The Visual Computer manuscript No.
(will be inserted by the editor)

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

Handling Degenerate Cases in Exact Geodesic Computation on Triangle Meshes

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 (2005). 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 sub-quadratic time.

The implementation in [5] can be regarded as a *generic algorithm*, i.e., it is guaranteed to be correct with generic situation, while 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 are 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 in Fig. 4.

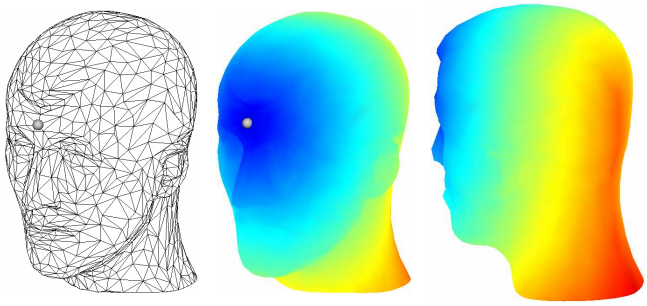


Fig. 1 Geodesic computation with a prescribed source point; points on the mesh are colored according to the geodesic distance to the source point.

In geometric computation, degenerate cases will increase the instability of the generic algorithm. Theoretically, degenerate cases can be handled by using the symbolic perturbation schemes [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

Department of Computer Science and Technology,
Tsinghua University, P. R. China
Tel.: +86-10-62784141
Fax: +86-10-62771138
E-mail: {liuyongjin, zqy, shimin}@tsinghua.edu.cn

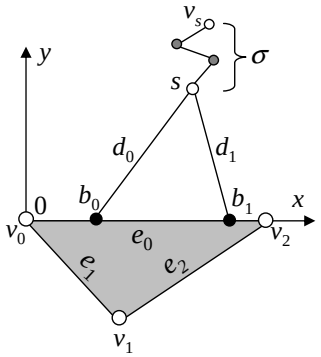


Fig. 2 6-tuple representation $(b_0, b_1, d_0, d_1, \sigma, \tau)$ of the interval.

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.

2 Review of the exact geodesic algorithm

We follow the notation in [5] to quick review the DGP algorithm [3]. Shortest paths on mesh are rays emanating from the source vertex along tangent directions. Interior to 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 (called *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. Refer to Fig. 2. Each interval is encoded by a 6-tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$. 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. σ is the length of the path from s back to the source v_s.

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. Refer to 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

$$Ap^2 + Bp + C = 0 \tag{1}$$

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 showing in Fig. 3. The reader is referred to [5] for full details of this algorithm.

3 Degenerate cases

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

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

3.1 Degeneracies on geometric intersection

Basically, there are 5 degenerate cases in this class, as shown in Fig. 4:

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. 3(c);
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. 3(b);
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 the e_1 in the case of Fig. 3(b) must be treated as the new interval on the e_1 in the case of Fig. 3(c);
5. Four points s, v_1, b_1, v_1 are in a straight line. This is a symmetric case of case 4.

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 coincides, the basic degenerate cases 1,2,4 occur simultaneously. Different degenerate cases must awake different procedures to process. Treating degenerate cases in random order will result in catastrophic failures in the algorithm. In Section 4.1, we present a concise decision procedure to properly handle all the degenerate cases.

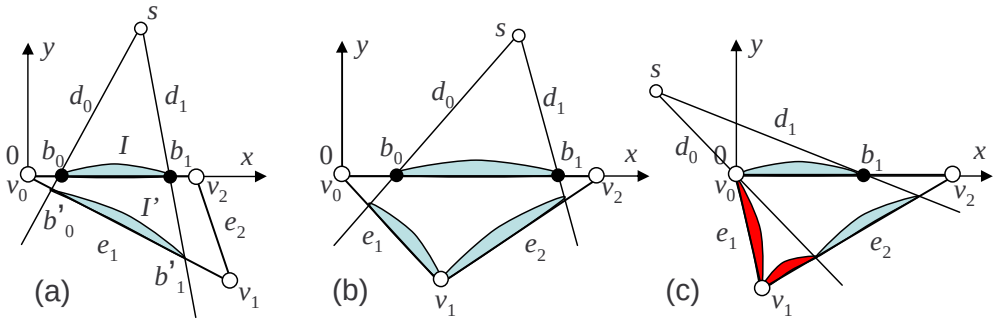


Fig. 3 Interval propagation. (a)One new interval created. (b) Two new intervals created. (c) One new normal interval and two additional intervals (in red) created.

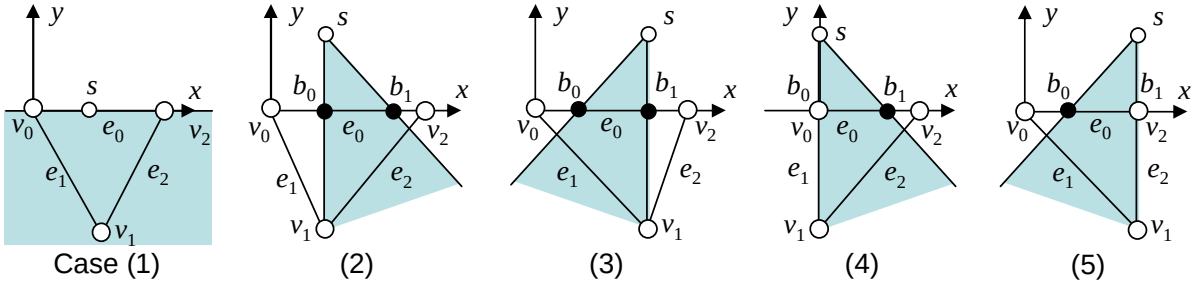


Fig. 4 Degenerate cases on geometric intersection; the shaded area indicates the wedge range of $b_0 \rightarrow s \rightarrow b_1$.

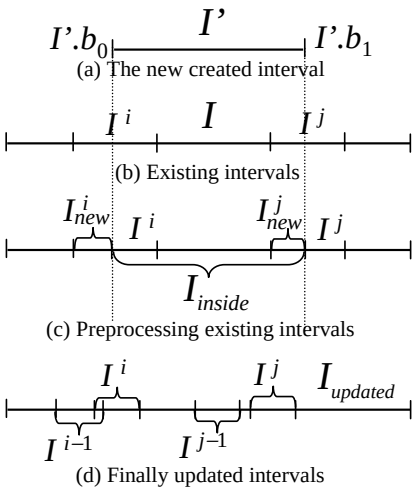


Fig. 5 Degeneracies on geodesic discontinuities.

3.2 Degeneracies on geodesic discontinuities

After the determination of intersection region between wedge $b_0 \rightarrow s \rightarrow b_1$ and edges e_1, e_2, new intervals are created. Refer to 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, $I^{i-1}.b_1 \leq I^i.b_0 < I^i.b_1 \leq I^{i+1}.b_0$. 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. Let $I_{updated} = \{I^0, I^1, \cdots\}$ 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 intersects;
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 Section 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.

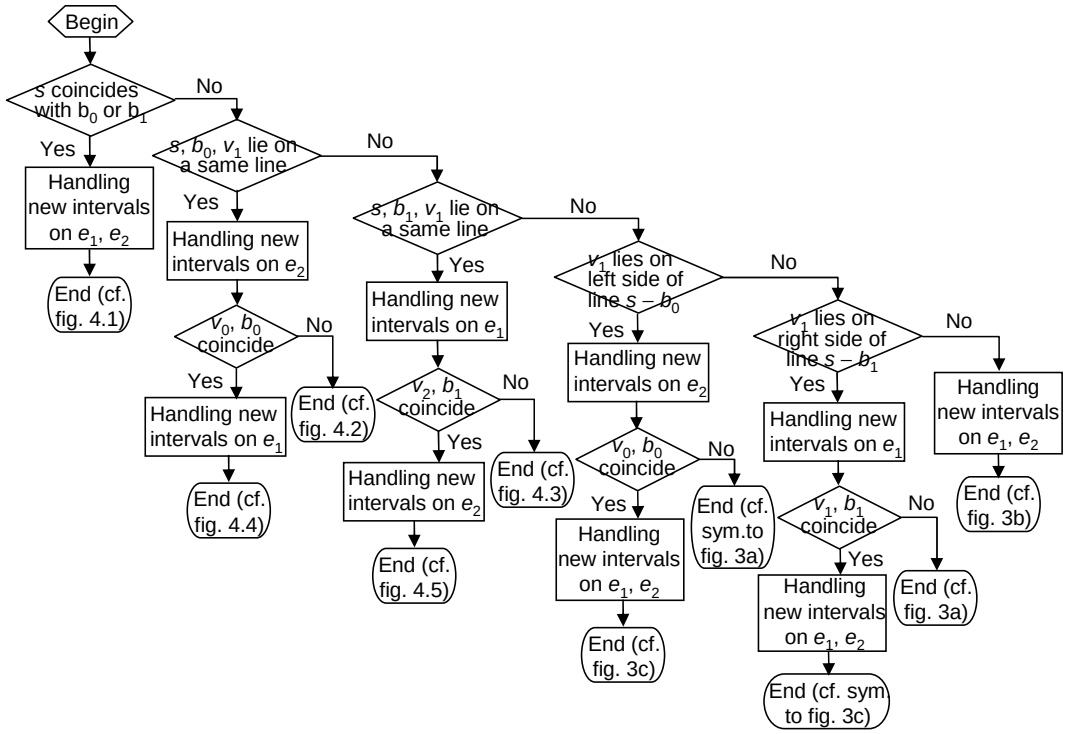


Fig. 6 The flowchart of the decision system to handling degeneracies on geometric intersection.

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 Sec. 4.1;
2.4. update intervals on e_1, e_2 and $\mathcal{Q}$ using the solution presented in Sec. 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. $p.length()$ return 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 of c. Denote the machine precision by ϵ. Refer to 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 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 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 reduce all possible decisions to a set of predicates as few as possible, which also guarantee to output a consist and right decision on choosing the order of different degenerate cases. We present such a non-trivial 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 to build.

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 the first glance. However, it not only give us a concise way of programming, but also it makes verification and error estimation possible and easy to realize at each step by providing deterministic status to check. The pseudo-codes handling

degeneracies on geodesic discontinuities (ref. the Step 2.3 in Algorithm 1) are as follows.

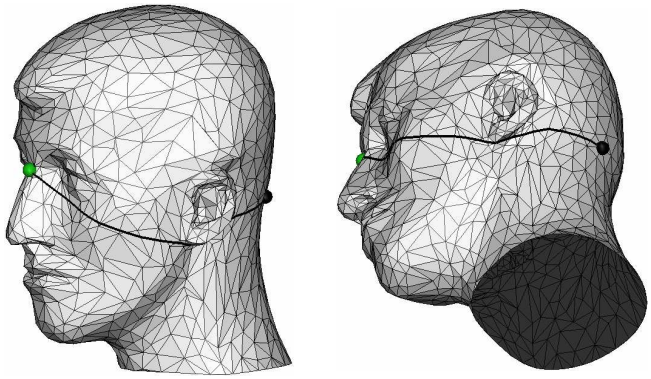
Algorithm 2
1. **for** all $I^i \in I$
1.1. let $interb0 = \max\{I^i.b_0, I'.b_0\}$, and $interb1 = \min\{I^i.b_1, I'.b_1\}$;
1.2. **if** $interb0 < interb1$
1.2.1. **if** $I^i.b_0 < interb0$
1.2.1.1. separate I^i at $interb0$;
1.2.1.2. let $I^i_{new} = (I^i.b_0, interb0)$ and $I^i = (interb0, I^i.b_1)$;
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 equation (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 I if needed.

Given the newly created interval I' and a set of already existed 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.

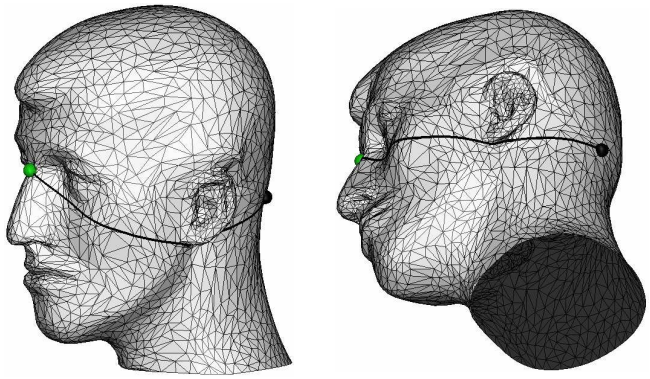
At Step 2 in Algorithm 2, denote the sorted subset by I_{inside} whose elements are completely inside the range of the new interval I'. We update intervals in I_{inside} in turn. Given $I_i \in I_{inside}$ and I', a quadratic equation is solved. According to the solution, $I_i = (I_i.b_0, I_i.b_1)$ may disappear or shrink into a smaller interval $I_{inew} = (I_{inew}.b_0, I_{inew}.b_1)$. In the latter case, we divide interval $I' = (I'.b_0, I'.b_1)$ into two parts, i.e., $I'_{new} = (I'.b_0, I_{inew}.b_0)$ and $I' = (I_{inew}.b_1, I'.b_1)$, 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 get an updated interval set I. It is not difficult to check that given the above rules, the elements in I cannot be intersected to each other. Due to numerical computation, tiny intervals and small gaps may occur. Refer to Fig. 5c and Steps 3,4,5 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}.



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 biggest advantage of Algorithm 2 is the result of 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 testing examples with the models of various distribution of triangles. In each example, the small green sphere indicates the position of the prescribed source point with which a distance field is built 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 degeneracies 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

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
fig7a	2000	8807	29.33%
fig7b	5000	23221	31.13%
fig8	47415	194851	33.25%
fig9a	12436	49697	32.40%
fig9b	11000	49028	29.22%
fig9c	11774	63992	27.96%
fig9d	12000	49621	31.41%
fig9e	21152	82397	35.27%

them are shown. In Fig. 8, the test is performed on the maxplunck 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 shows that (1) more smaller the triangles are, more degenerate cases occurs; (2) more irregular the triangle distribution is, more degenerate cases occurs.

We also test the implementation on a diversity of models with arbitrary triangles. Four 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 exact geodesic over triangle meshes.

6 Conclusions

Geometric algorithms are sensitive to degeneracies risen 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.

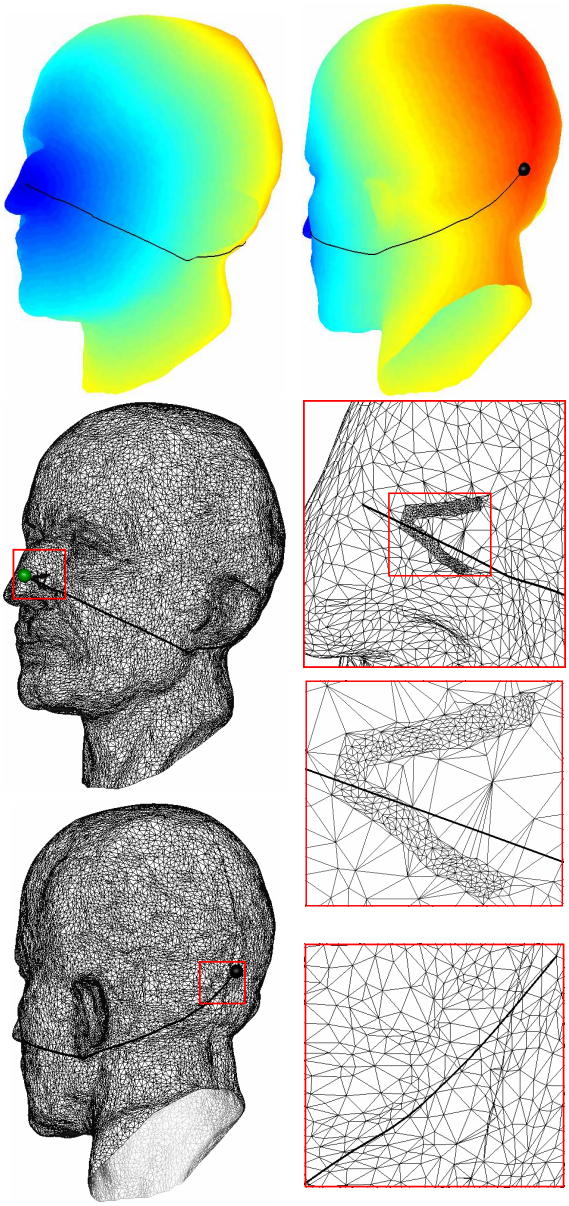


Fig. 8 The exact geodesics over the maxplunck head model which possesses different resolution over different regions. The code must be robust against large and small triangles simultaneously existed on a single mesh. The degeneracy rate of this model is 33.25%.

References

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

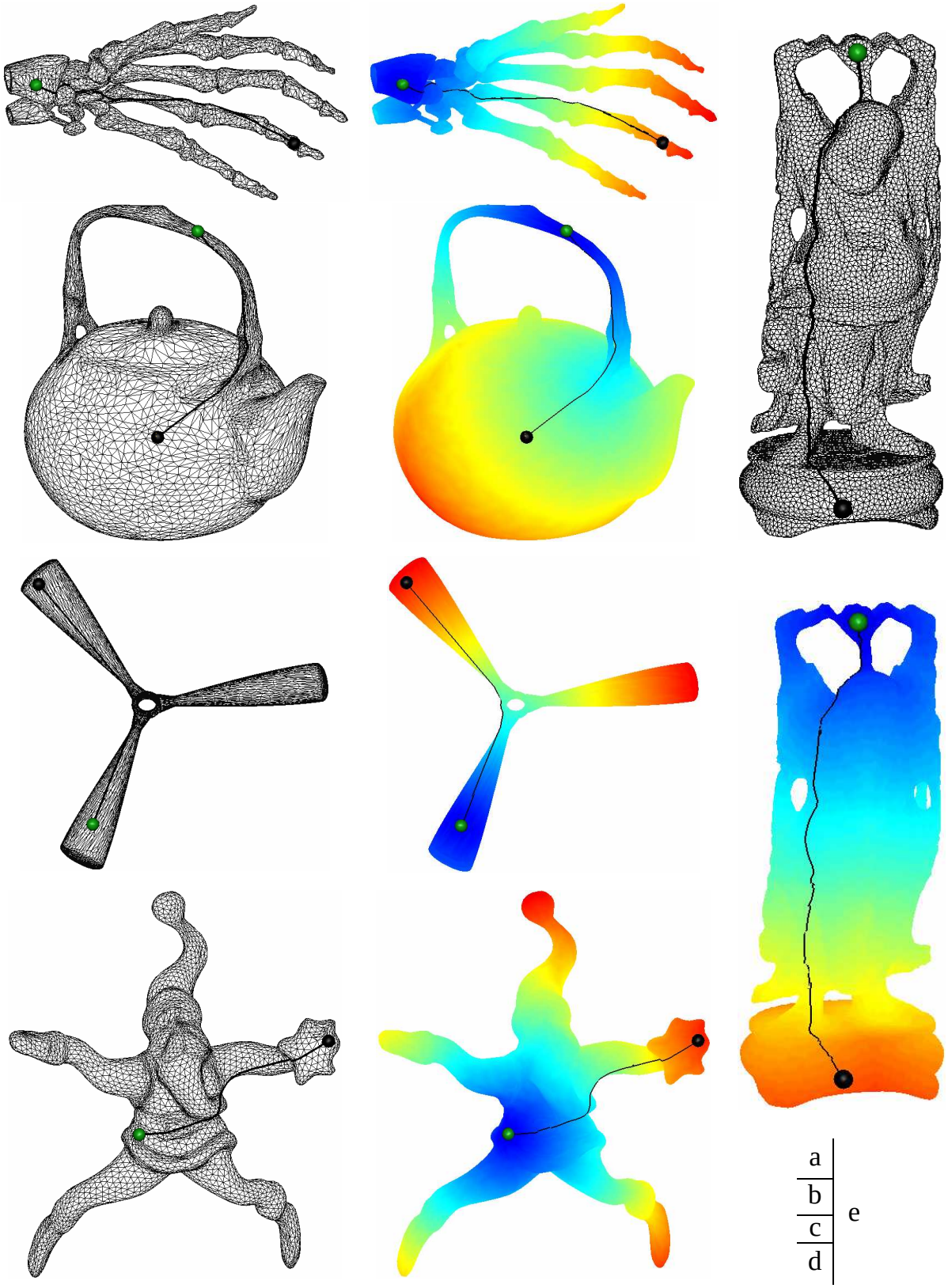


Fig. 9 The computation of exact geodesics over the diverse models with arbitrary triangles.