Visual Comput (2007) 23: 661–668
DOI 10.1007/s00371-007-0136-5

ORIGINAL ARTICLE

Yong-Jin Liu
Qian-Yi Zhou
Shi-Min Hu

Handling degenerate cases in exact geodesic computation on triangle meshes

Published online: 9 June 2007
© Springer-Verlag 2007

Y.-J. Liu (✉) · Q.-Y. Zhou · S.-M. Hu
Department of Computer Science and Technology, Tsinghua University, P.R. China
liuyongjin@tsinghua.edu.cn,
zqy@mails.tsinghua.edu.cn,
shimin@tsinghua.edu.cn

Abstract The computation of exact geodesics on triangle meshes is a widely used operation in computer-aided design and computer graphics. Practical algorithms for computing such exact geodesics have been recently proposed by Surazhsky et al. [5]. By applying these geometric algorithms to real-world data, degenerate cases frequently appear. In this paper we classify and enumerate all the degenerate cases in a systematic way. Based on the classification, we present solutions to handle all the degenerate cases consistently and correctly. The common users may find the present techniques useful when they implement a robust code of computing exact geodesic paths on meshes.

Keywords Exact geodesic computation · Degenerate cases · Robustness

1 Introduction

An exact geodesic between two points in a 2-manifold mesh is a union of line segments within the mesh, which connects the two points and is locally length-minimized. The computation of exact geodesic paths on triangle meshes is a widely used operation in computer-aided design and computer graphics.

In [5], a practical implementation of the DGP algorithm in [3] is proposed for computing exact geodesics from a source point to one or all other points efficiently. In the worst case the DGP algorithm has complexities of $O(n^2)$ space and $O(n^2 \log n)$ time, while in practice the algorithm is observed to run in subquadratic time.

The implementation in [5] can be regarded as a *generic algorithm*, i.e., it is guaranteed to be correct with a generic situation, but how to handle degenerate cases is not reported. In this paper we enumerate all the degenerate cases risen from implementation in [5] and show that in most cases with arbitrarily shaped triangles, the degenerate cases frequently appears. An example is illustrated in Fig. 1. The mesh used in Fig. 1 has 2000 faces, 6000 edges and 1028 vertices. The triangles in the mesh are arbitrarily shaped, including both obtuse and acute triangles. Given a prescribed source point, there are totally 8807 cases handled, in which 2583 cases are degenerate, about 29.33%. Some degenerate cases are illustrated later in Sect. 3.1.

In geometric computation, degenerate cases will increase the instability of the generic algorithm. Theoretically, degenerate cases can be handled by using the symbolic perturbation scheme [1]. Though it is a powerful tool, this scheme may not be applicable in the computation of exact geodesic paths. First, symbolic perturbation requires exact arithmetic, with which many users are not familiar. Second, using symbolic perturbation does not solve the degenerate case itself, but an arbitrarily chosen nearby general case. Topology-oriented implementation is another way to handle degenerate cases [4]. However, it only guarantees to output a topology-consistent solution, which may not be the desired topology-correct one.

In this paper, to develop a robust and fast exact geodesic algorithm, we present a systematic solution to efficiently handle all the degenerate cases with floating point computation [6]. By doing so, geometric predicates are treated consistently and thus the implemented algorithm is robust.

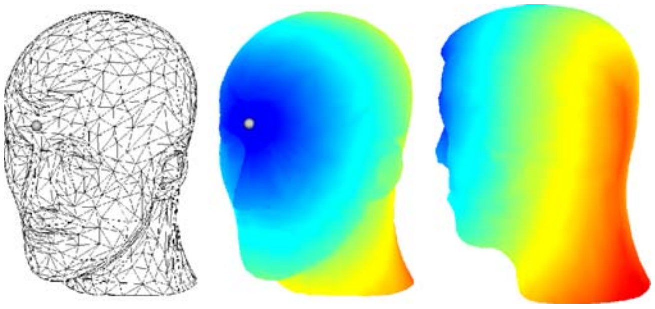


Fig. 1. Geodesic computation with a prescribed source point; points on the mesh are *colored* according to the geodesic distance to the source point

2 Review of the exact geodesic algorithm

We follow the notation in [5] to quickly review the DGP algorithm [3]. Shortest paths on mesh are rays emanating from the source vertex along tangent directions. Inside a triangle, a shortest path must be a straight line. When crossing an edge, a shortest path must be a straight line when the previous face is unfolded into the plane containing the next face. The only vertices (*geodesic vertices* below) that a shortest path can pass through are either boundary vertices or the vertices whose total surrounding angle is larger or equal to 2π. The basic idea of the DSP algorithm is to partition each mesh edge into a set of intervals (see Fig. 2). Each interval is encoded by a 6-tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$. The terms b_0, b_1 are parameters measuring distance along the edge. The unfolded position s of the geodesic vertex is encoded by its distances d_0, d_1 to the interval endpoints. A binary direction τ is used to specify the side of edge on which the source lies. The term σ is the length of the path from s back to the source v_s.

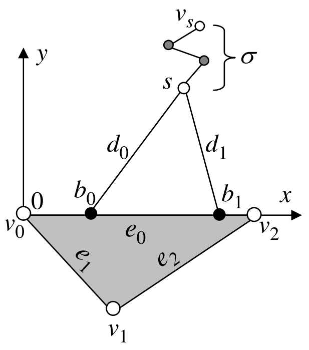


Fig. 2. A 6-tuple representation $(b_0, b_1, d_0, d_1, \sigma, \tau)$ of the interval

Given an interval I on an edge e_0, its distance field is propagated across an adjacent face to define new potential intervals on the two opposing edges e_1, e_2 (see Fig. 3). Three general cases exist for interval propagation. According to different cases, different new intervals are formed on the opposing edges. If intervals already exist on the opposing edges, the new interval may intersect some old ones. If two intervals intersect with a nonempty region δ, a quadratic equation

$$Ap^2 + Bp + C = 0 \qquad (1)$$

is solved to determine a new position $p \in \delta$ such that the updated ranges of the two intervals I and I' are (b_0, p) and (p, b'_1), respectively.

Starting from the source point, the DSP algorithm propagates distance information in a continuous Dijkstra-like fashion. When new intervals are created, they are placed in a priority queue sorted by minimum distance back to the source. When an interval is popped from the queue, interval propagation is performed in one of the three cases shown in Fig. 3. The reader is referred to [5] for a full description of this algorithm.

3 Degenerate cases

In the exact geodesic algorithm [5], two types of degeneracies occur in interval propagation:

1. *Degeneracies on geometric intersection*. These degeneracies arise from the determination of intersection region between the wedge and the line segments e_1 and e_2 (see Figs. 3 and 4).
2. *Degeneracies on geodesic discontinuities*. Due to the numerical errors in floating point computations, the solution of Eq. 1 often generates small gaps or overlaps between the newly resulted intervals; this gives rise to geodesic discontinuities along the intervals on the edge.

3.1 Degeneracies on geometric intersection

Basically, there are five degenerate cases in this class, as shown in Fig. 4:

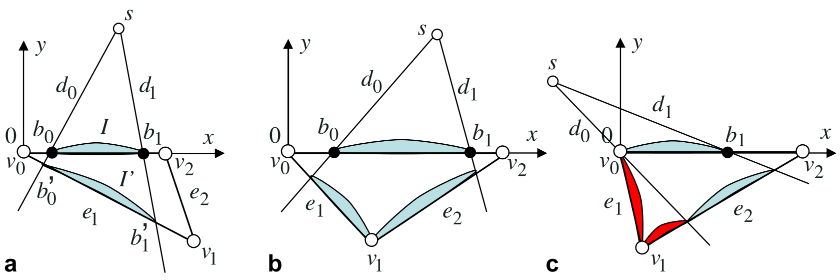


Fig. 3a–c. Interval propagation. **a** One new interval created. **b** Two new intervals created. **c** One new normal interval and two additional intervals (in *red*) created

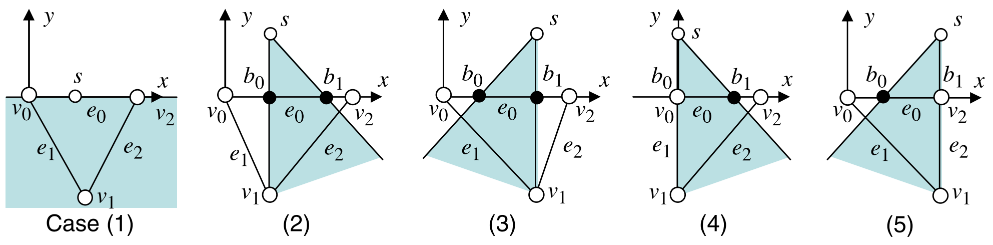


Fig. 4. Degenerate cases on geometric intersection; the *shaded area* indicates the wedge range of $b_0 \to s \to b_1$

1. The position of s lies on edge e_0. This case can happen if the interval is created on e_1 in the case of Fig. 3c.
2. Three points s, b_0, v_1 are in a straight line. This makes the new interval on the edge e_1 disappear in the case of Fig. 3b.
3. Three points s, b_1, v_1 are in a straight line. This is a symmetric case of Case 2.
4. Four points s, v_0, b_0, v_1 are in a straight line. This also means that points v_0 and b_0 coincide. In this case, the new interval on e_1 in the case of Fig. 3b must be treated as the new interval on e_1 in the case of Fig. 3c.
5. Four points s, v_1, b_1, v_1 are in a straight line. This is a symmetric case of Case 4.

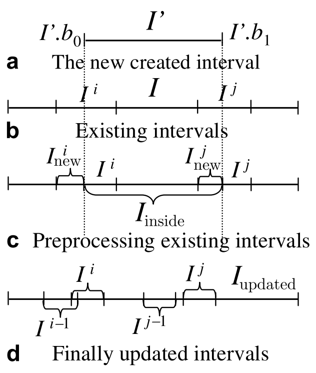
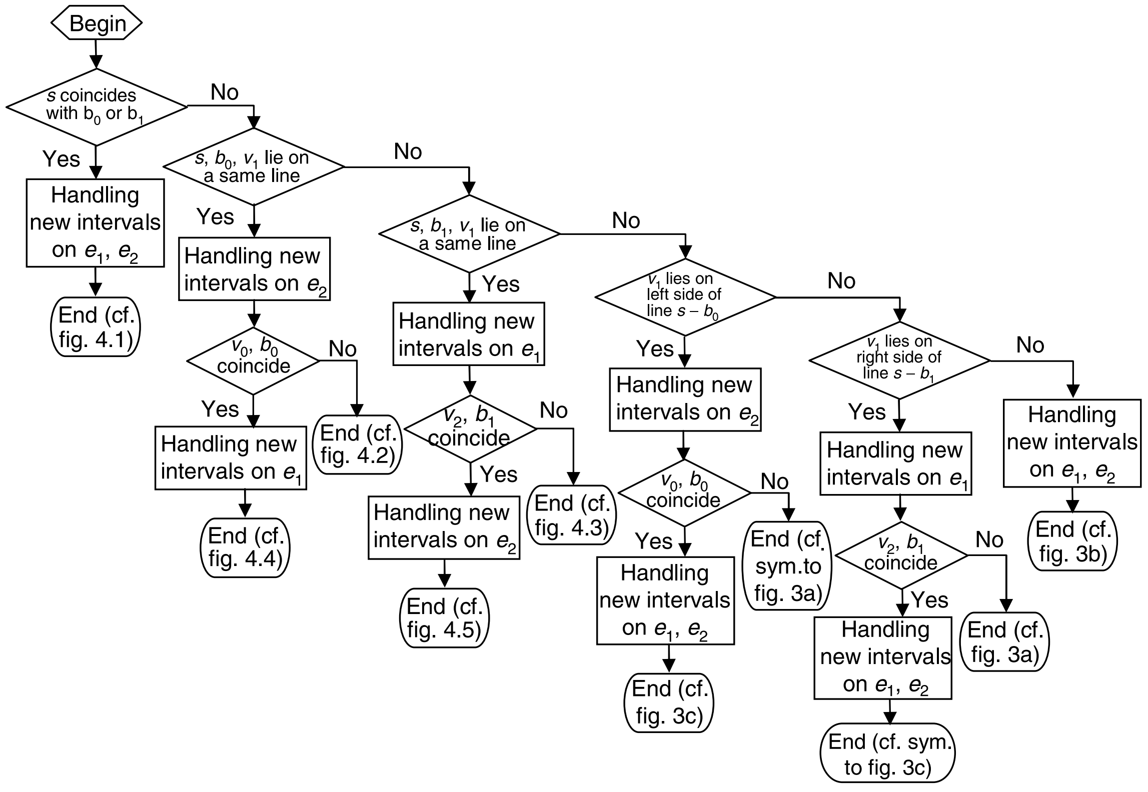


Fig. 5. Degeneracies on geodesic discontinuities

Notice that there are some degenerate cases composed of several basic cases. For example, referring to Fig. 4, if three points s, b_0, v_0 coincide, the basic degenerate cases 1, 2, and 4 occur simultaneously. Different degenerate cases must involve different procedures to process. Treating degenerate cases in random order will result in catastrophic failures in the algorithm. In Sect. 4.1, we



Begin
s coincides with b_0 or b_1
Yes
No
Handling new intervals on e_1, e_2
End (cf. fig. 4.1)
s, b_0, v_1 lie on a same line
Handling new intervals on e_2
v_0, b_0 coincide
Handling new intervals on e_1
End (cf. fig. 4.4)
End (cf. fig. 4.2)
s, b_1, v_1 lie on a same line
Handling new intervals on e_1
v_2, b_1 coincide
Handling new intervals on e_2
End (cf. fig. 4.5)
End (cf. fig. 4.3)
v_1 lies on left side of line $s - b_0$
Handling new intervals on e_2
v_0, b_0 coincide
Handling new intervals on e_1, e_2
End (cf. fig. 3c)
End (cf. sym.to fig. 3a)
v_1 lies on right side of line $s - b_1$
Handling new intervals on e_1
v_2, b_1 coincide
Handling new intervals on e_1, e_2
End (cf. sym. to fig. 3c)
End (cf. fig. 3a)
Handling new intervals on e_1, e_2
End (cf. fig. 3b)

Fig. 6. The flowchart of the decision system to handling degeneracies on geometric intersection

present a concise decision procedure to properly handle all the degenerate cases.

3.2 Degeneracies on geodesic discontinuities

After the determination of intersection region between wedge $b_0 \to s \to b_1$ and edges e_1, e_2, new intervals are created (see Fig. 5). Suppose that a new interval I' with range $(I'.b_0, I'.b_1)$ is created on edge e on which there already exists a set of intervals $I = \{I^0, I^1, \cdots\}$ sorted by positions on edge, $I^{i-1}.b_1 \le I^i.b_0 < I^i.b_1 \le I^{i+1}.b_0$. If the intervals I' and $I^i \in I$ have a nonempty intersection region $\delta = I' \cap I^i$, a quadratic equation needs to be solved to determine the minimal distance for points in δ and update the intervals I' and I^i along edge e. Letting $I_{\text{updated}} = \{I^0, I^1, \cdots\}$ be the set of updated intervals on e, four degenerate cases may occur:

1. Tiny intervals appear in I.
2. Two consecutive intervals in I intersect.
3. Two consecutive intervals in I separate by a tiny gap.
4. The geodesic distances at the common endpoint of two consecutive intervals are not the same.

Theoretically, if exact arithmetic is used, these cases will not happen or can be regarded as *errors*. However, in practice, when float point computation is used and numerical errors are unavoidable, these cases do occur and we regard them as *degenerate cases*. The solution to handle these degeneracies is presented in Sect. 4.2.

4 Handling degenerate cases

In geometric algorithms, testing degenerate cases relies heavily on the incidence decisions such as whether a point lies on a line or two points coincide [2]. Incidence decisions contribute to geometric predicates. A *predicate* is a numerical primitive computation whose value impacts the flow of control of an algorithm. To evaluate predicates with float point computation, we present a systematic solution in the following subsections. The pseudo-code of the overall algorithm is as follows:

Algorithm 1.

```
1.      Initialize a priority queue $\mathcal{Q}$ with a given
        source point in the mesh;
2.      while $\mathcal{Q}$ is not empty
2.1.        pop off the top element $q$ from $\mathcal{Q}$;
2.2.        establish the local system as shown in Fig. 3
>       based on $q = (b_0, b_1, d_0, d_1, \sigma, \tau)$;
2.3.        find the intersection of the wedge $b_0 \to s \to b_1$
            and $e_1, e_2$; handle the degeneracies using the
            solution presented in Sect. 4.1;
2.4.        update intervals on $e_1, e_2$ and $\mathcal{Q}$ using the
            solution presented in Sect. 4.2;
2.5.        if new intervals created
2.5.1.          add them into $\mathcal{Q}$.
```

4.1 Handling degeneracies on geometric intersection

Suppose that we implement the vector operation in a C++ class. Given a point (or a vector) p, $p.x$, $p.y$, $p.z$ retrieve its three coordinates. The term $p.length()$ returns the value of the vector length, $p \cdot q$ returns the value of the inner product of two vectors p, q, $p \times q$ returns the vector of the cross product of p, q, $abs(c)$ returns the absolute value

Table 1. Degeneracies tests on all the examples; the degeneracy rate is measured by dividing the degenerate cases resulted from geometric intersection over all the cases

Model	Face num.	All cases	Degeneracy rate
Fig. 7a	2000	8807	29.33%
Fig. 7b	5000	23 221	31.13%
Fig. 8	47 415	194 851	33.25%
Fig. 9a	12 436	49 697	32.40%
Fig. 9b	11 000	49 028	29.22%
Fig. 9c	11 774	63 992	27.96%
Fig. 9d	12 000	49 621	31.41%
Fig. 9e	21 152	82 397	35.27%

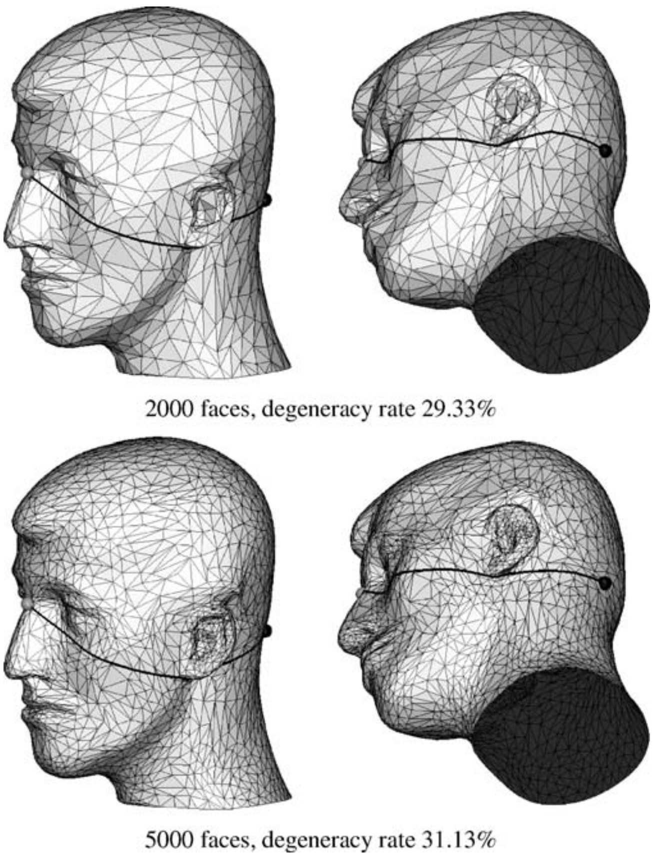

2000 faces, degeneracy rate 29.33%
5000 faces, degeneracy rate 31.13%

Fig. 7. An exact geodesic path over the head model with two different resolution meshes; the *colored* distance field is shown in Fig. 1

of c. Denote the machine precision by ϵ (see Fig. 3). The following rules consist of incidence decisions:

- If $(s - b_0).length() < \epsilon$, points s and b_0 coincide.
- If $(s - b_1).length() < \epsilon$, points s and b_1 coincide.
- If $abs(((s - b_0) \times (b_0 - v_1)).z) < \epsilon$, three points s, b_0, v_1 lie on a straight line.
- If $((b_1 - s) \times (v_1 - b_1)).z > \epsilon$, the vertex v_1 lies to the right of the wedge and the new interval will be on the e_1. That means case **a** in Fig. 3 occurs.
- If $((b_0 - s) \times (v_1 - b_0)).z < -\epsilon$, the vertex v_1 lies to the left of the wedge and the new interval will be on the e_2.
- If $((b_0 - s) \times (v_1 - b_0)).z > \epsilon$ and $((b_1 - s) \times (v_1 - b_1)).z < -\epsilon$, the vertex v_1 lies inside the wedge formed by two rays $b_0 - s$ and $b_1 - s$. That means case **b** in Fig. 3 occurs.

Given the above rules, our goal is to design a decision procedure that reduces all possible decisions to a set of as few as possible predicates, which also guarantee to output a consistent and correct decision on choosing the order of different degenerate cases. We present such a nontrivial decision tree in Fig. 6. Given the rules of incidence decisions and the decision tree as shown in Fig. 6, the code that can robustly and consistently handle all the degenerate cases in this class is readily built.

4.2 Handling degeneracies on geodesic discontinuities

Here we present a robust solution to handling degeneracies on geodesic discontinuities. The presented solution may seem unnecessarily complicated at first glance. However, it not only gives us a concise way of programming, but also makes verification and error estimation possible and easy to realize at each step by providing a deterministic status to check. The pseudo-codes handling degeneracies on geodesic discontinuities (see Step 2.4 in Algorithm 1) are as follows:

Algorithm 2.

```
1.        for all $I^i \in I$
1.1.          let $interb0 = \max\{I^i.b_0, I'.b_0\}$, and
                  $interb1 = \min\{I^i.b_1, I'.b_1\}$;
1.2.          if $interb0 < interb1$
1.2.1.            if $I^i.b_0 < interb0$
1.2.1.1.              separate $I^i$ at $interb0$;
1.2.1.2.              let $I^i_{new} = (I^i.b_0, interb0)$ and
                          $I^i = (interb0, I^i.b_1)$;
1.2.1.3.              insert $I^i_{new}$ into $I$;
1.2.2.            if $interb1 < I^i.b_1$
1.2.2.1.              separate $I^i$ at $interb1$;
1.2.2.2.              let $I^i_{new} = (I^i.b_0, interb1)$ and
                          $I^i = (interb1, I^i.b_1)$;
1.2.2.3.              insert $I^i_{new}$ into $I$;
2.        for all $I^i \in I$ which completely inside $I'$
2.1.          update $I^i$ and $I'$ by solving Eq. 1;
3.        Remove tiny intervals in $I$;
4.        Sew small gaps in $I$;
5.        In $I$ merge neighbor intervals with the
          same geodesic vertex;
6.        (Optional) verification of set $I$ if needed.
```

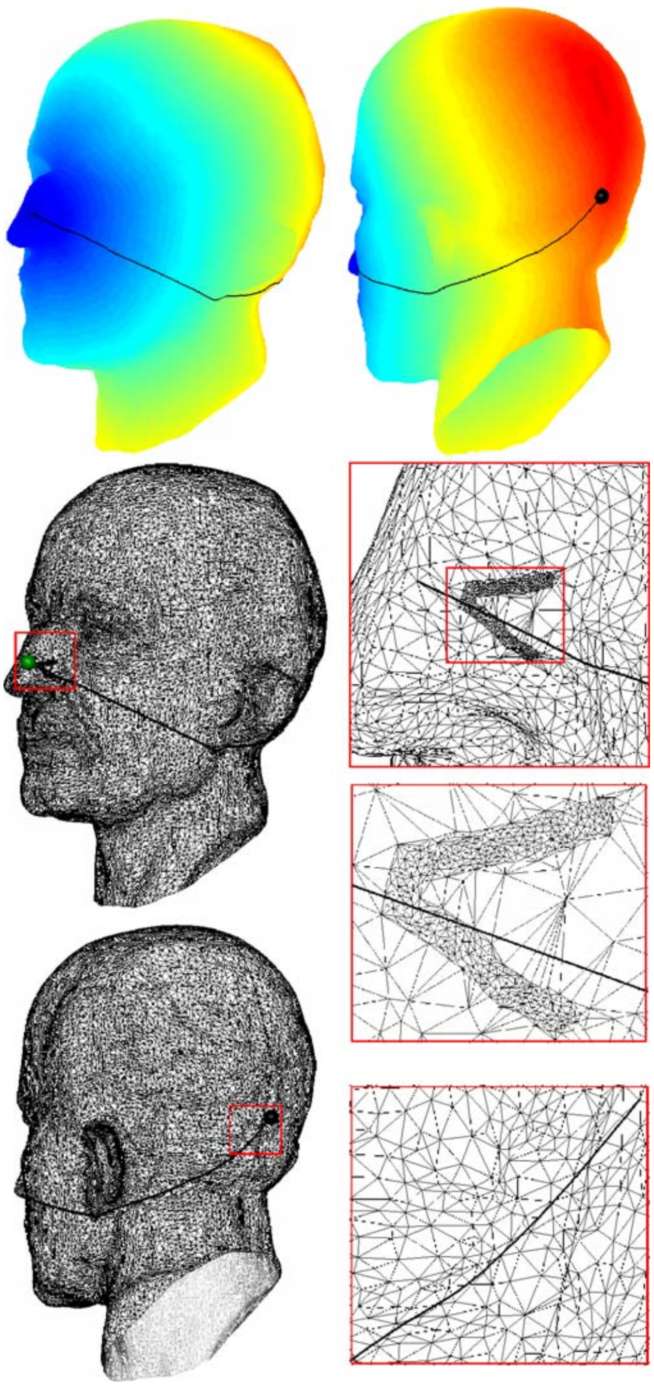


Fig. 8. The exact geodesics over the Max-Planck head model which possesses different resolution over different regions. The code must be robust against large and small triangles simultaneously existed on a single mesh. The degeneracy rate of this model is 33.25%

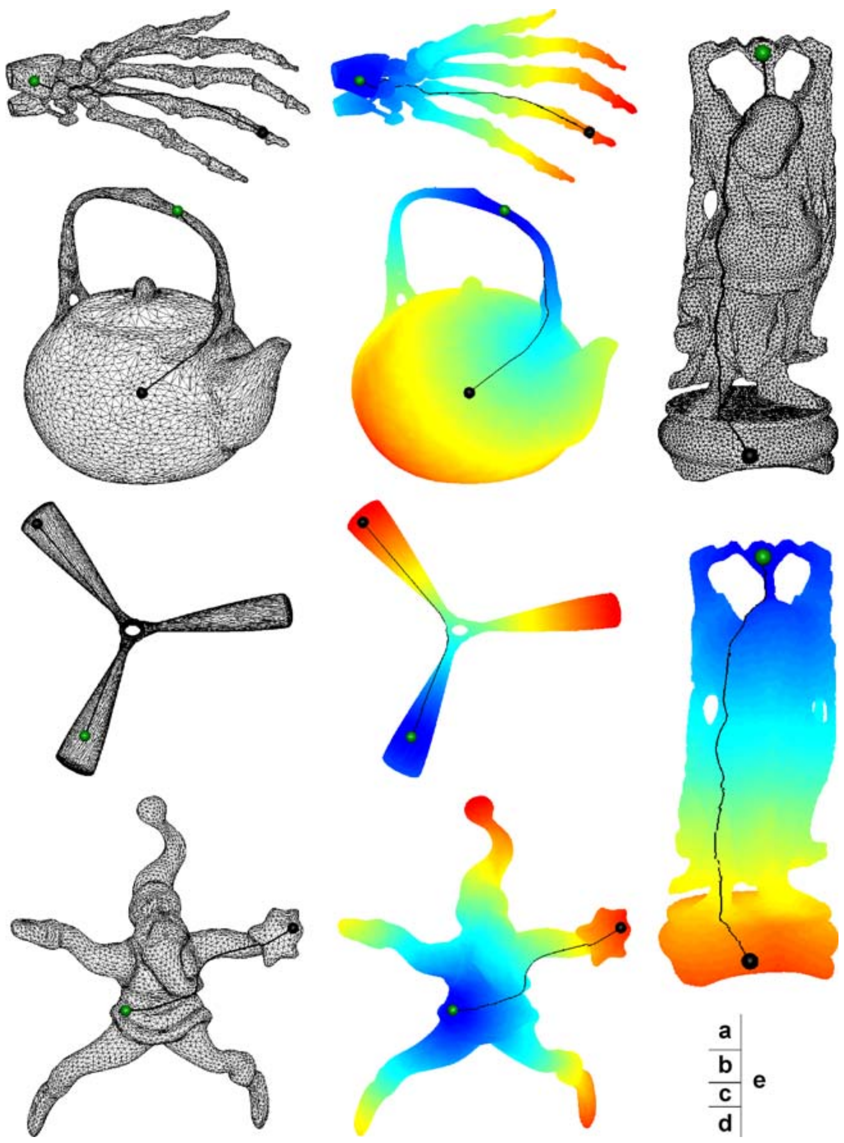


Fig. 9. The computation of exact geodesics over the diverse models with arbitrary triangles; the distance fields are *colored* by one-to-one mapping the geodesic lengths to an indexed color map

Given the newly created interval I' and a set of already existing intervals $I = \{I^0, I^1, \cdots\}$ on edge e, we first process all intervals in I such that for each interval in I, it is either completely outside range I' or completely inside I'. This process is illustrated in Fig. 5c and Step 1 in Algorithm 2 serves this need.

At Step 2 in Algorithm 2, denote the sorted subset by I_{inside} whose elements are completely inside the range of the new interval I'. We update intervals in I_{inside} in turn. Given $I^i \in I_{\text{inside}}$ and I', a quadratic equation is solved. According to the solution, $I^i = (I^i.b_0, I^i.b_1)$ may disappear or shrink into a smaller interval $I^i_{\text{new}} = (I^i_{\text{new}}.b_0, I^i_{\text{new}}.b_1)$. In the latter case, we divide interval $I' = (I'.b_0, I'.b_1)$ into two parts, i.e., $I'_{\text{new}} = (I'.b_0, I^i_{\text{new}}.b_0)$ and $I' = (I^i_{\text{new}}.b_1, I'.b_1)$, and insert I'_{new} into I. Then we continue to process I^{i+1} with I' until all elements in I_{inside} are processed.

Finally, we obtain an updated interval set I. It is not difficult to check that given the above rules, the elements in I cannot intersect with each other. Due to numerical computation, tiny intervals and small gaps may occur (see Fig. 5d and Steps 3, 4, and 5 in Algorithm 2). The following rules handle these degeneracies:

1. *Detect and remove tiny intervals.* $\forall I^i \in I$, if $I^i.b_1 - I^i.b_0 < \epsilon$, merge I^i with I^{i-1} or I^{i+1};
2. *Detect and sew small gaps.* If $I^{i+1}.b_0 - I^i.b_1 < \epsilon$, let $I^{i+1}.b_0 = I^i.b_1$ be the midpoint of the original $I^{i+1}.b_0$ and $I^i.b_1$.

3. *Merge intervals with the same source point*. For any pair I^i and I^{i+1}, let the unfolded position of geodesic vertex be s_i and s_{i+1}, respectively. If $(s_i - s_{i+1}).length() < \varepsilon$, merge intervals I^i and I^{i+1}.

The biggest advantage of Algorithm 2 is that every step is predictable and thus code verification is easy to check.

5 Results

By handling all the degenerate cases consistently and correctly, the implementation of the exact geodesic algorithm [3, 5] is very robust. In this section, we present some test examples with the models of various distributions of triangles. In each example, the small green sphere indicates the position of the prescribed source point with which a distance field is built and colored by computing the length of geodesic paths from the source to all other points on meshes. By tracing the gradient of the distance field, a geodesic path from the source to a destination point on mesh is also shown in each example. In all examples shown here, the degeneracy rate is measured by the percentage of degenerate cases over all the cases. Table 1 summarizes the degeneracy tests on all the examples.

In Fig. 7, a head example with two different resolution models is presented. Both models consist of irregular triangles. On both models, the source and destination points are the same and the geodesic paths connecting them are shown. In Fig. 8, the test is performed on the Max-Planck head model. This model possesses different mesh resolution over different regions. On this model, a geodesic path crossing regions of different resolutions is shown. These two examples show that (1) the smaller the triangles are, the more degenerate cases occur, and (2) the more irregular the triangle distribution is, the more degenerate cases occur.

We also test the implementation on a diversity of models with arbitrary triangles. Five typical examples are shown in Fig. 9. These examples show that real-world data is likely to contain a large number of degeneracies. By providing a concise and consistent solution to all the degenerate cases, the users may find the technique presented in this paper useful when he/she implements a robust code to compute an exact geodesic over triangle meshes.

6 Conclusions

Geometric algorithms are sensitive to degeneracies arising from special positions of several incident geometric objects. Although the general technique [1, 4] exists to handle the degeneracies theoretically in any geometric algorithms, certain particular applications permit much more efficient ways to handle degeneracies. In this paper we classify and enumerate all the degenerate cases in the computation of exact geodesics on triangle meshes. Based on the classification, we present a systematic treatment to handle all the degeneracies consistently. We also show by examples that the real-world data is likely to be degenerate. The common users may find the presented technique useful to obtain a robust implementation of the fast exact geodesic algorithm.

Acknowledgement The authors are supported by the National Natural Science Foundation of China (Project No. 60603085), the National High Technology Research and Development Program of China (Project No. 2006AA01Z304), the National Basic Research Program of China (Project No. 2006CB303102) and the Project JC2007022 funded by Tsinghua Basic Research Foundation.

References

1. Edelsbrunner, H., Mucke, E.: Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. ACM Trans. Graph. **9**(1), 66–104 (1990)
2. Hoffmann, C.: Robustness in geometric computations. J. Comput. Info. Sci. Eng. **1**(2), 143–155 (2001)
3. Mitchell, J., Mount, D., Papadimitriou, C.: The discrete geodesic problem. SIAM J. Comput. **16**(4), 647–668 (1987)
4. Sugihara, K., Iri, M., Inagaki, H., Imai, T.: Topology-oriented implementation: an approach to robust geometric algorithms. Algorithmica **27**(11), 5–20 (2000)
5. Surazhsky, V., Surazhsky, T., Kirsanov, D., Gortler, S., Hoppe, H.: Fast exact and approximate geodesics on meshes. In: ACM SIGGRAPH 2005, pp. 553–560 (2005)
6. Zachary, J.: Introduction to Scientific Programming. TELOS, Santa Clara, CA (1998)

YONG-JIN LIU is an assistant professor of computer science at Tsinghua University. He received his Ph.D. degree from Hong Kong University of Science and Technology in 2003. His research interests include computer graphics and computer-aided geometric design.

QIAN-YI ZHOU is a Master's student in the Department of Computer Science and Technology at Tsinghua University. He received his Bachelor's degree in computer science from Tsinghua University in 2005. His research interests include computer graphics, geometry modeling, mesh processing and CAGD.

SHI-MIN HU is currently a professor of computer science at Tsinghua University. He obtained his Ph.D. in 1996 from Zhejiang University. His research interests include digital geometry processing, video-based rendering, computer animation and computer-aided geometric design.