the wavefront is very smooth and such a K is obviously too conservative.
- Otherwise, the wavefront's quality is not good, so we should decrease K and propagate fewer windows in the $i + 1$-th iteration.

Inspired by these observations, we adaptively adjust K by setting $K_{i+1} = K_i + (c_{large} - c_{small})$. This simple strategy works remarkably well in practice. See Section 6 for detailed discussions.

4.3 Correctness & Complexity

The correctness of the FWP method relies on the following proposition.

Proposition 1. *Both the MMP and CH algorithms generate correct solutions regardless of the order in which the windows are processed in the queue.*

Proof. First, note that (1) the useless windows will be deleted when they are covered by other windows arrived later that provide shorter distances to the source and (2) for any windows that provide shortest distances to the source, their parent windows also provide shortest distances to the source. So all the windows in the correct solution will appear in the queue regardless of the window propagation order.

Second, we show that the algorithm will terminate in a finite number of steps regardless of the window propagation order. We assign an integer-valued level to each candidate window (including both useful and useless windows) in the queue. The windows facing the source are at level 1. When a level-i window is propagated, its child windows have level $i + 1$. Note that on an n-face mesh, a window's level cannot exceed n. Assume that an algorithm randomly picks a window w and propagates it. Since the level of w and all its descendants should be not larger than n, w has a finite number of descendants. Thus the total number of windows generated by this algorithm is finite. □

Note that Surazhsky et al. mentioned the above property (c.f. Section 3.4 [5]) but did not give a proof.

Our FWP method works well for real-world mesh models, as indicated in the following property with a moderate realistic assumption.

Proposition 2. *Assume that the degree of each vertex in M is bounded by a constant D. Both the FWP-MMP and FWP-CH algorithms produce $O(n^2)$ windows, and they have $O(n^2 \log n)$ and $O(n^2)$ time complexity, respectively, where n is the number of vertices in M.*

Proof. Upon the termination of the MMP or CH algorithm to the mesh M, we obtain a set of windows stored in each edge and vertex. We call these windows *final*, since they encode the shortest geodesic distance. In contrast, the windows, which were created during window propagation and deleted later, are useless windows and not final.

In our FWP-based algorithm, $K_i (\geq 1)$ windows of smallest labels are propagated at the i-th iteration. First note that the window of smallest label must be final since no other windows in the queue can replace it later. Then in the propagated K_i windows, they contain at least one *final* window. As long as a window becomes final, its status remains unchanged throughout the remaining iterations. Second, both the MMP and CH algorithms have no more than $O(n^2)$ final windows. Since at least one final window is extracted from the wavefront in each iteration, the FWP-based algorithm converges in $O(n^2)$ iterations at most.

Note also that K_i is bounded by a constant K_{max} (Line 8 in Algorithm 1). Therefore, at most mK_{max} child windows are generated and inserted into the buckets, where m is the maximum number of children a parent window can have. Thus, we have $m \leq D$ due to the assumption that the degree is no more than D. Finally, the total number of windows inserted into the buckets are $O(DK_{max}n^2) = O(n^2)$.

The FWP-MMP algorithm has an $O(\log n)$ overhead at each iteration, since there are, at most, $O(n)$ windows at each edge, and these windows are sorted in the same manner as the original MMP algorithm. Therefore, the FWP-MMP algorithm has an $O(n^2 \log n)$ time complexity. For the FWP-CH algorithm, the overhead per iteration is $O(1)$, resulting in an $O(n^2)$ time complexity. □

5 Experimental Results

Both the MMP and ICH algorithms as well as the FWP methods are undiscretized algorithms, that is, they can obtain the exact geodesics if numerical operations are exact. However, floating-point computation is often used in implementation of these algorithms due to its high efficiency. There are two sources of numerical error. First, floating point computation has truncated error, which is machine dependent and cannot be avoided in the algorithm. Second, propagating a small window is not cost-effective, since a small window covers only a narrow region on the mesh. In practice, both the MMP and ICH algorithms discard a window if its size is smaller than a user-specified threshold ε (e.g., [18]).

We observe that the truncated error with floating-point precision has little affect on these algorithms, given the robust techniques [18], [19], [20] for handling geometric degenerate cases. However, the threshold ε for determining tiny windows has a large impact on the performance. Take the 144K-face Bunny model (scaled to a unit cube) as an example. A strict threshold $\varepsilon = 10^{-6}$ results in $6.9M$ windows for the MMP algorithm and $6.2M$ windows for the ICH algorithm. A loose threshold $\varepsilon = 10^{-5}$, however, can reduce the windows by 26% and 10% for the MMP algorithm and the ICH algorithm, respectively. As the performance closely depends on window complexity, we adopt the same threshold $\varepsilon = 10^{-6}$ throughout our experiments to ensure a fair comparison between the two classes.

We thoroughly evaluated the performance of the original ICH and MMP algorithms, as well as our

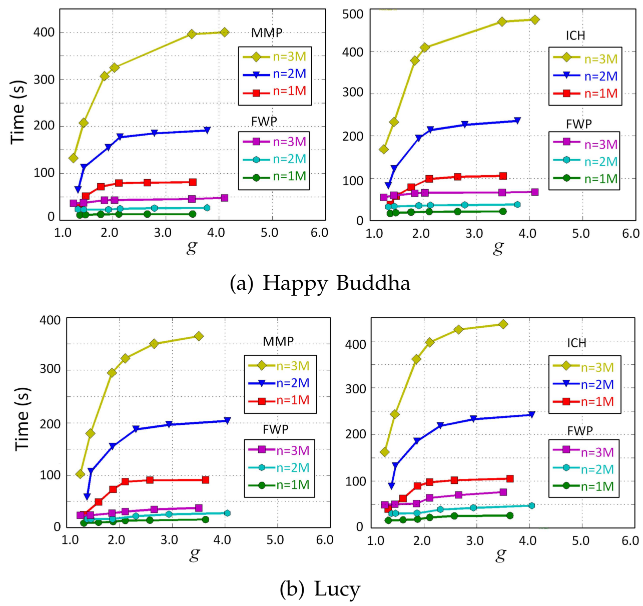


Fig. 6. By fixing the resolution, we create a sequence of meshes with various anisotropy using the method in [21]. Each curve corresponds to the timing of applying some algorithm to a sequence. We observe that both the MMP and ICH algorithms are highly sensitive to mesh triangulation, that is, increasing the mesh anisotropy greatly slows down their speeds. In contrast, our method is more robust to tessellation than the MMP and ICH algorithms.

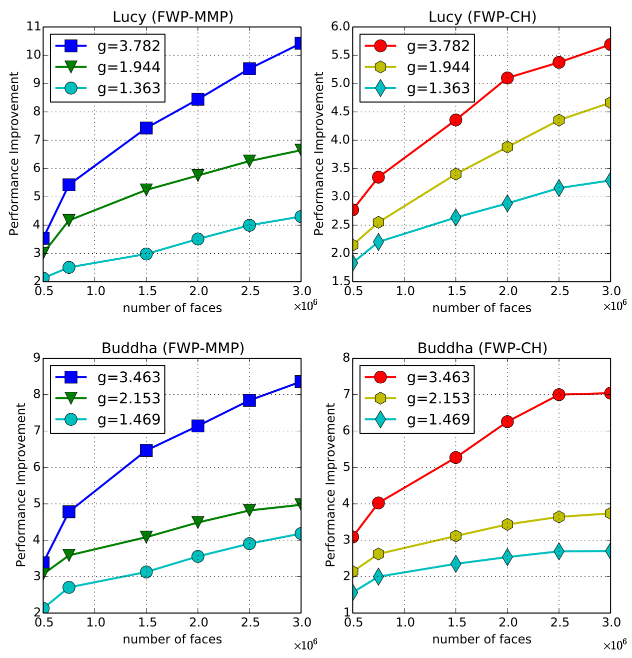


Fig. 7. The FWP technique can significantly speed up the MMP and CH algorithms. The vertical axis is the performance improvement (i.e., the ratio of the time of the original algorithm to the FWP algorithm) and the horizontal axis is the mesh resolution. We observe that the higher the resolution (measured by number of faces) and anisotropy (measured by g) of the mesh, the higher the speedup.

FWP-based improvements, on a wide range of models. Due to page limit, some test models are illustrated in Figure S3 in Supplemental Material. Timings were measured on a PC with an Intel Core i7-2600 CPU 3.40 GHz and 8GB memory. To obtain stable results, we randomly chose 100 points for each model and then reported the mean value. For the constant K_{max} (line 8 in Algorithm 1) that determines the upper bound of adaptive K in the FWP method, we empirically set $K_{max} = 20,000$ for various real-world models.

We observed the following characteristics via our experiments:

1) Robustness. The FWP method is less sensitive to mesh tessellation than the existing algorithms. This robust feature is due to the fact that our method propagates K windows per iteration, which can be considered as taking K samples simultaneously on the wavefront. With a large K, wavefront propagation is intrinsic to the geometry, therefore, the performance is not sensitive to mesh triangulation. We use $g(f) = \frac{P \cdot H}{2\sqrt{3}S}$ to measure the quality of triangle f, where P, H and S are respectively the half-perimeter, the longest edge length and area of f. Then we define $g = \frac{\sum_{f\in F} g(f)}{|F|}$ to measure the anisotropy of the input mesh M. An isotropic mesh has $g = 1$ and an anisotropic mesh has $g > 1$. In general, the larger the value of g, the higher the anisotropy the mesh has. As Figure 6 shows, by fixing the mesh's resolution, the speeds of the MMP and ICH algorithms become slower when the mesh becomes more anisotropic, whereas the speed of our FWP-based methods are very stable.

2) High performance. The FWP technique can significantly boost the speed of the MMP and ICH algorithms on all test models. As shown in Figure 7, the FWP-based algorithm is particularly favored for large-scale real-world models. The higher the resolution and anisotropy of the mesh, the better performance improvement it brings. For meshes with fairly regular triangulation, the FWP-CH algorithm can double the speed of the ICH algorithm, furthermore, the FWP-MMP algorithm is 3 times faster than the MMP algorithm. For large-scale models with anisotropic triangulations, the FWP-MMP algorithm can improve the performance of the MMP algorithm by an order of magnitude, and the FWP-CH algorithm is also 5 times faster than the ICH algorithm. Computational results show that the FWP-MMP algorithm is the most efficient *exact* discrete geodesic algorithm. See Supplementary Material for more results.

3) Unified framework. From a micro scale, the ICH and MMP algorithms adopt different window propagation schemes, producing different numbers of windows and requiring different numbers of iterations for convergence. Our FWP framework unifies the two classes of algorithms from the macro scale: As Ta-

Model	$\|F\|$	Ratio of iteration numbers	
		MMP/ICH	FWP-MMP/FWP-ICH
Fertility	60,000	0.836	$468/460 = 1.017$
Horse	96,965	0.714	$775/772 = 1.004$
Bunny	144,036	0.680	$865/911 = 0.950$
Golfball	245,760	0.509	$1478/1553 = 0.952$
Sphere	327,680	0.342	$1286/1348 = 0.954$
Armadillo	345,944	0.676	$1931/1998 = 0.966$
Lucy	525,814	0.685	$2286/2299 = 0.994$
Gargoyle	700,000	0.695	$2395/2446 = 0.979$
Blade	1,765,388	0.634	$4086/3844 = 1.063$
Dragon	4,000,000	0.788	$13748/12387 = 1.110$

TABLE 1
Mesh complexity (face number $|F|$) and the ratios of the number of iterations the algorithms need to converge for the pairs {MMP, ICH} and {FWP-MMP, FWP-ICH}.

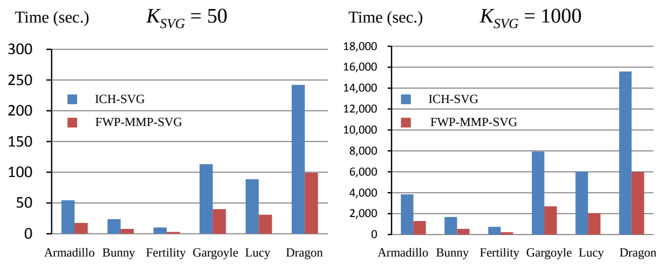
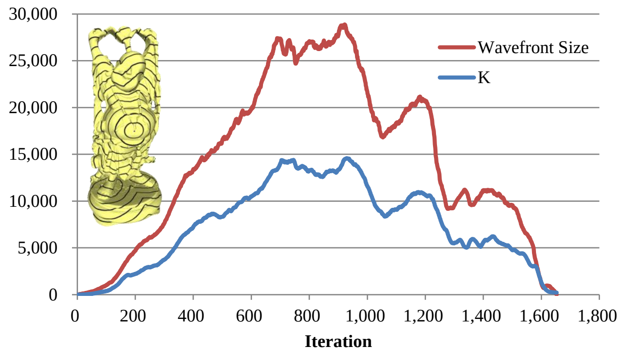


Fig. 8. The FWP method can improve SVG construction by a factor of 3 to 10. The parameter K_{SVG} controls the accuracy of the computed geodesic distance. The higher K_{SVG}, the smaller the error, and the longer the time for constructing the SVG. Timing was measured on a single CPU core.



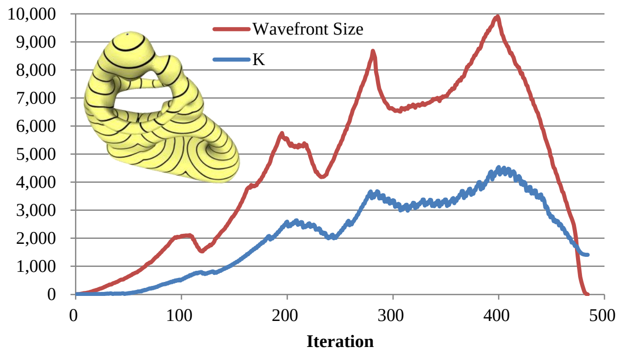


Fig. 9. K, the number of smallest-label windows used in each iteration, is adapted to wavefront size. The ICH algorithm has a similar performance as the MMP algorithm.

ble 1 shows, the FWP-CH and FWP-MMP algorithms propagate the wavefronts at a similar pace and take roughly the same number of iterations to converge. We believe that other window-based algorithms (if any) can also fit into the FWP framework.

4) Improving the SVG technique. Saddle vertex graph [9] is a sparse undirected graph that encodes the geodesic information of a give mesh. With SVG, the geodesic distance can be computed efficiently by Dijkstra's shortest path algorithm. However, constructing SVG is expensive, since one has to compute all direct geodesic paths[2]. In [9], the geodesic paths were computed using the ICH algorithm, which is time consuming. For example, computing the exact SVG for the 144K-face Bunny model takes half an hour on a single CPU core. Although SVG construction can be significantly improved by parallel computing, the GPU implementation is non-trivial. In this paper, we show that our CPU-based FWP method can be easily adapted to compute SVG. Experimental results show that our method can shorten the pre-computation time by a factor of 3 to 10. See Figure 8.

2. A geodesic path is direct if it does not pass through any saddle vertices.

6 Discussions

In this section we discuss three features as follows that make the FWP method significantly improve the speed of MMP/CH/ICH algorithms.

1) Adaptive K. The geometry of a wavefront is dependent on the model, the source point, and the time, all of which may vary significantly during the iterative procedure. So the number of windows propagated in each iteration should be adapted to the wavefront size in order to perform well. Figure 9 shows the relationship between K and the wavefront size for the Happy Buddha and Fertility models. We observe that the K-curve's shape is similar to the wavefront size curve. As Figure 10 shows, the adaptive K strategy outperforms the fixed K strategy significantly.

2) Wavefront quality. Unlike the MMP/ICH algorithm that propagates a single window per iteration, the FWP method processes a large number of windows simultaneously at each iteration. Thanks to our adaptive K strategy, the FWP method can both move the wavefronts efficiently and ensure they are of high quality. Figure 11 shows an example.

3) Bucket size. The selection of bucket size in the FWP method is heuristic. Figure 12 shows how the performance changes with the bucket size, from which we observe that the FWP method is fairly stable and reaches the highest performance when the bucket size falls into $1\times$ to $2 \times h/\sqrt{n}$. On the other side, the

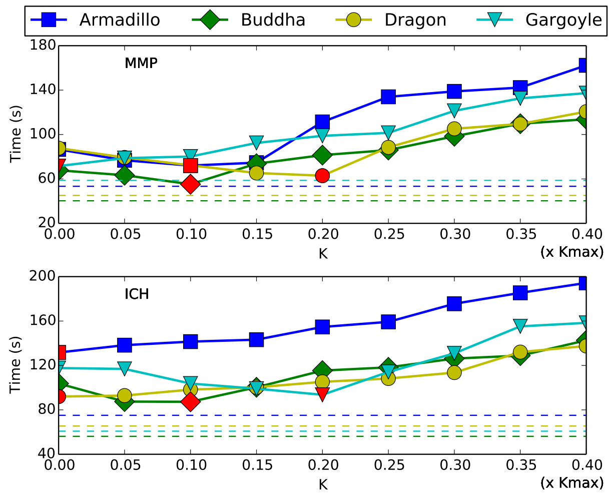
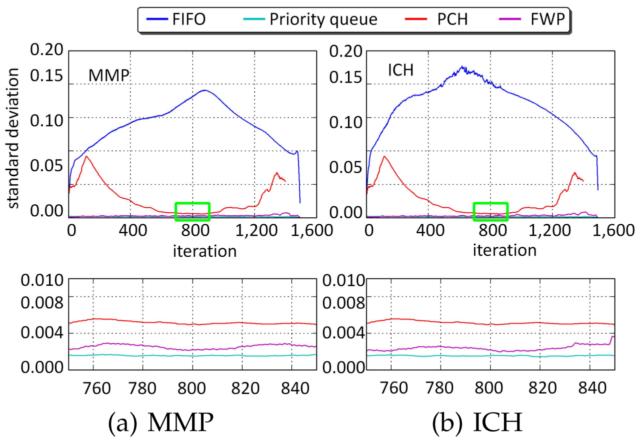


Fig. 10. Fixing K has worse performance than the dynamic K strategy. We measure the performance of the FWP method with various fixed K values, ranging from 1% to 40% of K_{max}. Each marker corresponds to the timing with a fixed K value and the optimal timing is highlighted in red. Clearly, the K values leading to the optimal performance vary significantly among the test models. The dashed lines below are the timing of the FWP method with adaptive K strategy.



(a) MMP (b) ICH

Fig. 11. Wavefront quality on the Dragon model. The priority queue based MMP and ICH algorithms produce high quality wavefronts, however, the wavefronts are moving slowly due to the expensive overhead. The FIFO queue based MMP and ICH algorithms have constant time overhead at each iteration, however, their performance is very poor due to the low quality wavefronts. Our FWP-MMP and FWP-CH algorithm are able to propagate the wavefronts much faster without compromising their quality. The horizontal axis is the normalized timing and the vertical axis is the standard deviation of the wavefront. The smaller the standard deviation, the smoother the wavefront, thus, the higher the quality it has. The close-up views show that the wavefronts of our method have similar quality to the wavefronts of the MMP and ICH algorithms. The PCH algorithm is also macro, however, its wavefront quality is worse than ours.

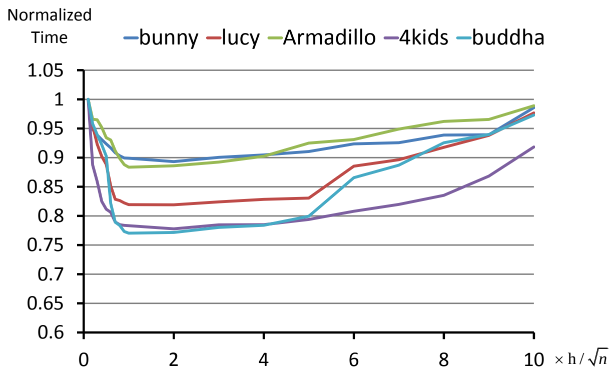


Fig. 12. Performance changes with the bucket size. The x-axis shows different bucket sizes (by a multiple of $h/\sqrt{n}$) and the y-axis gives the normalized time (divided by the longest running time of each model).

curves shown in Figure 12 are U-shaped that means both too small and too large bucket size will lead to relatively bad performance.

7 Comparison

We compares the FWP method with other representative geodesic algorithms, including MMP [2], ICH [7], PCH [8], FMM [13], [14], [22], the heat method [16] and the label correcting (LC) method [23]. We classify these algorithms in two groups, i.e., macro and micro. A micro algorithm processes a single element (e.g., a window in MMP/ICH or a triangle in FMM) per iteration, whereas a macro algorithm processes multiple elements simultaneously. Table 2 summarizes the features in these algorithms.

7.1 FWP vs. the Fast Marching Method

The Fast Marching Method (FMM) [13] is a popular technique for solving the boundary value problem of the Eikonal equation, $\nabla T = F(x, y)$, where $F > 0$ is the front moving speed and T is the travel time. Solving the Eikonal equation with $F \equiv 1$ and $T(s) = 0$ produces the polyhedral distance field at the source s. The FMM adopts a Dijkstra sweep and uses the fact that information only flows outward from the seeding area. The FMM is flexible and can be applied to both regular grids [13] and triangle meshes [14] with an $O(n \log n)$ time complexity, where the factor $\log n$ is due to the administration of a priority queue.

To improve the speed of the FMM on *regular grids*, Yatziv et al. [22] suggested using a so-called *untidy* priority queue within the FMM. Their novel idea is to use the bucket sort technique together with a quantization that does not distinguish between the values of T within a small range. Therefore, each entry of the priority queue may contain multiple elements. Yatziv et al. showed that the number of elements in each entry is $O(1)$, and finding the element with

Method		Domain	Data Structure	Overhead	Space Complexity	Time Complexity	Type
CH	CH [3]	triangle meshes	FIFO queue	$O(1)$	$O(n^2)$	$O(n^2)$	micro
	ICH [7]	triangle meshes	priority queue	$O(\log n)$	$O(n^2)$	$O(n^2 \log n)$	micro
	PCH [8]	triangle meshes	FIFO queue	$O(1)$	unbounded	unbounded	macro
	LC-CH	triangle meshes	FIFO queue	$O(1)$	$\Omega(n^3)$	$\Omega(n^3)$	micro
	FWP-CH	triangle meshes	bucket & FIFO queue	$O(1)$	$O(n^2)$	$O(n^2)$	macro
MMP	MMP [2]	triangle meshes	priority queue	$O(\log n)$	$O(n^2)$	$O(n^2 \log n)$	micro
	LC-MMP	triangle meshes	FIFO queue	$O(1)$	$\Omega(n^3)$	$\Omega(n^3)$	micro
	FWP-MMP	triangle meshes	bucket & FIFO queue	$O(\log n)$	$O(n^2)$	$O(n^2 \log n)$	macro
FMM	FMM [14]	regular grids	priority queue	$O(\log n)$	$O(n)$	$O(n \log n)$	micro
		triangle meshes	priority queue	$O(\log n)$	$O(n)$	$O(n \log n)$	micro
	Yatziv's FMM [22]	regular grids	untidy priority queue	$O(1)$	$O(n)$	$O(n)$	micro
		triangle meshes	untidy priority queue	$O(n \log n)$	$O(n)$	$O(n^2 \log n)$	micro

TABLE 2
Comparison of Dijkstra-like geodesic algorithms.

the minimal label also takes $O(1)$ time. As a result, their algorithm has a linear run-time on regular grids. Although an error is introduced, it is of the same order as the local truncation error of the discretization, since the change of T values in a Dijkstra iteration is bounded on regular grids.

It is worth noting that the linear $O(n)$ time complexity of Yatziv et al's method does not hold on triangle meshes, since it is not possible to bound the change of T values on triangle meshes, whose triangulation may be arbitrary. In fact, on triangle meshes, the range of the change of T values depends on the triangulation, instead of the number of vertices n. Consider a very skinny triangle t whose longest side is of the same order of the model's diagonal. Then the size of an entry (i.e., quantization step size) in the untidy priority queue must be big enough to contain the longest side of t, meaning that this entry contains $O(n)$ triangles. Since finding the element with the minimal label takes $O(n \log n)$ time, the time complexity becomes $O(n^2 \log n)$, which is much worse than the original FMM.

Our experimental results also confirm that the performance of Yatziv et al's FMM is highly dependent on the mesh triangulation. Moreover, the accuracy of Yatziv et al's method drops significantly on anisotropic meshes. See Figure 13(b). In terms of data structure, note that the entry size in Yatziv et al's FMM is triangulation dependent, while the bucket size in the FWP method is fixed. Furthermore, the overhead per window operation in Yatziv et al's FMM is $O(n \log n)$ and in our method is $O(1)$.

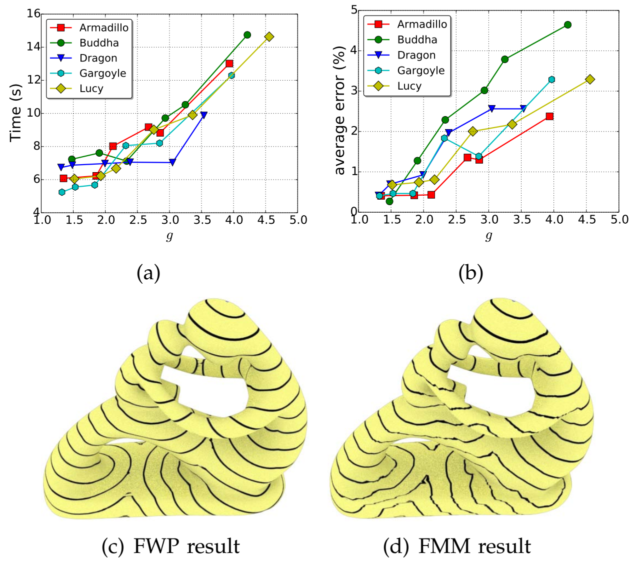
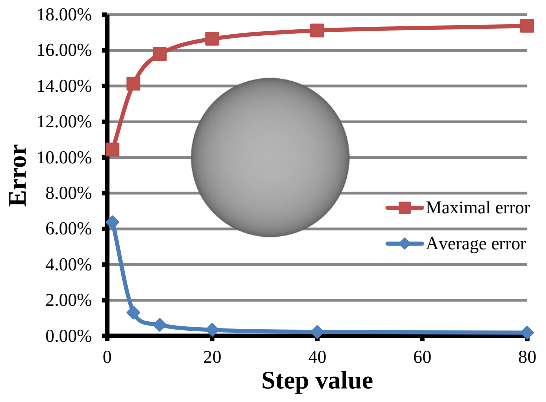


Fig. 13. (a) Yatziv et al.'s FMM, as a micro algorithm, its performance is sensitive to mesh triangulation. (b) Its average numerical error increases significantly on anisotropic meshes. (c) shows the FWP result and and (d) shows the FMM result with an average error 5.53%.

7.2 FWP vs. the Heat Method

The heat method [16] computes the geodesic distance by solving a Poisson equation from the normalized gradient of the heat flow. Based on standard numerical packages, it is easy to implement and highly flexible to support a wide range of geometric domains, including grids, triangle meshes and point clouds. Since the Laplacian matrix can be pre-factored, solving the Poisson equation takes only near-linear time. Below we compare the heat method and the FWP method in terms of accuracy and performance.

Accuracy. The accuracy of the heat method closely depends on the heat diffusion time t. Theoretically, the time approaching zero leads to the accurate solution of geodesic distance. However, a tiny time t results in serious numerical issue. Crane et al. [16] observed that the error-t plot is U-shaped and suggested $t = h^2$ for common models, where h is the average edge length. Similar to the fast marching method, the heat method computes a first-order approximation of geodesic distance. Thus, its results closely depend on the triangulation quality. Although the suggested parameter $t = h^2$ works fairly well on well-tessellated meshes, we observe that it leads to large error on anisotropic meshes. Figure 14 shows the results on the unit sphere with 200 longitude circles and 500 latitude circles, which has 199,600 triangles and an anisotropy



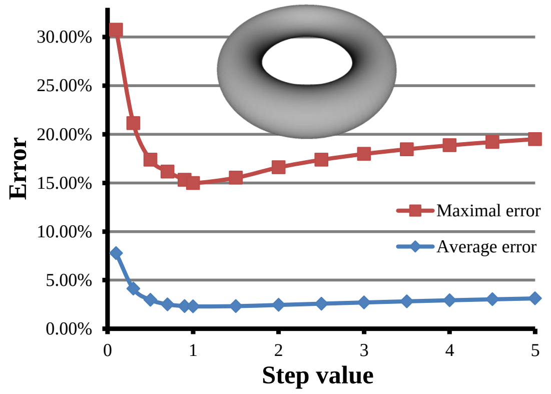


Fig. 14. The heat method controls the accuracy by the diffusion time $t = \lambda h^2$, where λ is the step value and h is average edge length. Given the $199,600$-face unit sphere model with the anisotropy measure $g = 2.28$, the default setting $\lambda = 1$ produces a mean relative error 6.3%. Increasing the step value λ is helpful to reduce the mean error. For example, the mean error drops to 0.18% when $\lambda = 80$, however, the maximum relative error increases significantly. Given the $20,000$-face torus model with $g = 2.36$, the default setting $\lambda = 1$ produces the best mean relative error 2.3% and the best maximum relative error 14.97%. Increasing or decreasing the step value λ cannot help to reduce the errors.

measure $g = 2.28$. Setting $t = h^2$ produces a mean relative error 6.36%, comparing to the ground truth geodesic distance computed by closed-form formula. Although taking a longer diffusion time smoothes the distances and reduces the mean error, the maximal error increases accordingly. Our method, in contrast, computes an accurate result with mean relative error 0.00063% and maximal error 0.0044%. Figure 14 also shows the results on a $20,000$-face torus model with $g = 2.36$. The heat method obtains the best mean relative error 2.3% and the best maximum relative error 14.97% at $\lambda = 1$, while our method computes a more accurate result with mean relative error 0.00976% and maximal error 0.1061%. In fact, as observed in [16] and [9], the discrete geodesic distances computed by the MMP and CH algorithms converge to the smooth geodesic distances at a quadratic speed, while the FMM and the heat method have only linear convergence rate.

Performance. As a pre-computation method, the heat method has two steps: pre-computation (factoring the Laplacian matrix) and solving (recovering the distances by solving a Poisson equation). Our FWP method is a direct approach, which is much slower than the heat method. However, as mentioned in Section 5, the FWP method complements the SVG method, since it reduces its pre-computation time. It is noted that the SVG method has similar solving performance as the heat method and its accuracy is much higher. See Table 3.

7.3 FWP vs. the Parallel CH Algorithm

The PCH algorithm [8] is a GPU-based parallel improvement of the classic CH algorithm. Both the PCH and FWP algorithms propagate a large number of windows at each iteration and thus both are macro. However, they differ in two ways. First, the number K of windows to be propagated in each iteration in FWP is time-dependent and adaptive to the wavefronts. As Figure 11 shows, the wavefront quality of the FWP method is better than the PCH algorithm. Second, although the PCH algorithm performs well with real-world models, it is not possible to bind its space as well as time complexities, which could be theoretically exponential. Our method, however, is theoretically sound and produces at most $O(n^2)$ windows, leading to provable time complexity of the FWP-MMP and FWP-CH algorithms. Computational results show that our CPU-based FWP-MMP algorithm has a speed comparable to PCH [8] (Table 3).

7.4 FWP vs. the Label Correcting Method

The MMP and ICH algorithms propagate windows in a continuous-Dijkstra fashion. Dijkstra's algorithm takes the node with the smallest label in a candidate list C and is known as a *label setting* method, since the node removed from the list is permanently labeled and never enters the list again. A *label correcting* (LC) method maintains a queue for the candidate list C so that the selection of the to-be-processed node takes only $O(1)$ time, at the expense of multiple entrances of nodes in C. Among many label correcting schemes, Bertsekas [23] observed that the SLF-LLL-THR algorithm performs extremely well in practice, significantly outperforming the original Dijkstra's algorithm on real-world sparse graphs.

We implemented the SLF-LLL-THR strategy in the MMP and ICH algorithms and we refer to the LC based methods as the LC-MMP and LC-CH algorithms. We compare the FWP-based algorithms with the LC-based algorithms and we observe that the FWP method runs consistently faster than the LC methods on all test models, the higher the mesh resolution and anisotropy, the better the performance improvement of the FWP method to the LC method.

Model	$\|F\|$	Heat method			SVG ($K = 50$)				ICH	FWP-MMP	PCH
					Pre-computation		Solving	Mean Error			
		Pre-computation	Solving	Mean Error	FWP-MMP	ICH					
Armadillo	346K	1.75s	0.11s	0.90%	17.52s	54.4s	0.08s	0.14%	9.39s	3.43s	1.39s
Bunny	144K	0.72s	0.04s	0.83%	7.82s	23.63s	0.02s	0.12%	5.43s	2.0s	1.15s
Fertility	60K	0.22s	0.01s	1.81%	2.95s	10.14s	0.01s	0.14%	1.74s	0.79s	0.41s
Horse	97K	0.35s	0.04s	0.72%	7.32s	15.91s	0.02s	0.12%	3.41s	1.38s	0.88s
Sphere	327K	4.92s	0.15s	0.19%	18.85s	51.29s	0.06s	0.10%	75.02s	16.56s	11.02s

TABLE 3
Performance statistics on meshes with fairly good tessellation. The PCH algorithm was tested on an Nvidia GTX580 card with 512 CUDA cores and all the other methods were implemented in single threaded C++ and tested on a PC with an Intel Core i7-2600 CPU (3.40 GHz). Both the heat method and the SVG method compute the approximate geodesic distances, whereas the others are exact. For the heat method, we set the the diffusion time $t = h^2$, where h is the average edge length. We construct a small-scale saddle vertex graph with $K = 50$ and report the mean relative error. The FWP-MMP based SVG construction is 2 to 3 times faster than the ICH method.

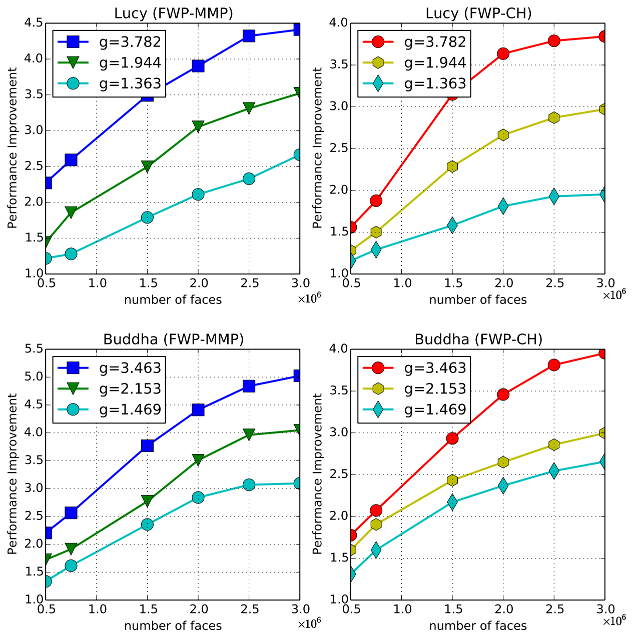


Fig. 15. The FWP method is consistently faster than the LC-based method. The vertical axes show the ratio of the timing of the LC method to our method.

See Figures 15. Moreover, as a macro algorithm, the FWP method is more robust to mesh triangulation (using an anisotropic measure g) than the *micro* LC algorithms.

Theoretically, the LC-based Dijkstra's algorithm [23] does not have a time bound due to the heuristics used for determining the threshold. For the LC-based discrete geodesic algorithms, in Supplementary Material we construct a triangulation pattern so that the LC method produces at least $\Omega(n^3)$ windows, leading to a $\Omega(n^3)$ time complexity. In sharp contrast, our bucket data structure and adaptive K strategy guarantees an $O(n^2)$ window complexity accompanied with an $O(n^2 \log n)$ time complexity for the FWP-MMP algorithm and an $O(n^2)$ time complexity for the FWP-CH algorithm.

8 Conclusion

This paper presented a fast wavefront propagation (FWP) framework, which has the following advantages. First, as a macro algorithm, the FWP method is less sensitive to mesh tessellation than the existing micro (MMP/CH/ICH) algorithms. It is also a generic computational framework that enables the MMP and ICH algorithms to propagate the wavefronts in a similar fashion. Second, the FWP method is theoretically sound and has provable time and space complexity. The FWP-CH and FWP-MMP algorithms have $O(n^2)$ and $O(n^2 \log n)$ time complexity, respectively. It is worth noting that the FWP-CH algorithm is the first window-oriented algorithm that reaches the $O(n^2)$ lower bound while performing well in practice. Third, computational results show that the FWP-MMP algorithm can improve the speed of the MMP algorithm by a factor of 10 and the FWP-CH algorithm is also 5 times faster than the ICH algorithm. Through extensive evaluation presented in Supplemental Material, we confirm that the FWP-MMP algorithm is the most efficient serial and *exact* algorithm for computing geodesic distances on triangle meshes. In the future, we will investigate the GPU-based parallelization of the FWP framework.

Acknowledgments

We thank Keenan Crane for his code of the heat method. The models are courtesy of Aim@Shape and Stanford University. This work was supported by the Natural Science Foundation of China (61322206, 61432003, 61272228), the National Basic Research Program of China (2011CB302202), TNList Cross-discipline Foundation and Beijing Higher Institution Engineering Research Center of Visual Media Intelligent Processing and Security, and Singapore Ministry of Education Grants (MOE2013-T2-2-011 and RG40/12).

REFERENCES

[1] P. Bose, A. Maheshwari, C. Shu, and S. Wuhrer, "A survey of geodesic paths on 3D surfaces," *Computational Geometry*, vol. 44, no. 9, pp. 486–498, 2011.

[2] J. S. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," *SIAM Journal on Computing*, vol. 16, no. 4, pp. 647–668, 1987.

[3] J. Chen and Y. Han, "Shortest paths on a polyhedron," in *Proceedings of the Sixth Annual Symposium on Computational Geometry*. ACM, 1990, pp. 360–369.

[4] Y.-J. Liu, D. Fan, C. Xu, and Y. He, "On discrete geodesics: tight bound and triangle anisotropy," *Submitted for publication*, 2015.

[5] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe, "Fast exact and approximate geodesics on meshes," in *ACM Transactions on Graphics*, vol. 24, no. 3. ACM, 2005, pp. 553–560.

[6] Y.-J. Liu, "Exact geodesic metric in 2-manifold triangle meshes using edge-based data structures," *Computer-Aided Design*, vol. 45, no. 3, pp. 695–704, 2013.

[7] S.-Q. Xin and G.-J. Wang, "Improving Chen and Han's algorithm on the discrete geodesic problem," *ACM Transactions on Graphics*, vol. 28, no. 4, p. 104, 2009.

[8] X. Ying, S.-Q. Xin, and Y. He, "Parallel Chen-Han (PCH) algorithm for discrete geodesics," *ACM Transactions on Graphics*, vol. 33, no. 1, pp. 9:1–9:11, 2014.

[9] X. Ying, X. Wang, and Y. He, "Saddle vertex graph (SVG): a novel solution to the discrete geodesic problem," *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 170:1–170:12, 2013.

[10] S.-Q. Xin, X. Ying, and Y. He, "Constant-time all-pairs geodesic distance query on triangle meshes," in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2012, pp. 31–38.

[11] ——, "Efficiently computing geodesic offsets on triangle meshes by the extended xin-wang algorithm," *Computer-Aided Design*, vol. 43, no. 11, pp. 1468–1476, 2011.

[12] S.-Q. Xin, Y. He, and C.-W. Fu, "Efficiently computing exact geodesic loops within finite steps," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 6, pp. 879–889, 2012.

[13] J. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of National Academy of Sciences*, vol. 93, pp. 1591–1595, 1996.

[14] R. Kimmel and J. Sethian, "Computing geodesic paths on manifolds," *Proceedings of National Academy of Sciences*, vol. 95, pp. 8431–8435, 1998.

[15] S.-Q. Xin, D. T. Quynh, X. Ying, and Y. He, "A global algorithm to compute defect-tolerant geodesic distance," in *SIGGRAPH Asia 2012 Technical Briefs*, 2012, p. 23.

[16] K. Crane, C. Weischedel, and M. Wardetzky, "Geodesics in heat: A new approach to computing distance based on heat flow," *ACM Transactions on Graphics*, vol. 32, no. 5, p. 152, 2013.

[17] K. Hildebrandt, K. Polthier, and M. Wardetzky, "On the convergence of metric and geometric properties of polyhedral surfaces," *Geometriae Dedicata*, vol. 123, no. 1, pp. 89–112, 2006.

[18] Y.-J. Liu, Q.-Y. Zhou, and S.-M. Hu, "Handling degenerate cases in exact geodesic computation on triangle meshes," *The Visual Computer*, vol. 23, no. 9-11, pp. 661–668, 2007.

[19] H. Edelsbrunner and E. P. Mücke, "Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms," *ACM Trans. Graph.*, vol. 9, no. 1, pp. 66–104, Jan. 1990.

[20] C. K. Yap, "A geometric consistency theorem for a symbolic perturbation scheme," in *Proceedings of the Fourth Annual Symposium on Computational Geometry*, ser. SCG '88, 1988, pp. 134–142.

[21] Z. Zhong, X. Guo, W. Wang, B. Lévy, F. Sun, Y. Liu, and W. Mao, "Particle-based anisotropic surface meshing," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 99:1–99:14, 2013.

[22] L. Yatziv, A. Bartesaghi, and G. Sapiro, "$O(N)$ implementation of the fast marching algorithm," *Journal of Computational Physics*, vol. 212, no. 2, pp. 393–399, 2006.

[23] D. P. Bertsekas, *Network Optimization: Continuous and Discrete models*. Athena Scientific Belmont, 1998.