Point-In-Polyhedra Test with Direct Handling of Degeneracies

CUI Shulin1, ZHANG Shuqing2, CHEN Xuanxi1, PANG Zhenping1, FU Xiaoyang1, ZHANG Xu3
1. Department of Computer Science and Technology, Zhuhai College of Jilin University, Zhuhai 519041, China
2. Northeast Institute of Geography and Agricultural Ecology, Chinese Academy of Sciences, Changchun 130012, China
3. Department of Arts, Zhuhai College of Jilin University, Zhuhai 519041, China
© Wuhan University and Springer-Verlag Berlin Heidelberg 2011

Abstract The Point-In-Polyhedron problem is to check whether a point is inside or outside of a given polyhedron. When a degenerate case is detected, the traditional ray-crossing algorithms avoid the case by selecting a different ray or erase the case by perturbing input data. This paper introduces a Threshold-Based Ray-Crossing (TBRC) algorithm for solving the Point-In-Polyhedron problem. The TBRC algorithm cope directly with degenerate cases by checking whether to count the face intersecting with the ray. It is worth mentioning that the TBRC algorithm can handle all degeneracies without extra computation and storage. Moreover, we analyze the basic algorithm and examine how to accelerate it. The experimental results show that TBRC algorithm is highly efficient and robust for the Point-In-Polyhedron problem, compared to a classical tetrahedron-based algorithm without pre-processing.

Keywords TBRC; the edge-face problem; Point-In-Polyhedron

Introduction

Because the increasingly complex tasks in many applications (such as urban planning, 3D cadastre, telecommunications, etc.) require the capability to model, analyze and visualize 3D data in an efficient and effective way, 3D geographical information systems (3D GIS) have become an important research area in the last decade or so.[1-8] One of the major challenging tasks is to support spatial analysis among different types of real 3D objects, since conventional 2D GIS methods are inadequate for performing 3D spatial analysis.[9-11] New algorithms operating on complicated 3D models are hence needed, and such a kind of studies draws more and more attention from GIS researchers. For instance, two basic intersection predicates including “line traversing” and “point inclusion” for geometric intersection between any two kinds of objects has been abstracted,[1] but when dealing with non-convex polyhedron, an avoidance strategy is adopted due to the degeneracy problem.

The point inclusion test checking whether a point is inside a given polyhedron is one of the most basic operations, which has been investigated widely in many computing areas such as computer graphics, computational geometry and virtual reality.[12,13] The inclusion test is not a complex problem, but a robust, correct and efficient algorithm is needed because the basic test has to be applied many times in the geo-
metric intersection test that should be performed within reasonable computational time.\cite{14,15} Moreover, the point inclusion test lays the foundation of other geometric computation for complex 3D models, e.g., geometric intersection for general polyhedra.

According to their requirements for pre-processing, the existing algorithms can be classified into two categories. The algorithms with pre-processing include octrees, BSP, and layer-based.\cite{13,16,17} It depends on some factors to carry out a pre-processing algorithm, including the spatial data models, the number of repetitions, the memory resources and the pre-processing time. Each approach has a different time and storage complexity for pre-processing, and time complexity for inclusion query, so no single one should be considered as the best for all situations.\cite{16}

The main algorithms without pre-processing, include voxel-based, ray-crossing, signed area, and tetrahedron-based.\cite{18,19} In fact, the ray-crossing algorithm could be considered the fastest way to solve the Point-In-Polyhedron problem.\cite{20} The main drawback of this method is that it is incapable of handling degeneracies that occur when the ray crosses edges, vertices, etc. Degeneracies can be avoided by selecting a different ray.\cite{21} However, it needs \( O(n) \) time to find this ray.

The treatment of degenerate cases is tedious and intricate, thus seldom included in the theoretical discussion. However, it remains a nontrivial matter for implementors.\cite{22} Besides degeneracy avoidance, a perturbation scheme that changes the original input instance into a nondegenerate one is often adopted.\cite{23,24,25} However, these strategies incur an extra computational cost. This paper presents a theoretical and practical approach that can be used to handle degeneracies without any additional calculations and storage.

The paper is organised as follows. Section 1 gives some fundamental definitions about the newly proposed method. Section 2 describes the edge-face problem in 3D as well as a related example. Two versions of the Threshold-Based Ray-Crossing (TBRC) algorithm for point inclusion test are introduced in Section 3 and Section 4 considers how to accelerate them. The performance of TBRC algorithm is demonstrated through experiments in Section 5, by comparing with the tetrahedron-based algorithm. Finally, conclusions are drawn in Section 6.

1 Basic definitions

Consider a polyhedron as a geometric solid in 3D with \( n \) flat faces, \( p = (f_1,\cdots,f_j,\cdots,f_n) \), where all points constructing a face must be in the same plane. Let the \( j \)th face \( f_j \) be given by \( m \) vertices going counter-clockwise around the face, \( f_j = (v_1,\cdots,v_j,\cdots,v_m) \), and let \( v_{m+1} = v_1 \). Also, let the \( j \)th edge \( e_j \) (line segment) be implicitly defined by two successive vertices, \( e_j = \langle v_j, v_{j+1}, j \in (1,2,\cdots,m) \rangle \). There are two basic contacts between vertices, edges and faces in 3D, Vertex-Face called Type-A contact and Edge-Edge called Type-B contact.\cite{26} These basic contacts are redefined here on the basis of our algorithms.

**Definition 1.** Type-A contact takes place when face \( f_i \) of the polyhedron is in contact with point \( v_i \), which is defined by

\[
A_{v_i,f_i} = \begin{cases} 1, & \text{if } v_i \cdot v_j, v_k > 0 \\ 0, & \text{if } v_i \cdot v_j, v_k \leq 0 \end{cases}
\]  

(1)

where \( \{v_i, v_j, v_k\} \) is an ordered arbitrary representative set of vertices of face \( f_i \). If \( v_i \) is on the negative side of the plane defined by \( v_1, v_2 \), and \( v_i \) (called the \( f_i \) plane), then \( A_{v_i,f_i} = 1 \); otherwise, \( A_{v_i,f_i} = 0 \).

**Definition 2.** Type-B contact takes place when edge \( e_j = \langle v_j, v_{j+1} \rangle \) of \( f_j \) is in contact with line segment \( e = \langle v_1, v_k \rangle \), which is defined by

\[
B_{v_j,e_j} = \begin{cases} 1, & \text{if } v_j \cdot v_1, v_k > 0 \\ 0, & \text{if } v_j \cdot v_1, v_k \leq 0 \end{cases}
\]  

(2)

For example, if \( e_j \) rotates around \( e \) in counterclockwise direction, then \( B_{v_j,e_j} = 1 \); otherwise, \( B_{v_j,e_j} = 0 \). Similarly, If \( e_j \) rotates around \( e \) in clockwise direction, then \( B_{v_j,e_j} = 1 \), where \( -e_j = \langle v_{j+1}, v_j \rangle \).

2 The edge-face problem in 3D

In this section, we define a robust method to compute the intersection between line segment \( e = \langle v_1, v_k \rangle \) and face \( f_j \) of the polyhedron. The goal of the following method is first to determine if the two endpoints of the line segment lie on opposite sides of the \( f_j \) plane, and then to determine if the in-
tersection point is located inside \( f_i \).

**Step 1.** Intersecting the \( f_i \) plane

Step 1 could be determined by

\[
X_{e,f_i} = A_{v,f_i} \oplus A_{v,f_i}
\]

(3)

If \( v_j, v_k \) are on the opposite sides, then \( X_{e,f_i} = 1 \), otherwise, \( X_{e,f_i} = 0 \). The logical operation exclusive \( \ominus \) (XOR) is a type of logical disjunction on two operands that results in a value of true if exactly one of the operands has a value of true. If \( X_{e,f_i} = 1 \), then a further test will be performed.

**Step 2.** Intersecting the face \( f_i \)

Fig. 1 demonstrates a typical case of a concave polygon \( f_i \) in 3D. \( e \) is a line segment that needs to be tested, i.e., to determine if it intersects with the polygon. Consider a \( f = (v_j, v_k, v_l) \) plane containing \( e \) and any other point, say \( v_i \). The solution is to compare each edge of the polygon with the \( f \) plane (the plane is called the \( f \) threshold), so that all edges whose endpoints are not on opposite sides of the \( f \) plane are discarded. To deal with degeneracies, vertices exactly on the \( f \) plane must be considered to belong to one side of the \( f \) plane. According to the Definition 1, they will belong to the positive side of the \( f \) plane.

![Fig.1 An example of the edge-face problem with degeneracies](image)

The line supporting \( e \) divides the \( f \) plane into two half planes (called the half-\( f \) planes). If the number of edges intersecting each the half-\( f \) plane is odd, then \( f_i \) is intersected by \( e \); otherwise, \( f_i \) is not intersected by \( e \).

Thus, the edge \( e_j \) intersecting the half-\( f \) plane to the right of the line supporting \( e \), could be expressed as,

\[
Y_{e,f_i} = (\neg A_{v,f_i} \land A_{v,f_i} \land B_{v_j}) \lor
(A_{v,f_i} \land \neg A_{v,f_i} \land B_{e,e_j})
\]

(4)

where \( \land, \lor \) and \( \neg \) denote logic operator AND, OR and NOT respectively. If \( Y_{e,f_i} = 1 \), then \( e_j \) is counted, denoting that a node is generated; otherwise, it is not counted. \( B_{e,e_i} \) and \( B_{e,e_j} \) guarantee that edges intersecting the left half-\( f \) plane will not be counted, where no node is generated.

Whether the intersection point between \( e \) and \( f_i \) plane is inside of \( f_i \) can then be checked as follows,

\[
Y_{e,f_i} = \odot Y_{e,f_i}
\]

(5)

The values of \( A_{v,f_i} \) in Fig.1 are given in Table 1. For example, as \( v_1 \) belongs to the positive side of the \( f \) plane, we then have \( |v_1,v_1,v_1| < 0 \) and \( A_{v_1,f} = 0 \). Point \( v_3, v_6 \) and \( v_j \) are exactly on the \( f \) plane, but they belong to the positive side of the \( f \) plane; we thus have \( A_{v_3,f} = 0, A_{v_6,f} = 0, \) and \( A_{v_j,f} = 0 \).

According to the values of \( A_{v,f_i} \), whether edge \( j \) intersects with the \( f \) plane can be determined. For example, \( A_{v_1,f} = 0 \) and \( A_{v_1,f} = 1 \) \( \Rightarrow \) edge \( e_i = \langle v_1, v_1, v_1 \rangle \) intersects with the \( f \) plane; but \( A_{v_1,f} = 0 \) and \( A_{v_1,f} = 0 \) \( \Rightarrow \) edge \( e_i = \langle v_1, v_1, v_1 \rangle \) does not intersect. We finally know that \( e_i (i = 1, 2, 3, 4, 7, 9) \) intersects with the \( f \) plane, while \( e_i (j = 5, 6, 8, 10) \) does not.

**Table 1** The values of \( A_{v,f_i} \) in Fig.1

| \( v_j \) | \( v_1 \) | \( v_2 \) | \( v_3 \) | \( v_4 \) | \( v_5 \) | \( v_6 \) | \( v_7 \) | \( v_8 \) | \( v_9 \) |
|---|---|---|---|---|---|---|---|---|---|
| \( A_{v,f_i} \) | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

The values of \( B_{e,e_i} \) and \( Y_{e,f_i} \) in Fig.1 are given in Table 2. Among edges \( e_1, e_2, e_3, e_4, e_5, e_6 \) and \( e_9 \), the only one to the left side of line segment \( e = \langle v_1, v_1, v_1 \rangle \) is \( e_9 \), thus we have \( Y_{e_1,f} = 0 \) and \( Y_{e_9,f} = 1 \) (where \( j = 1, 2, 3, 4, 7 \)). That is, edge \( e_1, e_2, e_3, e_4, e_5, e_6, e_9 \) generate a node, but \( e_9 \) does not. Therefore, \( Y_{e_1,f} = 1 \odot 1 \odot 1 \odot 1 \odot 1 \odot 0 \odot 0 \odot 0 \odot 0 \odot 0 = 1 \) \( \Rightarrow \) the intersection point between line segment \( e = \langle v_1, v_1, v_1 \rangle \) and \( f_i \) plane is inside of \( f_i \).

**Table 2** The values of \( B_{e,e_i} \) and \( Y_{e,f_i} \) in Fig.1

| \( e_j \) | \( e_1 \) | \( e_2 \) | \( e_3 \) | \( e_4 \) | \( e_5 \) | \( e_6 \) | \( e_7 \) | \( e_8 \) | \( e_9 \) |
|---|---|---|---|---|---|---|---|---|---|
| \( B_{e,e_i} \) | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| \( Y_{e,f_i} \) | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |

Based on the above-mentioned two steps, the checking of the edge-face problem (denoted as \( \text{Inte}_{e,f_i} \)) can be determined by:

\[
\text{Inte}_{e,f_i} = X_{e,f_i} \land Y_{e,f_i}
\]

(6)
3 The Point-In-Polyhedron problem

3.1 Inclusion test for polyhedra with non-convex faces

Given a tested point \( v_{\text{test}} \), detecting whether \( v_{\text{test}} \) is inside a polyhedron \( p \) can be reduced to the problem of checking the number of faces pierced by the line segment \( e = v_{\text{test}} \rightarrow v_{\text{test}} \), where \( v_{\text{test}} \) is a point outside \( p \). Based on the Jordan Curve Theorem, the Point-In-Polyhedron problem denoted by \( \text{Insi}_{\text{ins},p} \) can be expressed as:

\[
\text{Insi}_{\text{ins},p} = \bigotimes_{e \in f} \text{Inte}_{e,f}
\]  

(7)

If \( \text{Insi}_{\text{ins},p} = 1 \), then \( v_{\text{test}} \) is inside of \( p \); otherwise, it is outside of \( p \).

The threshold-based ray-crossing (TBRC) algorithm for polyhedra with non-convex faces is presented in Fig. 2. If \( A_{\text{in},e} = A_{\text{in},f} \), then face \( f_i \) does not generate a node; otherwise, if \( A_{\text{in},e} \neq A_{\text{in},f} \) and \( Y_{e,f} = 1 \), then face \( f_i \) generates a node.

In Fig. 3, polyhedron \( p \) consists of eight faces, i.e., \( f_1 = (v_1, v_5, v_3); f_2 = (v_1, v_5, v_4); f_3 = (v_3, v_4, v_5); f_4 = (v_3, v_4, v_5), f_5 = (v_4, v_5, v_1); f_6 = (v_4, v_5, v_2); f_7 = (v_4, v_5, v_2); f_8 = (v_2, v_1, v_5). \) Consider the \( f = (v_{\text{test}}, v_{\text{out}}, f_i) \) plane supporting face \( f_i \), where line segment \( e = v_{\text{test}} \rightarrow v_{\text{test}} \) and point \( v_f \) (non-collinear with \( e \)) are on the \( f \) plane.

The intersection of \( e \) and a face can be the empty set, a point, or a line segment. The intersection of \( e \) and \( f_1, f_2, f_8 \) is \( v_i \), the intersection of \( e \) and \( f_3, f_6, f_7 \) are points belonging to edges; and the intersection of \( e \) and \( f_5 \) is a line segment. However, no intersection set, i.e., the empty set, exists between \( e \) and face \( f_4 \).

![Fig.3 An example of the inclusion test for a general polyhedron](image)

| Algorithm: For the Point-In-Polyhedron problem. |
|------------------------------------------------|
| Input: a test point \( v_{\text{test}} \), a polyhedron \( p \), two arbitrary points \( v_i \) and \( v_{\text{test}} \) \( (v_{\text{test}}, v_{\text{test}}) \) and \( f = (v_{\text{test}}, v_{\text{test}}, v_f) \). |
| Output: \( \text{Insi}_{\text{ins},p} \rightarrow (0, 1) \)/\( v_{\text{test}} \) is inside \( p \) if \( \text{Insi}_{\text{ins},p} \) is 1; otherwise, \( v_{\text{test}} \) is outside \( p \) if \( \text{Insi}_{\text{ins},p} \) is 0. |
| \( \text{Insi}_{\text{ins},p} = 0; \) |
| For each face \( f_i = (v_1, \ldots, v_r, v_f) \) of the polyhedron |
| \{ |
| if \( A_{\text{in},e} = A_{\text{in},f} \) continue; \( v_{\text{test}}, v_{\text{test}} \) is not on opposite sides of the \( f_i \) plane |
| \( Y_{e,f} \Rightarrow \text{Inte}_{e,f}; \) |
| \( \text{Insi}_{\text{ins},p} \rightarrow \text{Insi}_{\text{ins},p} \bigotimes Y_{e,f}; \) |
| } return \( \text{Insi}_{\text{ins},p} ; \) |

![Fig.2 The basic algorithm for polyhedra with non-convex faces](image)

| Table 3 The point in polyhedron test analysis of the example in Fig.2 |
|---------------------------------------------------------------|
| \( X_{e,f} \) \( Y_{e,f} \) \( Y_{e,f} \) \( Y_{e,f} \) \( Y_{e,f} \) \( Y_{e,f} \) |
|---------------------------------------------------------------|
| \( f_1 \) \( 1 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 1 \) \( < v_1, v_3 > \) \( 0 \) \( 1 \) |
| \( f_2 \) \( 1 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( 0 \) |
| \( f_3 \) \( 0 \) \( < v_1, v_3 > \) \( 1 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 1 \) \( 0 \) |
| \( f_4 \) \( 1 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( 0 \) |
| \( f_5 \) \( 1 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( 0 \) |
| \( f_6 \) \( 1 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( 0 \) |
| \( f_7 \) \( 1 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( 0 \) |
| \( f_8 \) \( 1 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( < v_1, v_3 > \) \( 0 \) \( 0 \) |
| \( \text{Insi}_{\text{ins},p} \) \( 0 \) \( 0 \) \( 0 \) \( 0 \) \( 0 \) \( 0 \) |
3.2 Inclusion test for polyhedra with convex faces

If face \( f_i \) is a convex polygon, the interference detection can be easily implemented by checking the following Boolean expression (Eq. (8)): If it is true, then edge \( e \) intersects \( f_i \), otherwise it does not.\(^{[26]}\)

\[
\text{Inte}_{e, f_i} = (A_{x, f_i} \land A_{y, f_i} \land B_{w, e_{i+1}}) \lor (A_{x, f_i} \land A_{y, f_i} \land B_{w, e_{i-1}})
\]

(8)

If \( e \land B \) or \( e \land B \) is true which denotes edges \( e_i \) traversed counter-clockwise or clockwise respectively, then the intersection point between \( e \) and \( f_i \) is located inside \( f_i \).

Similarly, the TBRC algorithm for polyhedra with convex faces also needs to deal with degenerate cases (i.e., \( e \) and \( e_i \) coplanar), and the edge-face problem has to be checked with Eq.(6) in such situation.

4 Efficient implementation

The two versions of TBRC algorithm mentioned above are robust. However, due to the calculation of the \( 4 \times 4 \) determinants for \( A_{x, f_i} \) and \( B_{w, e} \), it may turn out to be a bottleneck for many applications. It is therefore necessary to optimize the algorithms.

The first step is to check whether the ray \( (e) \) and the plane supporting \( f_i \) intersect, and then to check whether the intersect point is inside the face \( f_i \) in 2D space (i.e., \( xy \)-plane) after projection, i.e., first do a ray-plane test followed by a 2D test.

For each convex face, if degeneracies do not occur it is only necessary to calculate two \( 4 \times 4 \) determinants in Step 1 and \( n \times 3 \times 3 \) determinants in Step 2 in the worst case, where \( n \) is the number of edges of the convex face. In the best case, Step 2 would be solved by calculating two \( 3 \times 3 \) determinants. Therefore, an optimization can be realized by changing the order of Step 1 and Step 2, as a large percentage of faces not intersecting with the ray could be filtered in Step 2, thus avoiding calculating \( 4 \times 4 \) determinants for \( A_{x, e_{i+1}} \) and \( A_{y, e_{i-1}} \) in Step 1.

5 Experimental results and discussion

We made experiments on a PC with a 2.4GHz Intel Core 2 Quad PC and 2GB of RAM, and the algorithms were implemented in MS Visual C++ 6.0. Here TBRC algorithm for polyhedra with convex faces was compared with the classical tetrahedron-based method without pre-processing.\(^{[18]}\) whose source code is downloaded from http://jgt.akpeters.com/papers/SeguraEtAl05/. The tested models (Fig.4) in the range from 69451 to 1765388 triangles are downloaded from http://www.cc.gatech.edu/projects/large_models/and modified to be enclosed and manifold by using programs from http://www.cse.wustl.edu/~taoju/code/polymender.htm. Table 4 shows the characteristics of the models used. In the experiments, 100, 1000 and 10000 points randomly located in the minimal bounding box of the polyhedron were tested (the execution times (in seconds) computed by summing the CPU times for all tested points), and the results were compared (Table 5), and the corresponding reasons were analyzed.
6 Conclusion

In this paper, we presented a new technique to handle directly degeneracies in 3D, rather than to perturb input data or to avoid degeneracies. A method solving the edge-face problem is first developed based on definitions of two basic contacts: Type-A contact and Type-B contact, and based on which, the TBRC algorithm is then given. One of the major advantages of the TBRC algorithm is its capability when handling degeneracies without extra computational cost and storage. Our experimental results show that the TBRC algorithm optimized by the dimension reduction technique provides a very efficient and robust solution to the Point-In-Polyhedron problem, compared with the tetrahedron-based algorithm.

| Models         | Opened meshes | Enclosed and manifold meshes |
|----------------|---------------|-----------------------------|
|                | Vertices     | Triangles                   | Vertices | Triangles |
| Stanford bunny | 35947        | 69451                       | 45860    | 91716     |
| Skeleton hand  | 32732        | 65466                       | 66340    | 132780    |
| Dragon         | 437645       | 871414                      | 135224   | 270444    |
| Happy buddha   | 543652       | 1087716                     | 106152   | 212440    |
| Turbine blade  | 882954       | 1765388                     | 209704   | 419768    |

Table 5  Execution times (s) of each studied algorithm and acceleration ratios for our algorithm over the tetrahedron-based algorithm

| Models         | No. of tested points | 100 | 1000 | 10000 |
|----------------|-----------------------|-----|------|-------|
|                | Tetrahedron-based    | Our algorithm | Acceleration Ratios* (%) | Tetrahedron-based | Our algorithm | Acceleration Ratios* (%) | Tetrahedron-based | Our algorithm | Acceleration Ratios* (%) |
| Stanford bunny | 1.294                 | 1.047 | 23.591 | 13.342 | 10.923 | 22.145 | 132.520 | 107.311 | 23.491 |
| Skeleton hand  | 2.046                 | 1.547 | 32.255 | 20.344 | 15.570 | 30.661 | 202.507 | 155.465 | 30.258 |
| Happy buddha   | 3.062                 | 2.452 | 24.877 | 30.890 | 24.203 | 27.628 | 308.186 | 244.009 | 26.301 |
| Dragon         | 3.998                 | 3.125 | 27.936 | 40.286 | 31.420 | 27.628 | 400.038 | 314.936 | 27.021 |
| Turbine blade  | 6.174                 | 4.765 | 29.569 | 62.142 | 48.151 | 29.056 | 619.591 | 481.064 | 28.795 |

*Acceleration ratios are computed as (c – d)/d, where c and d refer to the execution times for the tetrahedron-based algorithm and our algorithm.

Table 6  Filter ratios for our algorithm (%)

| Models         | No. of tested points | 100 | 1000 | 10000 |
|----------------|----------------------|-----|------|-------|
|                | Filter ratio (%)     | 100 | 1000 | 10000 |
| Stanford bunny | 99.996               | 99.996 | 99.996 | 99.996 |
| Skeleton hand  | 99.751               | 99.750 | 99.750 | 99.750 |
| Happy buddha   | 99.998               | 99.998 | 99.998 | 99.998 |
| Dragon         | 99.997               | 99.997 | 99.997 | 99.997 |
| Turbine blade  | 99.993               | 99.993 | 99.993 | 99.993 |

*Filter ratio is computed as f/a, where f and a refer to the number of faces filtered in Step 2 and the number of all faces.

References

[1] Zhang S Q, Zhang J Y (2010) Theoretical analytics of stereographic projection on 3D-objects’ intersection predicate[J]. International Journal of Geographical Information Sciences, 24 (1): 25-46

[2] Shiode N (2001) 3D urban models: recent developments in the digital modeling of urban environments in three-dimensions [J]. GeoJournal, 52(3):263-269

[3] Coors V (2003) 3D-GIS in networking environments[J]. Computers, Environment and Urban Systems, 27(4): 345-357

[4] Kwan M P, Lee J (2004) Geovisualization of human activity patterns using 3D GIS: a timegeographical approach[M]/Goodchild M F, Janelle D G (Eds.). Spatially integrated social science. New York: Oxford University Press

[5] Lee J, Kwan M P (2005) A combinatorial data model for representing topological relations among 3D geographical features in micro-spatial environments[J]. International Journal of Geographical Information Science, 19(10): 1039-1056

[6] Stoter J E (2002) 3D cadastres, state of the art: from 2D parcels to 3D registrations[J]. GIM International, the World Magazine for Geomatics, 16(2)

[7] Zlatanova S (2002) Advances in 3D GIS[J]. Quarterly Review of Disegno Digitale e Design, 1(4): 24-29
[8] Zlatanova S, Bandrova T (1998) User requirements for the third dimensionality[C]. E-mail Seminar of Cartography 1998: Maps of the Future, Sofia, Bulgaria
[9] Lee J (2008) Spatial data analysis in 3D GIS[M]// Oosterom P, Zlatanova S, Penninga F, et al. (Eds.). Advances in 3D Geoinformation System: Lecture Notes in Geoinformation and Cartography. New York: Springer
[10] Guo W, Zhan P, Chen J (1998) Topological data model for 3D GIS[J]. IAPRS, 32(4): 657-661
[11] Ramos F (2002) A multi-level approach for 3D modelling in geographical information systems[J]. International Archives of Photogrammetry and Remote Sensing, 34(4): 43-47
[12] Segura R, Feito F R, Ruiz de Miras J (2005) An efficient point classification algorithm for triangle meshes[J]. Journal of Graphics, GPU, & Game Tools, 10(3):27-35
[13] Wang W C, Li J, Sun H Q, et al. (2008) Layer-based representation of polyhedrons for point containment tests[J]. IEEE Trans. Vis. Comput. Graph, 14(1): 73-83
[14] Verbrue E (2005) Towards a 3D feature overlay through a tetrahedral mesh data structure[J]. Cartography and Geographic Information Science, 32(4): 303-314
[15] Jiménez P, Thomas F, Torras C (2001) 3D collision detection: a survey[J]. Computers & Graphics, 25(2): 269-285
[16] Ogayar C J, Segura R J, Feito F R (2005) Point in solid strategies[J]. Computers and Graphics, 29(4): 616-624
[17] Meagher D (1982) Geometric modelling using octree encoding[J]. Computer Graphics and Image Processing, 19(2): 129-147
[18] Feito F R, Torres J C (1997) Inclusion test for general polyhedra[J]. Computers and Graphics, 21(1): 23-30
[19] Carvajal P C P, Cavalcanti P R (1995) Point in polyhedron testing using spherical polygons[M]// Paeth A W (Eds.). Graphics Gems V. New York: Academic Press
[20] Möller T, Haines E (2002) Real-time rendering (2nd Edition)[M]. Boca Raton: A K Peters/ CRC Press
[21] O’Rourke J (1998) Computational Geometry in C (2 edition)[M]. Cambridge: Academic Press
[22] Emiris I, Canny J (1992) An efficient approach to removing geometric degeneracies[C]. Proceedings of the 8th Annual symposium on Computational geometry, Berlin, Germany
[23] Edelsbrunner H, Mücke E P (1990) Simulation of simplicity: a technique to cope with degeneracies in geometric algorithms[J]. ACM Transactions on Graphics, 9(1): 66-104
[24] Stewart J A (1994) Local robustness and its application to polyhedral intersection[J]. International Journal of Computational Geometry & Applications, 4(1): 87-118
[25] Thomas F, Torras C (2002) A projectively invariant intersection test for polyhedra[J]. The Visual Computer, 18(7): 405-414
[26] Canny J (1987) The complexity of robot motion planning[M]. Cambridge MA: MIT Press