Precise Hausdorff Distance Computation between Polygonal Meshes

Michael Bartoń\textsuperscript{a}, Iddo Hanniel\textsuperscript{a,b}, Gershon Elber\textsuperscript{a}, Myung-Soo Kim\textsuperscript{c}

\textsuperscript{a}Technion – Israel Institute of Technology, Haifa 32000, Israel
\textsuperscript{b}SolidWorks Corporation, 300 Baker Avenue, Concord, MA 01742 USA
\textsuperscript{c}Seoul National University, Seoul 151-744, Korea

Abstract
We present an exact algorithm for computing the precise Hausdorff distance between two general polyhedra represented as triangular meshes. The locus of candidate points, events where the Hausdorff distance may occur, is fully classified. These events include simple cases where foot points of vertices are examined as well as more complicated cases where extreme distance evaluation is needed on the intersection curve of one mesh with the self-bisector of the other mesh. The self-bisectors for pairs of primitives (i.e. vertex, edge, or face) are analytically constructed and they are intersected with the other mesh, yielding a set of conic segments. For each conic segment, the distance functions to all primitives are constructed and the maximum value of their low envelope function may correspond to a candidate point for the Hausdorff distance. The algorithm is fully implemented and several experimental results are also presented.

Key words: Hausdorff distance, polygonal mesh, geometric algorithm, bisector surface, self-bisector, low envelope

1. Introduction

Measuring the distance between different geometric objects is often needed as a first step for other geometric computations such as collision detection, robot motion planning and haptic rendering [1, 2, 3]. Thus the development of efficient algorithms for distance computation is very important for improving the performance of related geometric problems. Among many different distance measures, the Euclidean distance and recently the Hausdorff distance have attracted considerable research attention in geometric modeling, computational geometry, and computer graphics [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15].

An extensive family of applications for the Hausdorff distance may also be found in visualisation, namely in the case of surface remeshing [16, 17, 18]. Typically, having a dense starting mesh, its simplification is required to be within a certain distance measure. The one-sided Hausdorff distance from $A$ to $B$, $h(A, B)$, measuring the maximum deviation of an object $A$ from the other
object $B$, is defined as
\[
h(A, B) = \max_{a \in A} d(a, B),
\]
where $d(a, B) = \min_{b \in B} \|a - b\|$ is the minimum Euclidean distance from $a$ to $B$. Since the one-sided distance is non-symmetric, the Hausdorff distance $H(A, B)$ is defined as
\[
H(A, B) = \max \{ h(A, B), h(B, A) \},
\]
which introduces all metric properties and the Hausdorff distance measures the shape similarity between two geometric shapes [9].

For planar convex polygons, Atallah [12] developed a linear-time algorithm for computing the Hausdorff distance. Llanas [13] considered two approximate algorithms for non-convex polytopes. Cignoni et al. [14] reported an efficient algorithm for approximating the Hausdorff distance between a non-convex polygonal mesh and a set of point samples on a nearby polygonal mesh of similar shape. In a recent work, Tang et al. [15] presented a real-time algorithm for approximating the Hausdorff distance between two triangular meshes within a given error bound.

There are also some previous results on the more general case for freeform shapes. Jüttler [19] presented a method to estimate bounds on the Hausdorff distance between two freeform/implicit curves. Alt and Scharf [10, 11] developed an exact algorithm for the Hausdorff distance between planar rational curves. In a recent work, Elber and Grandine [20] extended this result to a more general problem for rational curves/surfaces. The performance of the proposed algorithm is quite efficient for the case of planar/space curves. However, for the Hausdorff distance between a freeform surface and a curve or another surface, the algorithm is too slow.

In this work, we consider a slightly simpler problem for two polygonal meshes and present an exact algorithm for the Hausdorff distance computation. From a theoretical point of view (in the sense of computational geometry), Alt et al. [21] also considered this problem. They first explained a simple approach based on the lower envelope of squared distance function surfaces over each triangle. The worst case time complexity is $O(n^5)$, where $n$ is the number of triangles. (Employing a very theoretical machinery, they also derived a randomized algorithm of $O(n^{3+\epsilon})$ expected time complexity, for any $\epsilon > 0$. However, their randomized algorithm has never been implemented due to its non-trivial intricacy.) Even their simple approach has never been implemented. Compared with lower envelopes of surfaces, it is considerably easier to implement an algorithm that is based on a low envelope of curves. In this paper, we present such an algorithm with its worst-case time complexity $O(n^4 \log n)$ with a complete implementation.

Without loss of generality, we assume that
\[
h(A, B) > h(B, A),
\]
and the Hausdorff distance $H(A, B)$ is realized at two points $a_0 \in A$ and $b_0 \in B$.
Figure 1: Triangle $T_A$ of mesh $A$ where the one-sided distance $h(A, B)$ is attained.
(a) The footpoint $b_0 \in B$ is either unique (Case 1) or (b) there exist two footpoints (Case 2). In the latter case, $a_0$ lies on the intersection curve (red) between $T_A$ and (a part of) the self-bisector of $B$.

such that

\[
H(A, B) = \max_{a \in A} d(a, B) = d(a_0, B) = \min_{b \in B} \|a_0 - b\| = \|a_0 - b_0\|. \tag{1}
\]

Note that the point $b_0$ is the footpoint of $a_0$ to the other object $B$. There are two cases to consider: (1) the footpoint $b_0$ is unique or (2) there are multiple (typically two) footpoints $b_0, b'_0 \in B$ (i.e., the point $a_0$ is on the self-bisector of $B$), see Fig. 1.

An important advantage in considering the special case of two polyhedra is that, in the first case, the Hausdorff distance is realized at a mesh vertex $a_0$ as the minimum distance to the other polygonal mesh $B$. (There are some degenerate cases where $a_0$ can be located in an edge/face interior; but these cases can easily be reduced to a case where $a_0 \in A$ is a vertex or $a_0 \in A$ is located on a self-bisector of the other mesh as discussed in Section 2.) Moreover, in the second case, the point $a_0$ of one mesh is on a self-bisector surface of the other mesh $B$. Since the self-bisector surfaces of a polygonal mesh are simple quadratic surfaces, the problem essentially reduces to solving quadratic equations and searching some extreme distance points on conic curve segments. Unlike [14, 15], the presented method does not require any point sampling or subdivision of input triangles; and hence the precise\textsuperscript{1} Hausdorff distance is computed.

The rest of the paper is organized as follows. Section 2 fully classifies the locus of candidate points, events where the Hausdorff distance may occur. Con-

\textsuperscript{1}By precise we mean up to floating point numbers.
sequently, in Section 3, we develop the algorithm that inspects those candidate locations and computes the precise Hausdorff distance. The complexity of the proposed algorithm is analyzed in Section 4. Several examples are given in Section 5 and finally the paper is concluded in Section 6.

2. Candidate points

Let $A$ and $B$ be two triangular meshes and let $h(A,B)$ be the one-sided Hausdorff distance from $A$ to $B$. In this section, we introduce a set of candidate points, a locus of points in $A$ where $h(A,B)$ may be attained, and we prove this set is sufficiently described by one of two cases:

- **Case 1**: $a_0$ is a vertex of $A$ and the maximum of the Euclidean distance is attained between $a_0$ and an object - face, edge or vertex - of $B$.
- **Case 2**: $a_0$ is a point on the intersection curve between a triangle of $A$ and a self-bisector of $B$. This intersection curve is (a segment of) a conic section and the point $a_0$ maximizes the distance between this conic section and the mesh $B$.

Before we come to the proof, we need to recall some basic notions from the Voronoi theory [22].

2.1. Preliminaries

**Definition 1.** Given a set of $(n+1)$ objects $O^0, O^1, \ldots, O^n$ in $\mathbb{R}^3$, the Voronoi cell (VC) of an object $O^i$, denoted as a base object, is the set of all points in $\mathbb{R}^3$ closer to $O^i$ than to $O^j$, $\forall i \neq j$. The Voronoi Diagram (VD) is then the union of the Voronoi cells of all $(n+1)$ objects.

**Definition 2.** The locus of points that are equidistant from $O^i$ and $O^j$, for some $j \neq i$, is called (the $O^iO^j$) bisector.

In general, bisectors in $\mathbb{R}^3$ are surfaces. For some special mutual limit positions of objects $O^i$ and $O^j$, the bisector may degenerate either to a curve, or to the whole $\mathbb{R}^3$ (if $O^i \equiv O^j$). Note that the boundaries of the Voronoi cells consist of bisector surfaces.

In the case of simple objects like polygonal meshes, the geometric entities dealt with are points, lines and planes in $\mathbb{R}^3$. Therefore, all the bisectors between these entities are quadric surfaces and thus their intersections with a plane are conic section curves in $\mathbb{R}^3$. The fact that these bisectors are quadrics is obvious from the prescription of the (squared) distance function, which measures the distance from a point, a line or a plane, respectively (hereafter bold letters denote vector values):

- **The squared distance function from a point**:
  \[
  d^2 = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2, \quad (2)
  \]
where \( p_0 = (x_0, y_0, z_0) \) is the point from which we compute the distance, and \( d \) is the computed distance of the Euclidean point \( p = (x, y, z) \in \mathbb{R}^3 \) from \( p_0 \).

- **The squared distance function from a plane:**

\[
d^2 = \frac{\alpha x + \beta y + \gamma z + \delta}{\alpha^2 + \beta^2 + \gamma^2},
\]

where \((\alpha, \beta, \gamma, \delta)\) are the plane’s coefficients.

- **The squared distance function from a line:**

\[
d^2 = \langle p - c_0, p - c_0 \rangle - \frac{\langle p - c_0, d_0 \rangle^2}{\langle d_0, d_0 \rangle},
\]

where \( c_0 + td_0 \) is the parametric line equation and \( d \) is the distance of the point \( p \) from the line.

**Lemma 1.** The bisector surfaces between points, lines and planes in \( \mathbb{R}^3 \) are quadric surfaces.

**Proof.** By definition, the bisector is the locus of all equidistant points. Thus, the result of equating two squared distance functions and eliminating \( d^2 \) gives the implicit representation of the bisector. Since both sides of the equation hold only, at most, degree-2 polynomials (in \( x, y, z \)), a (at most) degree-2 implicit surface is obtained. \( \square \)

Note that planes are degenerate quadric surfaces and indeed the plane-plane and point-point bisectors are, in fact, planes. Hereafter, we denote by a mesh-bisector, the bisectors between points, lines and/or planes.

**Corollary 1.** The intersection of a plane and a mesh-bisector is a conic section curve.

**Proof.** Since the bisector surface is a quadric, its intersection with a plane is a conic section curve. \( \square \)

**Remark 1.** In Section 3, we will measure the squared distance \( d^2(t) \) from a conic curve segment \( C_{ij}^k(t) \) (the intersection between a triangle \( T_A^k \) and an \( O_B^i \) bisector quadric) to an object \( O_B^i \) (a point, a line, or a plane). The squared distance function is computed by replacing the point \( p \) of (2)–(4) by a moving point \( p(t) = C_{ij}^k(t) = (x(t), y(t), z(t)) \) along the conic section curve. Since the conic curve can be represented as a quadratic rational curve, the corresponding squared functions will be quartic rational functions in \( t \).
2.2. Sufficient Subset of Candidate Points

In this subsection, we show that the one-sided Hausdorff distance can be computed by only checking two sets of candidate points; either at vertices of $A$ (Case 1) or on the intersections of $A$ with the self-bisector surfaces of $B$ (Case 2). An examination of these two cases detects some points $a_0 \in A$, $b_0 \in B$ such that $h(A, B) = d(a_0, b_0)$. We show that the examination of the candidate points described by these two cases is sufficient to compute the precise Hausdorff distance.

Example 1. Points defined in Cases A and B form a set of candidates at each of which the distance needs to be calculated. Note that the set of points where the Hausdorff distance may be attained can be an infinite set. Consider a mesh $A$ consisting of only one triangle $T_A = A$ which is completely contained in the $VC$ of an edge $O_{iB} \in B$ such that the farthest edge of $T_A$ is parallel to $O_{iB}$, as depicted in Fig. 2. Then, every point of that edge is a candidate point. In this particular example, the Hausdorff distance is revealed at vertex $a_0'$ (Case 1).

Theorem 1. The one-sided Hausdorff distance $h(A, B)$ from $A$ to $B$ can be attained either at a vertex of $A$ (Case 1) or on the intersection of $A$ with a self-bisector surface of $B$ (Case 2).

Proof: Consider the $VD$ of a mesh $B$. By definition, the $VC$ of an object $O_{iB} \subset B$ (face, edge or vertex) is the region in $\mathbb{R}^3$ where all points are closer to $O_{iB}$ than to any other object in $B$. Let us denote by $a_0 \in A$, the point where $h(A, B)$ is attained and let $a_0 \in VC$ of $O_{iB}$ for some $i$. Since the $VC$s are disjoint in their interior and their union covers $\mathbb{R}^3$, $a_0$ lies
• either strictly in the interior of the VC of $O'_B$, \\
• or on its boundary.

If $a_0$ lies on the boundary, $a_0$ lies on a self-bisector surface of $B$, since the boundary of any VC consists of bisector surfaces between two objects of $B$ (Case 2).

What remains to prove is that, if $a_0$ lies in the interior of the VC of some $O'_B$, then $a_0$ is

(i) directly a vertex of $A$, or if not then

(ii) there exists a vertex $a'_0 \in A$ lying in the interior of VC such that $h(A, B) = d(a_0, B) = d(a'_0, B)$, or if not then

(iii) there exists $a''_0 \in A$ lying on the boundary of the VC such that $h(A, B) = d(a_0, B) = d(a''_0, B)$.

Assume $a_0$ is an interior point of the VC of $O'_B$. Since $a_0$ is the point where the one-sided Hausdorff distance $h(A, B)$ is attained, $d(a_0, B) \geq d(a, B)$ for any $a \in A$. Depending on the type of $O'_B$ (a vertex, an edge, a face), three cases need to be considered:

• **Vertex:** $a_0$ is on a sphere of radius $h(A, B)$ centered at a vertex $O'_B$, and all the other points of $A \cap VC$ are contained in this sphere. In particular, the triangle of $A$ to which $a_0$ belongs is contained inside the sphere and this can only happen if $a_0$ is a vertex of the triangle; case (i) above.

• **Edge:** $a_0$ lies on a cylinder of radius $h(A, B)$ whose axis is along an edge $O'_B$, and all other points of $A \cap VC$ are contained in this cylinder. In particular, the triangle $T_A$ to which $a_0$ belongs is contained in the cylinder, see Fig. 2. Assume that $a_0$ is not a vertex of the triangle. Since it is on the cylinder and no point of $T_A$ is outside the cylinder, $a_0$ must be on a triangle edge of $A$ that is parallel to the cylinder axis (and there are an infinite number of points on the triangle edge with the same distance from $O'_B$). This edge is either entirely contained in VC (which immediately gives (ii)) or if not then it intersects a boundary of VC at some point $a''_0$ $(\Rightarrow$ (iii)).

• **Face:** In a similar manner, $a_0$ is on a plane parallel to the face plane $O'_B$, and offset by $h(A, B)$, and all the other points of $A \cap VC$ are contained between these two planes. If $a_0$ is not a vertex of a triangle then it is either a point on an edge of the triangle and the edge must be parallel to the plane of the triangle $O'_B$, or it is an inner triangle point and the whole triangle $T_A$ must be parallel to the plane of the face $O'_B$. In both cases, a similar argument to the edge case shows that either there is a vertex inside the VC $(\Rightarrow$ (ii)), or the distance $h(A, B)$ is attained at an intersection of the triangle with the VC boundary $(\Rightarrow$ (iii)).
Thus, in order to compute the one-sided Hausdorff distance \( h(A, B) \), it is sufficient to check only the point sets defined in Cases A and B since the union of these two sets always contains a candidate point at which the sought Hausdorff distance is attained. Note that there may exist some (other) \( a'_0 \in A \) and \( b'_0 \in B \) such that \( h(A, B) = d(a_0, b_0) = d(a'_0, b'_0) \) and the point \( a'_0 \) belongs to none of the sets described by Cases A and B, see Example 1.

3. Algorithm

In this section, we present an algorithm for computing the precise Hausdorff distance between two triangular meshes. Moreover, in order to reduce the computational cost, we also introduce some local optimizations.

Remark 2. The computation of the Voronoi diagrams for meshes is difficult in practice. Instead of the precise construction of \( VD \), we handle only the bisector surfaces \( B^i_B \), the mesh-bisectors for pairs of objects \( O^i_B, O^j_B \). Because only some
sub-part of $B_{ij}^B$ contributes to the $VD$, the bounding panels of objects $O_{ij}^B$, $O_{ij}^I$ are considered:

**Definition 3.** The bounding panel of edge $O_{ij}^B$ is a subset of $R^3$ bounded by two planes that pass through the endpoints of $O_{ij}^B$ while also being orthogonal to the edge. The bounding panel of face $O_{ij}^I$ is a locus of all points $x \in R^3$ such that $x_\beta \in O_{ij}^I$, where $x_\beta$ denotes the orthogonal projection of $x$ to $\beta$, the plane of face $O_{ij}^I$.

The bounding panel of $O_{ij}^B \in B$ determines a part of $R^3$, where the particular object may become a Hausdorff distance candidate. Apparently, this easy-to-compute bounding region is a superset of the $VC$ of $O_{ij}^B$, see Fig. 3(c) and (d). Bounding panels are constructed in order to purge those objects of $A$, where the one-sided Hausdorff distance can not be attained. Note that bounding panels are, as well as $VC$, open sets and their boundaries are the corresponding planes.

Based on observations discussed in Section 2.2, we formulate an algorithm, which checks the set of candidate locations (Cases A and B) and returns the precise Hausdorff distance. We now describe several elementary steps, building blocks of the algorithm, in more detail:

- The (minimum) distance between a point $a$ and a triangle $T_B \subset B$, is accomplished by determining the Voronoi cell of the object $O_{ij}^B \subset T_B$, which contains $a$, see Fig. 3(b). The sought distance is then $d(a,T_B) = d(a,O_{ij}^B)$. Consequently, Algorithm 1 summarizes the point–mesh distance computation.

- The intersection of an implicit quadric $B_{ij}^B = 0$ (the mesh-bisector of objects $O_{ij}^B$ and $O_{ij}^I$) in $R^3$ with the plane of a triangle $T_A = V_A^1V_A^2V_A^3 \subset A$, $P(u,v) = V_A^1 + u(V_A^2 - V_A^1) + v(V_A^3 - V_A^1)$, $u,v \in R$, $\text{(5)}$

![Figure 4: The uv-space of triangle $V_A^1V_A^2V_A^3$ is intersected with mesh-bisector $B_{ij}^B$ resulting in implicit conic section $C(u,v) = 0$ (red). This curve is parameterized and trimmed by the triangle, yielding curve $C_{ij}^B(t)$ (superscript $k$ corresponds to the numbering of triangles of mesh $A$).](image-url)
Algorithm 1

{ Minimal point–mesh distance }

1: \textbf{INPUT:} Triangular mesh $B$ consisting of $n$ triangles and point $a$
2: \textbf{for} every triangular face $T_B^i$ of $B$ \textbf{do}
3: $d_i = d(a, T_B^i)$, see Fig. 3(b);
4: \textbf{end for}
5: $d(a, B) = \min_{i=1,\ldots,n} d_i$;
6: \textbf{OUTPUT:} $d(a, B)$.

is an implicit conic section $C(u, v) = 0$ in the parametric $(u, v)$-plane. This curve is parameterized according to [23] and trimmed by the triangle $T_A$, see Fig. 4. Note that multiple branches may be obtained and each branch can be represented as a quadratic rational plane curve.

- **Curve-mesh (minimal) distance:** Consider a quadratic rational curve $C_{ij}^k(t)$, $t \in [t_{min}, t_{max}]$, a branch of the intersection curve of a mesh-bisector $B_B^i$ with a triangle $T_A^k \subset A$. To detect a candidate point $C_{ij}^k(t^*)$ (Case 2) on the curve $C_{ij}^k(t)$, a parameter $t^* \in [t_{min}, t_{max}]$, which maximizes $d^2(C_{ij}^k(t), B)$, is sought.

For the computation of $d^2(C_{ij}^k(t), B)$, we consider the lower envelope $LE_{ij}^k(t)$ of the set of squared distance functions $d^2(C_{ij}^k(t), O_l^B)$ between (a trimmed sub-segment of) the curve $C_{ij}^k(t)$ and all objects $O_l^B \subset B$, whose bounding panels have non-empty intersection with $C_{ij}^k$, see Fig. 5. Then, the candidate parameter $t^*$ is a parameter value, where

1. the maximum of $LE_{ij}^k(t)$ is attained:
   \[
   LE_{ij}^k(t^*) \geq LE_{ij}^k(t), \quad \forall t \in [t_{min}, t_{max}] \tag{6}
   \]
   and
2. $t^*$ lies in the part of $LE_{ij}^k(t)$ which corresponds to the squared distance function from the curve $C_{ij}^k(t)$ to the objects $O_{l_B}^l$ and $O_{l_B}^r$. (That is, the objects of $B$ which are the closest to the candidate point $C_{ij}^k(t^*)$ must be $O_{l_B}^l$ and $O_{l_B}^r$.) Otherwise, the candidate point is not relevant since it is closer to other objects $O_{l_B}^l$ than to $O_{l_B}^r$ and $O_{l_B}^r$, see Fig. 5; consequently the curve $C_{ij}^k(t)$ can be ignored in the remaining computation.

- **Local optimization using the bounding panel of an object:** Consider a mesh-bisector’s triangle intersection curve $C_{ij}^k(t)$, $t \in [t_{min}, t_{max}]$, and an object $O_{l_B}^l \subset B$. $O_{l_B}^l$ may contribute to the lower envelope $LE_{ij}^k(t)$ only if some part of $C_{ij}^k(t)$ lies inside the VC of $O_{l_B}^l$ (and the bounding panel bounds VC, see Def. 3). Hence, consider a restriction of $C_{ij}^k(t)$, if any, to the bounding panel of $O_{l_B}^l$. Note that only this sub-segment of
Remark 3. Note that the majority of the objects of $B$ do not contribute to the lower envelope. Only those objects whose VC's contain (a part of) the intersection curve $C_{ij}^k(t)$ may contain footpoints from the curve. The curve trimming process described above is introduced in order to reduce the number of objects, and consequently, the number of square distance functions for which the lower envelope is computed.

Algorithm 2 { One-sided Hausdorff distance between two triangular meshes}

1: INPUT: Two triangular meshes $A$ and $B$
2: $d := 0$
3: for every vertex $a$ of $A$ do
4: \[ \text{candidate}(a) = d(a, B); \] \hspace{1cm} (Algorithm 1, Case 1)
5: if candidate($a$) > $d$; then
6: \[ d \leftarrow \text{candidate}; \]
7: end if
8: end for
9: for every mesh-bisector $B^{ij}_B$ of $B$ do
10: for every triangular face $T^k_A$ of $A$ do
11: \[ C_{ij}^k(t) = B^{ij}_B \cap T^k_A; \] \hspace{1cm} (Case 2)
12: \[ \text{candidate}(C_{ij}^k) = \max_{t \in [t_{\min}, t_{\max}]} LE_{ij}^k(t); \]
13: if candidate($C_{ij}^k$) is relevant and candidate($C_{ij}^k$) > $d^2$ then
14: \[ d^2 \leftarrow \text{candidate}(C_{ij}^k); \]
15: end if
16: end for
17: end for
18: OUTPUT: One-sided Hausdorff distance $h(A, B) = d$.

Algorithm 3 { Hausdorff distance between two triangular meshes}

1: INPUT: Two triangular meshes $A$ and $B$
2: \[ H(A, B) := \max(h(A, B), h(B, A)) \] \hspace{1cm} (Algorithm 2)
3: OUTPUT: Hausdorff distance $H(A, B)$.

Summing up all the previous building blocks, the one-sided Hausdorff distance computation method is formulated in Algorithm 2. Finally, for the sake of formal completeness, the Hausdorff distance computation is given in Algorithm 3.
Figure 5: Hausdorff distance candidate point on a bisector-triangle intersection curve: (a) Mesh B and one triangle $T_A^k$ of mesh A. (b) The implicit bisector $B_{ij}^B$ between face $O_i^B$ and vertex $O_j^B$ of mesh B is intersected with $T_A^k$ resulting in a conic section segment (light grey). (c) Parameter $t^\star$ of conic segment $C_{ij}^k(t)$ which maximizes $d(C_{ij}^k(t^\star), B)$ is sought. (d) Lower envelope $LE_{ij}^k(t)$ (thick black) of a set of quartic rational functions is constructed. These functions measure the squared distances between $C_{ij}^k(t)$ and objects of B, whose bounding panels have non-empty intersection with $C_{ij}^k(t)$. (e) Candidate parametric value $t^\star$ defines candidate point $C_{ij}^k(t^\star)$. Since the closest object of B (edge $O_l^B$) is neither $O_i^B$ nor $O_j^B$, candidate $C_{ij}^k(t^\star)$ is irrelevant.
4. Complexity Analysis

In this section, we examine the complexity of Algorithm 2 (and consequently of Algorithm 3).

Let \( m \) be the number of objects (vertex, edge, face) of \( A \) and let \( n \) be the number of objects in \( B \). In line 4 of Algorithm 2, \( \mathcal{O}(m) \) vertex-mesh distance computations are needed, since every vertex-mesh distance computation takes \( \mathcal{O}(n) \), the total complexity of Case 1 is \( \mathcal{O}(mn) \). In lines 9 and 10, every mesh-bisector is intersected with every triangle of \( A \) which gives \( \mathcal{O}(mn^2) \) intersection curves. Above the parametric domain of each of these curves, the lower envelope of \((\text{at most})\ n \) squared distance functions is constructed. The lower envelope of a set of \( n \) functions are \((n, s)\)-Davenport-Schinzel sequences [24], where \( s \) is the maximal number of intersections between any pair of functions (a small constant in our case). The complexity of an \((n, s)\)-Davenport-Schinzel, is denoted by \( \lambda_s(n) \) and is an almost linear function\(^2\). Therefore, the computation of the lower envelope of \( n \) squared distance functions over the parametric domain takes \( \mathcal{O}(\lambda_s(n) \log n) \) and finding its maximum requires \( \mathcal{O}(\lambda_s(n)) \) calculations. Thus, the total complexity is

\[
C = \mathcal{O}(mn) + \mathcal{O}(mn^2)\mathcal{O}(\lambda_s(n) \log n + \lambda_s(n)) = \mathcal{O}(mn^2\lambda_s(n) \log n). \tag{7}
\]

For any practical purposes \( \lambda_s(n) \) can be considered \( \mathcal{O}(n) \). Therefore, the total complexity of Algorithm 2 can be considered to be \( \mathcal{O}(mn^3 \log n) \). For the common case when \( m = \mathcal{O}(n) \) we have the complexity \( \mathcal{O}(n^4 \log n) \).

5. Examples

In this section we present several examples of computing the Hausdorff distance between two polygonal meshes. All examples were created using the GuRit GUI interface of the Irit solid modeling environment \(^3\), where the presented algorithm is implemented as a shared library.

All presented examples have been tested on a PC with an Intel(R) Pentium(R) CPU (2.8GHz), 1GB of RAM. Table 1 shows the parameters and timings for meshes shown in Figs. 6–11.

Examples of Figs. 7 and 8 show two relatively simple configurations, where the one-sided distance \( h(A, B) \) is attained in the interior of a triangle of \( A \). This event can occur, for instance, when the entire bisector-triangle intersection curve is fully contained in the interior of a triangle (Fig. 8) as well as when

---

\(^2\lambda_s(n)\) depends on \( \alpha(n) \) — the inverse Ackerman function, which is an extremely slow-growing function that appears frequently in algorithms. If \( n \) is less than or equal to an exponential tower of 65536 2’s, then \( \alpha(n) \leq 4 \) (see [25], for example)

\(^3\)www.cs.technion.ac.il/~irit
the extremal distance value corresponds to the footpoints (at equal distance) to three different objects of mesh B (Fig. 7).

In general, as observed in Table 1, the candidate footpoints occur more frequently at the boundary of the bisector-triangle intersection curve, see Figs. 6
Figure 8: The bisector-triangle intersection curve (red) is at equal distance from vertex \( O_i \) and face \( O_j \). The maximal candidate \( a_0 \) is "purely" determined by the distance function on the bisector curve: in contrast to Fig. 7, there is no other object \( O_k \in B \) such that \( d(a_0, O_i) = d(a_0, O_j) = d(a_0, O_k) \). A side view of the same situation is shown in (b).

Figure 9: Both one-sided Hausdorff distances are attained along the bisector-triangle intersection curves (red). Each line is the intersection of a triangle with the edge-edge \( (h(A, B)) \) or the face-face \( (h(B, A)) \) bisectors. The top view of the same example is shown in (b).

and 10.

Table 1 also reports the number of relevant and irrelevant candidates arisen
Figure 10: The one-sided Hausdorff distance \( h(B, A) \) is attained at the boundary point \( b_0 \) of the bisector-triangle intersection curve (red), whereas \( h(A, B) \) occurs at a vertex \( a_0 \).

Figure 11: Time demanding example, where both one-sided Hausdorff distances are attained at vertices.

from the bisector-mesh intersection curves. These experimental data indicate the dominance of irrelevant candidates in the majority of practical situations. The irrelevancy of a candidate means that the corresponding bisector-mesh intersection curve thus constructed eventually turned out to be redundant after some later distance comparison with other objects of \( B \). Note that in the current implementation all bisectors have been considered. Moreover, the computation of the candidate points (for the Hausdorff distance) on the bisector-mesh intersection curves is the most time-consuming part of the whole algorithm. Hence, an optimization procedure such as Tang et al. [15] for efficiently culling redun-
Table 1: $T_A$, $T_B$ are the numbers of triangles, $B_A$ vs. $T_A$ reports the number of bisector-triangle intersection curves and consequently the number of candidates on these bisector curves. The candidates are further classified as relevant and irrelevant based on the discussion in Section 3. Analogously for $h(A,B)$. Moreover, for the bisector-triangle curves, we report the numbers of cases where the maximum of lower envelope appear at the boundary of curve domain or in the interior.

<table>
<thead>
<tr>
<th></th>
<th>$T_A$ vs. $T_B$</th>
<th>$B_B$ vs. $T_A$</th>
<th>Max of $LE$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T_A$</td>
<td>$T_B$</td>
<td>Rel.</td>
<td>Irrel.</td>
</tr>
<tr>
<td>Fig. 7</td>
<td>(1,3)</td>
<td>0</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>Fig. 8</td>
<td>(1,3)</td>
<td>0</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Fig. 9</td>
<td>(4,4)</td>
<td>10</td>
<td>190</td>
<td>13</td>
</tr>
<tr>
<td>Fig. 6</td>
<td>(16,17)</td>
<td>53</td>
<td>6,544</td>
<td>173</td>
</tr>
<tr>
<td>Fig. 10</td>
<td>(18,34)</td>
<td>305</td>
<td>32,228</td>
<td>223</td>
</tr>
<tr>
<td>Fig. 11</td>
<td>(67,49)</td>
<td>1,029</td>
<td>119,995</td>
<td>1,024</td>
</tr>
</tbody>
</table>

dant triangles at an early stage of the computation seems to be highly promising in reducing the number of objects whose bisectors may contain relevant candidates and consequently to significantly reduce the computation time of the entire process.

6. Conclusion

We have presented an algorithm for computing the precise Hausdorff distance between two general polyhedra represented as triangular meshes. As observed in Sections 4 and 5, due to the high complexity, the computational timing exceeds reasonable range even for meshes consisting of several tens of triangles (hundred objects). Note that the current implementation computes the mesh-bisectors for all pairs of objects, which involves highly redundant computations. In order to improve the asymptotic timing behavior of the method, some heuristic which purges away redundant objects is needed. Hence, as a future research, our algorithm could be extended to use an efficient culling technique in a preprocessing stage, such as the algorithm of Tang et al. [15], which is expected to significantly reduce the number of candidate objects and hopefully get closer to a real-time result.

Acknowledgment

This work was supported in part by the Israeli Ministry of Science Grant No. 3-4642, in part by the Israel Science Foundation (grant No. 346/07), and also in part by KICOS through the Korean-Israeli Binational Research Grant (K20717000006) provided by MEST in 2007.
References


