DOI: 10.1111/cgf.14395

COMPUTER GRAPHICS *forum*
Volume 0 (2021), number 0 pp. 1–11

Economic Upper Bound Estimation in Hausdorff Distance Computation for Triangle Meshes

Yicun Zheng, Haoran Sun, Xinguo Liu, Hujun Bao and Jin Huang

State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China
hj@cad.zju.edu.cn

Abstract

The Hausdorff distance is one of the most fundamental metrics for comparing 3D shapes. To compute the Hausdorff distance efficiently from a triangular mesh A to another triangular mesh B, one needs to cull the unnecessary triangles on A quickly. These triangles have no chance to improve the Hausdorff distance estimation, that is the parts with local upper bound smaller than the global lower bound. The local upper bound estimation should be tight, use fast distance computation, and involve a small number of triangles in B during the reduction phase for efficiency. In this paper, we propose to use point-triangle distance, and only involve at most four triangles in B in the reduction phase. Comparing with the state-of-the-art proposed by Tang et al. in 2009, which uses more costly triangle-triangle distance and may involve a large number of triangles in reduction phase, our local upper bound estimation is faster, and with only a small impact on the tightness of the bound on error estimation. Such a more economic strategy boosts the overall performance significantly. Experiments on the Thingi10K dataset show that our method can achieve several (even over 20) times speedup on average. On a few models with different placements and resolutions, we show that close placement and large difference in resolution bring big challenges to Hausdorff distance computation, and explain why our method can achieve more significant speedup on challenging cases.

Keywords: computational geometry, modeling

ACM CCS: • Computing methodologies → Shape analysis; Mesh geometry models;

1. Introduction

The Hausdorff distance is a classical, *de facto* standard criterion to measure the difference between two geometric models. Its efficiency and accuracy is very important for applications like reliable shape matching [DJ94, ABB95, ZPYW18], accurate registration [Mem07], error controllable decimation [BF05], quality measurement of mesh generation [ASCE02] and so on.

Compared with point clouds, triangular meshes raise significant challenges in computing Hausdorff distance, because they include infinite number of points. Some methods approximate the problem by computing Hausdorff distance on point clouds by sampling points on triangular meshes. Although sampling more points improves the accuracy, it is difficult for the user to decide the proper number of sampling points for the required accuracy.

To compute the Hausdorff distance from triangle mesh A to another triangle mesh B, more effective methods adopt a branch-and-bound strategy [TLK09]. This strategy can quickly find the triangles in A including the 'Hausdorff points', that is the points in A with the farthest distance to the other mesh B, then accurately locate the Hausdorff points in them. The key is to cull the triangles irrelevant to the Hausdorff points quickly by using a spatial data structure and the associated bound estimation method. The BVH (bounding volume hierarchy) is a widely applied [TLK09] spatial data structure here. Though the uniform grid shows excellent performance for 'near-zero' cases [KKYK18, KYKK19], its performance strongly depends on how close the models are to each other, and whether the user properly chooses the grid density.

To provide a generally applied method, we re-explored the BVH-based method [TLK09]. This method estimates the local Hausdorff upper bounds in two phases: one selects a set L of triangles from mesh B with respect to a known triangle in mesh A, and the other estimates the upper bound of the Hausdorff distance based on the set L (see Section 3). Though any subset L gives an upper bound, the one leading to a tight upper bound is desirable for higher culling ratio. Tang et al. [TLK09] rely on the triangle-triangle distance

computation to find an L possibly composed of many triangles, which makes the upper bound estimation the most time-consuming part. We propose to sample four points in $\triangle$ and use their closest triangles in B as L. Though the resulting upper bound is slightly less tight than Tang's, our four-point strategy is much faster because it only involves a few efficient point-triangle distance computations and makes the size of L very small. In other words, we provide a more economic upper bound estimation which strikes a balance between the tightness and computational cost.

Except for the experiments about the tightness and speed of upper bound estimation, we also evaluate the overall performance of our method under various conditions. The experiments on a large dataset (Thingi10K) show over four times speedup on average, even over 10 times speedup on some models. We also check how the different placements and resolutions impact the performance of the algorithm on a few models. Results show that our method can achieve more significant speedup on challenging cases with near-zero Hausdorff distance and large resolution difference. By analysing the culling rate and the time consumption at each stage, we provide the explanation about this behaviour.

2. Background and Related Work

Hausdorff distance has various applications ranging from computer graphics [LRC*03, ABCO*03], computer vision [JKF01] to image analysis [HKR93]. The methods to calculate Hausdorff distance can be grouped as direct ones and culling-based ones.

2.1. Direct methods

Computes the exact Hausdorff distance without using the iterative cull strategy on certain spatial hierarchy data structure. Such methods usually give exact Hausdorff distance, but are only applicable to relatively simple geometry entities. Although there is a linear-time algorithm for two 2D convex polygons [Ata83], for triangular meshes, the complexity of methods [BHEK10] and [ABG*03] relying on Voronoi diagram is over $O(n^3)$ for a mesh with n triangles, which makes them not practical. For parametric surfaces or splines, the continuity of the patches implies the possibility of finding exact distance by solving equations. For example, the method in [EG08] computes the accurate distance by finding the root of a few non-linear equations. However, the complexity is high even for curves (at least $O(n^2)$ for two curves composed of $O(n)$ coefficients).

2.2. Culling-based methods

These are more suitable for many complex situations. To estimate the lower and upper bounds of the Hausdorff distance, they recursively update the bounds and cull the spatial data structure (e.g. BVH). If the upper bound of a region is smaller than the current lower bound, this region can be safely culled without any sacrifice of accuracy, thus efficiently reducing the scale of the problem. Obviously, the bound estimation through distance computation is the key of the performance.

For point cloud, the key is to avoid computing the distance of all point pairs. For two sets consisting of n and m 3D points, the algorithm [ABG*03] proposed by Alt et al. has the complexity of $O((n+m+(nm)^{\frac{2}{3}})\log(n+m))$ in randomized expected time. More efficient methods usually adopt lower and upper bound driven culling strategies. Based on aggregate nearest neighbours (ANN), Papadias et al. [PTMH05] proposed an algorithm with better performance. Nutanong et al. [NJS11] improved the algorithm in [PTMH05] by using two R-Trees for nearest neighbour querying in both directions simultaneously. Later, Taha et al. [TH15] proposed an efficient algorithm with nearly-linear complexity by combining the early breaking strategy and the random sampling method. Zhang et al. [ZHH*17] used local subalgorithms to deal with the efficiency problem in high-overlapping data. Chen et al. [CHWH17] also tried to address this problem by using local start search (LSS).

For B-spline curves, Chen et al. [CMXP10] accelerated the procedure of root-finding with a similar scheme, which prunes the subintervals via lower and upper bound estimation. Using biarc approximation, Kim et al. [KOY*10] proposed a method of the lower bound estimation to trim the 2D freeform curves with the help of a GPU rendered distance map. Except for utilizing the GPU, the key ingredients of methods [KMH11] and [HKM12] are also about effective bound estimation. In [KOY*13], to address the case of close proximity, they proposed to use a hierarchy of Coons patches to approximate the NURBS surface for more effective bound estimation.

For triangular meshes, Cignoni et al. [Str07] computed the exact Hausdorff distance from all relevant triangle pairs on the two triangular meshes. To accelerate the algorithm, they applied the voxel grid as a spatial acceleration structure. Though performance is not reported in the paper, without an efficient culling when finding the relevant triangle pairs, the computational cost should be high. The 'Metro' method [CRS98] samples one mesh and computes the distance from the sample points to another mesh with the spatial acceleration structure of the grid. Besides the problem of the unclear error bound because of the sampling strategy, no bound guided culling makes its complexity still far from satisfactory. Aspert et al. [ASCE02] accelerated the method of [CRS98] by skipping unnecessary triangles when computing the distance from the sampled points to another mesh. Instead of sampling on the mesh, Guthe et al. [GBK05] proposed an algorithm by using a subdivision strategy. It yields more efficiency, especially in high accuracy situation, and can get upper and lower bounds of Hausdorff distance, but still does not use them for further acceleration. The state-of-the-art work [TLK09] provides error bounds estimation for triangular mesh via iteratively increasing the lower bound and decreasing the upper bound, and successfully beats previous techniques in performance. In [TLK09], the authors also proposed a new upper bound estimation method to achieve higher accuracy, with which they make further culling to achieve better performance.

The performance of the method in [TLK09] is affected by the Hausdorff distance between objects. When the two models are highly overlapping, the Hausdorff distance is usually very close to zero, and only a few triangles can be culled which reduces the overall performance of the algorithm. By utilizing a uniform grid with an appropriate size, [KKYK18] and [KYKK19] get rid of an unnecessary traverse on the BVH tree and improve the performance significantly. This method is suitable for the 'near zero' cases, but not universally applicable. Moreover, the grid size for good performance is hard to determine automatically.

3. Preliminary and Overview

Before elaborating our key contribution about the upper bound estimation, we first introduce the basic concepts about Hausdorff distance computation and the BVH-based framework proposed in [TLK09].

3.1. Hausdorff distance and approximation

The Hausdorff distance [DD09] measures the maximum difference between the two models, which is defined as the maximum distance as follows:

$$h(A, B) \equiv \max_{a \in A} \left\{ \min_{b \in B} d(a, b) \right\}, \tag{1}$$

where $A \subset R^3$ and $B \subset R^3$ are two compact 3D models, and $d(\cdot, \cdot)$ denotes the Euclidean distance between two points. The Hausdorff distance is not symmetric, that is, $h(A, B) \neq h(B, A)$. To remedy this issue, a symmetric (or two-sided) Hausdorff distance is defined as $H(A, B) = max\{h(A, B), h(B, A)\}$.

In this work, we consider only the one-sided, non-symmetric Hausdorff distance defined in Equation (1).

Suppose that $A' \subseteq A$, $B' \subseteq B$ be two subsets. One can show that

$$\boxed{h(A', B) \leq h(A, B) \leq h(A, B'),} \tag{2}$$

which is proved in [TLK09]. Computing Hausdorff distance is a classical, fundamental problem in computational geometry and many graphics applications. A naïve way to compute the Hausdorff distance is to iterate all primitives on model A directly, compute the nearest distance to model B, and take the maximum. This naïve method is however not practical due to the high computational cost.

We take an approximate approach to compute the Hausdorff distance. Specifically, we compute a lower bound $\underline{h}(A, B)$ and an upper bound $\overline{h}(A, B)$, instead of the exact Hausdorff distance $h(A, B)$, as the approximated Hausdorff distance, which satisfies:

$$\underline{h}(A, B) \leq h(A, B) \leq \overline{h}(A, B).$$

Given an error tolerance ϵ, the approximation error is obviously below ϵ, as long as the bounds satisfy the following condition:

$$\overline{h}(A, B) - \underline{h}(A, B) \leq \epsilon. \tag{3}$$

Therefore, Hausdorff distance can be asymptotically approximated by a series of bounds $\{[\underline{h}_k, \overline{h}_k]\}$:

$$\underline{h}_1 \leq \underline{h}_2 \leq \cdots \leq \underline{h}_k \leq h(A, B) \leq \overline{h}_k \leq \cdots \leq \overline{h}_2 \leq \overline{h}_1,$$

until $\overline{h}_k - \underline{h}_k \leq \epsilon$.

3.2. BVH-based framework

We adopt the BVH-based framework proposed by [TLK09]. In order to make it easier to follow, we introduce the whole pipeline here. Readers familiar with the calculation of Hausdorff distance in [TLK09] can jump to Section 4 for the introduction and analysis of our method, or to Section 5 for experiment results.

Algorithm 1. Compute Hausdorff Distance $h(A,B)$

Require: model A, model B, error tolerance ε
Ensure: Hausdorff distance bound $[\underline{h}, \overline{h}]$
1: Build bounding volume hierarchy (BVH) for model A and model B
2: $H \leftarrow$MakeEmptyMaxHeap() // for storing triangles
3: TraverseModel(A.root, B, 0, H);
4: $[\underline{h}, \overline{h}] \leftarrow$ ReduceBound(A, B, ε)
5: return $[\underline{h}, \overline{h}]$

First, assuming that the models involved are represented as triangular meshes or triangle soups:

$$A = \{\triangle_j\}_{j=1}^{|A|}, \quad B = \{\blacktriangle_j\}_{j=1}^{|B|},$$

where $\triangle$ represents triangles on model A, $\blacktriangle$ represents triangles on model B.

As shown in Algorithm 1, the main steps of [TLK09] are:

1. Initialization — Build a BVH tree for A and B, and make an empty max heap.
2. Traversing — Traverse the BVH tree of model A in depth first order, obtaining a set of triangles, sorted by the upper bound of the triangles' Hausdorff distance to model B. See Algorithm 2.
3. Reduction — Examine the triangles in the heap to reduce the gap of the global Hausdorff distance bounds, until the termination condition is satisfied. See Algorithm 3.

The BVH (Bounding Volume Hierarchy) used in Algorithm 1 is a kind of spatial acceleration data structure. The basic idea of BVH is to use proxies to wrap original geometry objects and form a hierarchy tree structure to exploit pruning operation in searching operations. Different types of bounding volume generate different BVH trees. For example, the BVH tree using axis-aligned bounding box is called AABB tree, which is also used in this paper. Readers can refer to [Ber97, LGLM00] and [Eri04] for more details about BVH. In this work, AABB tree is used as the hierarchy structure.

Before going to explain the details, we first show some important properties and lemmas about the Hausdorff distance, as well as some bound estimation results, which are necessary for understanding the ideas.

3.3. Bound analysis on Hausdorff distance

Let $\{A_1, \ldots, A_J\}$ be a cover of model A, that is $A = \bigcup_{j=1}^{J} A_j$. Then, by the definition of Hausdorff distance, we have

$$h(A, B) = \max_{1 \leq j \leq J} h(A_j, B).$$

Then, a lower bound and an upper bound can be computed as:

$$\underline{h}(A,B) = \max_{1\le j\le J} \underline{h}(A_j,B), \quad \overline{h}(A,B) = \max_{1\le j\le J} \overline{h}(A_j,B).$$

Lemma 3.1. *Suppose $[\underline{h},\overline{h}]$ be a bound of $h(A,B)$, and $A'\subset A$ be a subset of model A. If $\underline{h}' = \underline{h}(A',B) > \underline{h}$, then $[\underline{h}',\overline{h}]$ is a tighter bound of $h(A,B)$.*

3.4. Subset culling by proxy bound

Let $\mathbf{P}(\cdot)$ denote a bounding volume proxy of a model. Since $A \subseteq \mathbf{P}(A)$ and it gives $\max_{\mathbf{x}\in A} d(\mathbf{x},B) \le \max_{\mathbf{x}\in\mathbf{P}(A)} d(\mathbf{x},B)$, we have

$$h(A,B) \le h(\mathbf{P}(A),B).$$

Lemma 3.2. *Suppose $[\underline{h},\overline{h}]$ a bound of $h(A,B)$, and $A'\subset A$ a subset of model A. If $\overline{h}(\mathbf{P}(A'),B) < \underline{h}$, then*

$$h(A,B) = h(A\setminus A',B).$$

Proof. From Equation (1):

$$h(A,B) = \max(h(A\setminus A',B), h(A',B)).$$

Since $A'\subset\mathbf{P}(A')$ and Equation (2), there is

$$h(A',B) \le \overline{h}(\mathbf{P}(A'),B) < L.$$

Therefore,

$$h(A,B) = h(A\setminus A',B). \qquad \square$$

This lemma states a condition to discard a subset model during computing $h(A,B)$.

3.5. Upper bound for BVH proxy

Let $\mathbf{P}$ be an AABB proxy, $\{\mathbf{p}_i\}_{1\le i\le 8}$ be the vertices of $\mathbf{P}$, and $\mathbf{b}\in B$ be an arbitrary point on model B. Since any point in the AABB can be expressed as a convex combination of the AABB's vertices, say $\mathbf{p} = \sum_i w_i\mathbf{p}_i$, $\sum_i w_i = 1$. Then we have

$$d(\mathbf{p},\mathbf{b}) = d\left(\sum_i w_i\mathbf{p}_i,\mathbf{b}\right) \le \sum_i w_i d(\mathbf{p}_i,\mathbf{b}) \le \max_i\{d(\mathbf{p}_i,\mathbf{b})\}.$$

Therefore, we have the following upper bound for the AABB proxy:

$$\boxed{h(\mathbf{P},B) \le h(\mathbf{P},\mathbf{b}) = \max_{\mathbf{p}\in\mathbf{P}} d(\mathbf{p},\mathbf{b}) \le \max_i\{d(\mathbf{p}_i,\mathbf{b})\} \equiv \overline{h}(\mathbf{P},B).} \quad (4)$$

In order to get a tighter upper bound, $\mathbf{b}$ needs to be as close to the proxy as possible. A heuristically good choice for $\mathbf{b}$ is the closest point to the proxy's center, that is

$$\mathbf{b} = \arg\min_{\mathbf{y}\in B} d(\mathbf{P}.center,\mathbf{y}), \quad (5)$$

the closest point b can be retrieved by searching the BVH tree.

3.6. Bound for triangles

Consider a triangle $\triangle = (\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3)$ on the model A. According to Equation (2), a lower bound of $h(\triangle,B)$ can be defined as

$$h(\triangle,B) = \max_{x\in\triangle} d(x,B) \ge \max_{1\le i\le 3} d(\mathbf{v}_i,B) \equiv \underline{h}(\triangle,B). \quad (6)$$

For the upper bound, derived from Equation (2), by picking some $\blacktriangle$ to build L, since any L is a subset of B, there is

$$h(\triangle,B) \le \min_{\blacktriangle\in L} h(\triangle,\blacktriangle). \quad (7)$$

The Hausdorff distance $h(\triangle,\blacktriangle)$ is computed as [Ata83]:

$$h(\triangle,\blacktriangle) = \max_{1\le i\le 3} d(\mathbf{v}_i,\blacktriangle). \quad (8)$$

The way to build set L plays a key role in Hausdorff distance computation. Generally speaking, the more triangles L contains, the tighter the upper bound is. However, a large L will incur more computation cost in upper bound estimation. Therefore, L should be carefully constructed to be as small as possible and to provide tight upper bound estimation for effective culling.

In [TLK09], L is built by a traversal on B and culling triangles on B by $d(\triangle,\blacktriangle) > \overline{h}(\triangle,B)$. It needs to calculate the distance $d(\triangle,\blacktriangle)$ between triangles. During the BVH culling to find $\blacktriangle$, the culling also involves the computation of the distance $d(\triangle,\mathbf{P})$ for BVH proxy $\mathbf{P}$ in B. The computation of such distance between a triangle and another entity is expensive compared with the computation between points and entities. In our experiments, computing point-triangle distance is about 5–10 times faster than triangle–triangle distance. According to this observation, we propose a new method to build set L which only involves the distance between point to another entity (triangle or BVH proxy). The resulting upper bound is slightly less tight than the above one, but its computational cost is much lower. As a result, the overall performance improves significantly because of the more economical balance between the tightness and computational cost. The details will be elaborated in Section 4.

3.7. Traverse phase

Line 3 in Algorithm 1 makes a traversal on A, which computes a series of ascending lower bounds and uses the lower bounds to prune the child BVH nodes and the triangles whose local upper bounds are below the global lower bound. The results are a set of triangles that can be used to approximate the Hausdorff distance. The pseudocode of this algorithm is shown in Appendix Algorithm 2.

3.8. Reduction phase

The reduction phase comes after the traverse phase, which takes advantage of the triangle heap to reduce the gap between the lower bound and the upper bound. The heap H is a max heap ordered by the upper bound Hausdorff distance from triangle to B, which is also used in [KYKK19] and [KKYK18]. The detailed algorithm is put in Appendix Algorithm 3.

Algorithm 2. TraverseModel - traverse model $\boldsymbol{A}$, culling and sorting triangles

Require: tree node N, model B, lower bound $\underline{h}$, triangle heap H
Ensure: new lower bound $\underline{h}$, new triangle H
1: N.upper $\leftarrow$ NodeBound(N) // using Eq. 4 and Eq. 5
2: **if** N.upper $< \underline{h}$ **then**
3: **return** $\underline{h}$ // N is pruned by Lemma 3.2
4: **else if** N is leaf node **then**
5: **for** each $\triangle \in N.trilist$ **do**
6: $(\triangle.lower, \triangle.upper) \leftarrow$ TriBound($\triangle$) // using Eq. 6, 10
7: **if** $\triangle.upper \geq \underline{h}$ **then**
8: H.push($\triangle$)
9: $\underline{h} \leftarrow \max\{\underline{h}, \triangle.lower\}$ // by Lemma 3.1
10: **end if**
11: **end for**
12: **else**
13: **for** each $N_c \in N.childlist$ **do**
14: N_1.upper $\leftarrow$ NodeBound(N_1, $\underline{h}$) // Using Eq. 4 and Eq. 5
15: **end for**
16: Sort $N.childlist$ by their upper bound in descending order
17: **for** each $N_c \in N.childlist$ **do**
18: $\underline{h} \leftarrow$ TraverseModelA(N_c, H, $\underline{h}$)
19: **end for**
20: **end if**
21: **return** $\underline{h}$

Algorithm 3. ReduceBound - reduce bound gap

Require: model A, model B, error tolerance ε, triangle heap H, global lower bound l
Ensure: Hausdorff distance bound $[\underline{h}, \overline{h}]$
1: $[\underline{h}, \overline{h}] \leftarrow [l, H.top.upper]$ // initialize the bound
2: **while** $\overline{h} - \underline{h} > \varepsilon$ **do**
3: $\triangle = H$.pop() // the triangle with the greatest upper bound
4: $\overline{h} \leftarrow \triangle.upper$
5: **for** each sub triangle $\mathbf{t} \in$ Subdivision($\triangle$) **do**
6: $(\mathbf{t}.lower, \mathbf{t}.upper) \leftarrow$ TriBound($\mathbf{t}$)
7: **if** $\mathbf{t}.upper \geq \underline{h}$ **then**
8: $\mathbf{t}.upper \leftarrow \min\{\mathbf{t}.upper, \overline{h}\}$
9: $H.push(\mathbf{t})$
10: $\underline{h} \leftarrow \max\{\underline{h}, \mathbf{t}.lower\}$
11: **end if**
12: **end for**
13: **end while**
14: **return** $[\underline{h}, \overline{h}]$

4. Four-Point Estimation for Upper Bound

In this section, we introduce our method to estimate the upper bound of $h(\triangle, B)$ in detail and explain the motivation behind it.

Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_J$ be a set of points sampled on triangle $\triangle \subset A$. And let $\mathbf{x}_j^* \in B$ denote the closest point to $\mathbf{x}_j$ on model B, and $\blacktriangle_{\mathbf{x}_j^*} \subset B$ denote an owner triangle of $\mathbf{x}_j^*$, that is $\mathbf{x}_j^* \in \blacktriangle_{\mathbf{x}_j^*} \subset B$.

Since every $\blacktriangle_{\mathbf{x}_j^*}$ is a subset of B, by Equation (2), we have

$$h(\triangle, B) \leq \min_{1 \leq j \leq J} h(\triangle, \blacktriangle_{\mathbf{x}_j^*}) \equiv \overline{h}(\triangle, B). \tag{9}$$

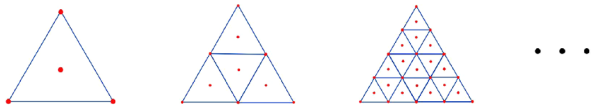

Figure 1: *Sampling stencils on a single triangle: the first takes three vertices and one centroid, the second uniformly subdivides the original triangle into four and takes the vertices and centroid points of all sub-triangles. By such stencils, the number of sampling points are: 4, 10, 31, 109, 409, 1585, and we name the upper bound estimation with a stencil containing n points as an n-point strategy.*

The triangles $\{\blacktriangle_{\mathbf{x}_j^*}, 1 \leq j \leq J\}$ that are used in the upper bound above form a set L, which has a similar meaning to the notation in [TLK09].

Substituting Equation (8) into Equation (9), we have the following upper bound for a triangle:

$$\boxed{h(\triangle, B) \leq \min_{1 \leq j \leq J} \left(\max_{1 \leq i \leq 3} d(\mathbf{v}_i, \blacktriangle_{\mathbf{x}_j^*}) \right).} \tag{10}$$

Picking different number of points on $\triangle$ can generate a series of methods to estimate the upper bound. In general, choosing fewer points leads to a faster but coarser approximation, while more points leads to tighter bounds at a slower speed. In our implementation, we use the sampling stencils illustrated in Figure 1 to take sampling points.

The key difference to Tang's algorithm is the way to build set L, which can be defined mathematically as:

$$\boxed{L_{\text{Tang}} = \left\{ \blacktriangle | d(\triangle, \blacktriangle) > \overline{h}(\triangle, B), \blacktriangle \in B \right\},} \tag{11}$$

$$\boxed{L_{\text{Ours}} = \left\{ \blacktriangle_{\mathbf{x}_j^*} | \blacktriangle_{\mathbf{x}_j^*} = \arg\min_{\blacktriangle} d(\mathbf{x}_j, \blacktriangle), \blacktriangle \in B \right\},} \tag{12}$$

where L_{Tang} and L_{Ours} represent the resulting subsets L in [TLK09] and our method, respectively. In Equation (11), L_{Tang} needs to calculate the distance between triangles, while in Equation (12), we use a much cheaper operation: the distance between point and triangle. In detail, to calculate the distance between triangles, one needs to calculate the distance between each edge pair, test whether the two triangles are overlapped, and so on. On the contrary, the distance between point and triangles can be easily calculated by a projection and some simple tests [Eri04]. In our experiment, we found the calculation of distance between point and triangle can be ten times faster than the calculation of distance between triangles. It should be noticed that during the BVH culling to find the closest triangle to a point also involves point-triangle distance evaluation. Besides, the size of L_{Ours} is limited by the number of $\mathbf{x}_j$, that is, $|L_{\text{Ours}}|$ is never larger than $|\mathbf{x}_j|$. However, $|L_{\text{Tang}}|$ depends on the threshold $\overline{h}(\triangle, B)$, which could be very large. In our experiments, $|L_{\text{Tang}}|$ could be huge if mesh B has much higher resolution than mesh A.

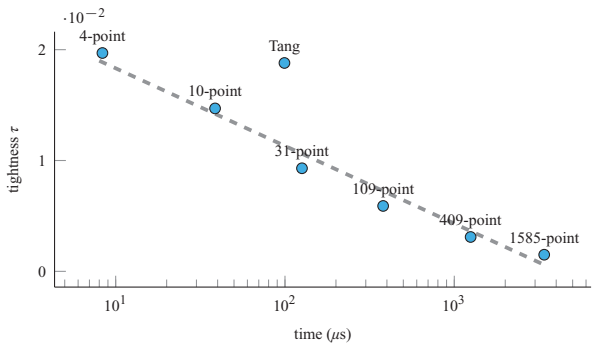

Figure 2: *Experiment results on the tightness of the upper bound (vertical axis) and its computation time (horizontal axis). 'Tang' represents the method in [TLK09] and "n-point" represents our method using n sampling points. The dashed line shows the trend that as the number of sampling points increases, the time consumption increases and the estimation becomes tighter. The slope of the trend line in this figure is* -5.3×10^{-6}.

Figure 2 shows some experimental results on the relationship between the tightness τ of upper bound $\overline{h}(\triangle, B)$ and the computation time. The tightness of the upper bound is computed as:

$$\tau = \frac{\overline{h}(\triangle, B) - h(\triangle, B)}{h(\triangle, B)}, \tag{13}$$

where $h(\triangle, B)$ is calculated by iteratively subdividing $\triangle$ until the estimated error is below 10^{-9}. We use this $h(\triangle, B)$ as the reference to estimate the tightness of each method. In this experiment, the model A is the 'dec4' model of 'hand' in Table 2 and the model B is the original model of 'hand' in Table 2.

Figure 2 gives a clear view that as the number of sampling points goes up, the upper bound gets tighter (smaller value of τ), but the computation time also increases. In our implementation, we choose '4-point' strategy. What's more, because the vertices of triangles are shared with their neighbours, we can cache the traversal results for further speedup.

5. Implementation Details and Experiments

We implement our algorithm in C++ and use an AABB tree to build the BVH. Uniform subdivision is adopted in subdivision phase. As mentioned before, the overall framework is highly consistent to [TLK09], but our method is easier to implement since we use a simpler method to estimate the upper bound.

The experiments include two parts. In the first part, we make a comparison on computation performance between our method and [TLK09] based on the Thingi10K dataset [ZJ16]. In the second part, we show the detailed algorithm behaviour under different placements and resolutions on a few models. All experiments were run on the same machine with Intel i7-8700K CPU.

5.1. Termination condition

Culling-based methods usually terminate the iteration once the lower and upper bounds are close enough. A natural and widely applied termination condition [TLK09] uses the difference between the two bounds at k-th iteration:

$$\epsilon = \overline{h}_k - \underline{h}_k, \tag{14}$$

which is an upper bound for the absolute error.

Normalizing the error in above by the diagonal length of the bounding box of A, that is $\mathrm{Diag}(A)$, gives an error measurement independent of mesh scaling:

$$\epsilon_{diag_A} = \frac{\overline{h}_k - \underline{h}_k}{\mathrm{Diag}(A)}. \tag{15}$$

ϵ_{diag_A} is in terms of mesh size instead of the exact Hausdorff distance, therefore it is not related to relative error.

Here, we would like to introduce a method to compute the upper bound of the relative error. Ideally, the relative error should be computed as:

$$\delta = \epsilon / h(A, B). \tag{16}$$

In practice, the exact Hausdorff distance $h(A, B)$ is unknown, so we use the current lower bounds to approximate it:

$$\delta = \epsilon / \underline{h}_k, \tag{17}$$

where $\underline{h}_k$ is lower than the exact Hausdorff distance, which means the error tolerance defined in Equation (17) will not exceed the definition in Equation (16). Therefore, Equation (17) defines an upper bound of the relative error.

As with Equation (15), the value of Equation (17) is independent of mesh scaling as well, and it relates to the exact Hausdorff distance. Therefore, it is more intuitive when user does not have much prior knowledge of the meshes.

However, in order to make fair comparisons to Tang's algorithm, we use Equation (14), that is the upper bound of absolute error in the termination condition in following experiments, that is the iteration stops when $\epsilon < 10^{-3}$.

5.2. Performance benchmark

In this part, we build the benchmark datasets based on Thingi10K to show the performance on models with various vertex numbers ranging from tens to millions. In the package downloaded from the Thingi10K website, we found six models without any faces, and used the 9994 models remaining below.

For a model A in the Thingi10K dataset, we use the following two methods to get another mesh as model B.

- We use the decimate modifier of blender to halve the number of vertices. The dataset generated by this method is named as 'Blender dataset', and contains 9994 pairs.
- For 9823 models in the Thingi10K dataset, TetWild [HZG*18] generates a dataset (https://github.com/Yixin-Hu/TetWild) composed of the volumetric remeshing results. We take the surface

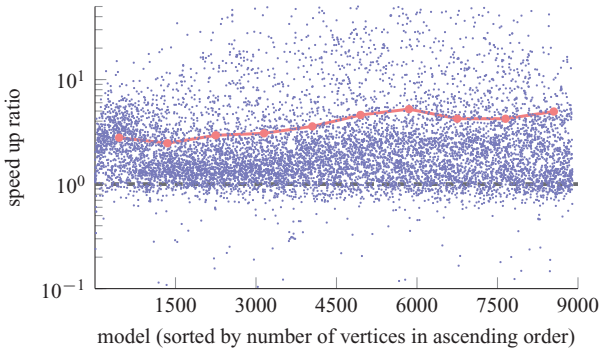

Figure 3: *Speedup on Blender dataset. There are 8896 blue points in total. The red points on the red line indicates the averaged speedup ratio in each group.*

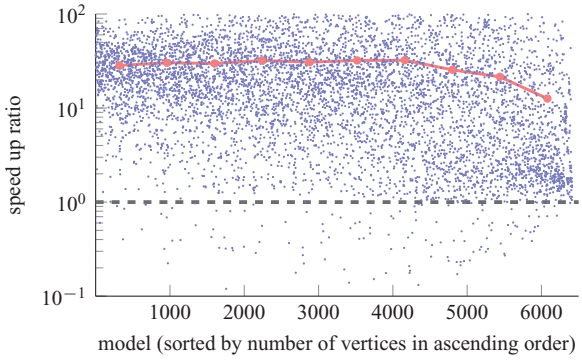

Figure 4: *Speedup on TetWild dataset. There are 6412 blue points in total. The red points on the red line indicates the averaged speedup ratio in each group.*

meshes of these volumetric meshes as model B. The dataset generated by this method is named as 'TetWild dataset', and contains 9823 pairs.

Because the two models in each pair are similar in this experiment, the ground truth Hausdorff distance is close to zero, which is a major challenge for Hausdorff distance calculation. In order to make the experiment time affordable, if our method or Tang's method does not terminate the iteration on a testing pair in 3 min, we marked this pair as a failure and did not calculate the speed-up ratio for it. In the following experiments, we only take the testing pairs that successfully terminate in 3 uus for both methods. To analyse the possible relationship between the speedup ratio and the vertex number in model A, we sort these pairs in ascending order of vertex number in model A, and plot a blue point for each pair with the coordinates (order, speedup ratio) in Figure 3 and Figure 4. For each dataset, we divide the pairs into 10 groups according to the vertex number in model A, and each group has the same number of pairs. For each group of pairs, we plot a point with the coordinates (center of the group, average speedup ratio in the group). The red curve interpolates these points piecewise linearly. As indicated by the red curves for the two datasets, the speedup ratio has no strong relationship to the vertex number in model A.

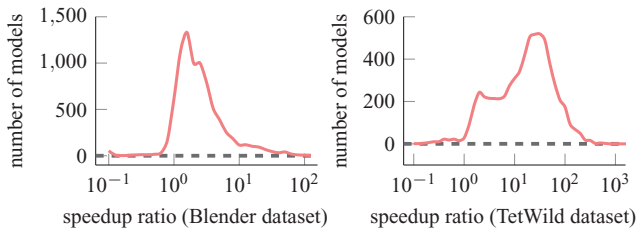

Figure 5: *Distribution histogram of speedup ratio.*

Table 1: *Statistics of the benchmark experiments. The column of '< 3 min' lists the percentage of finishing computation in 3 min. From left to right, the three numbers in this column are for our method, [TLK09] and both methods, respectively. The other two columns lists the average speedup ratio and ratio of winnings of our method over [TLK09], respectively.*

	< 3 min	average	winnings
Blender dataset	97.65%/91.05%/89.01%	4.38	87.94%
TetWild dataset	97.49%/66.44%/65.28%	27.50	97.86%

We also show the histogram of the speedup in Figure 5, and list some statistics of the experiments in Table 1. In some cases, our method is slower than [TLK09] (12.06% of cases in models generated by Blender and 2.14% of cases in models generated by TetWild). The main reason is that the upper bound of our method is weaker, which may bring extra subdivision operations, the cost of which cannot be compensated for by the faster upper bound estimations. There are cases that only one method (our method or Tang's method) terminates the iteration in 3 min, so in the column of '< 3 min', the last number is smaller than the previous two.

5.3. Behaviour analysis with respect to different input conditions

Besides the overall benchmark on a large dataset, we are also very interested in the performance behaviour on various input conditions. Since our algorithm is based on culling and subdivision operations, the number of triangles culled and the number of subdivisions needed to reach the desired precision are crucial to the overall performance. We noticed that the culling rate and subdivisions are highly dependent on the conditions of input models, especially the two key factors: the placements of the models and the 'resolution' difference between models. 'Resolution' here means the number of triangles in a model. The high-resolution models have more triangles than low-resolution models. In the following, we first discuss the placement, then resolution, and the models used in this experiment are shown in Figure 6.

5.3.1. *With respect to different placements*

The relative placement between models affects the culling rate in the traversal stage of the algorithm. As an intuitive example, for two identical models placed at the same place with the same orientation, the Hausdorff distance will be zero. Therefore, the lower bound of Hausdorff distance is zero as well, so no triangle can be culled in the

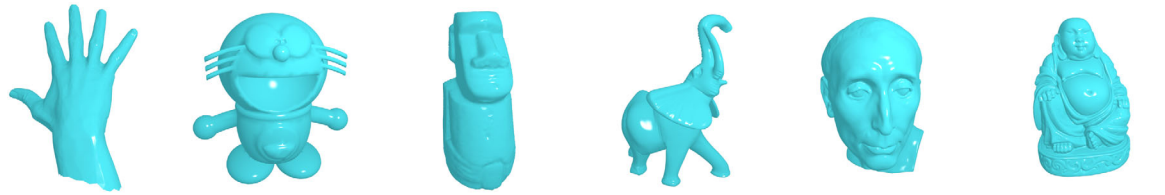

Figure 6: *Dataset used in algorithmic behaviour analysis. The name and number of vertices are listed below each model.*

Table 2: *Number of vertices in the original and decimated meshes.*

model	original	dec1	dec2	dec3	dec4
hand	3426	2054	1232	738	442
robocat	7512	4506	2702	1620	972
moai	20000	12000	7200	4320	2592
elephant	49918	29950	17970	10782	6468
nicolo	100444	60265	36159	21695	13017
buddha	126524	75914	45548	27328	16396

Figure 7: *Illustration of difference placements parameterized by d. In each pair, model A is shifted to model B by distance $d \times Diag(A)$. From left to right, $d = 0.01, 0.05, 0.09$.*

traversal stage and the culling rate will be zero. A low culling rate increases the computation cost in the traversal stage and significantly raises the number of triangles remaining in the subdivision stage.

To illustrate this factor, for each model A in Table 2, we shift A by distance $d \times \text{Diag}(A)$ as model B, and calculate the Hausdorff distance from A to B. Figure 7 shows a few examples of different placements parameterized by the value d.

The culling rate and the time consumption at each stage are shown in Figure 8. One can find that the culling rate decreases quickly as the distance between models becomes smaller. We also find that the culling rate of our algorithm is a little bit lower than Tang's algorithm because their upper bound is slightly tighter than ours, but our overall performance is much higher. That's why we claim our method is more economic in the sense of better balance between the tightness and computational cost. When the distance is close to zero, the benefit from the economic balance is more obvious because there are more upper bound estimations involved.

With respect to different resolutions. We use the Decimate Modifier tool of Blender to generate a series of meshes with different resolutions. Every time, the number of vertices is decimated by 40%. The detail about the simplified meshes is shown in Table 2.

In this experiment, we calculate the Hausdorff distance from each simplified mesh to the original one. Both meshes are put at the same position with the same orientation, so the Hausdorff distance is close to zero. The culling rate is relatively low for both our method and Tang's algorithm, which matches the behaviour of the previous experiments about different placements. The time consumption at each stage is shown in Figure 9, and the detailed results can be found in the supplementary material.

The results show that as the simplification goes on, the time cost in traversal stage tends to decrease, but the time cost in subdivision tends to grow. Such a behaviour for the traversal stage is intuitive: the main computation cost is in the traversal on BVH trees. As the number of vertices in A decreases, the number of proxies in the BVH tree of A decreases too, which leads to a faster traversal stage. Although the behaviour during the subdivision stage looks strange at first sight, it is reasonable because lower resolution A needs more subdivisions of triangles to terminate the iteration for a fixed error tolerance. In this experiment, the time cost of Tang's algorithm grows much faster than ours because it suffers from the increasing size of L. To be more specific, the mesh A with lower resolution leads to a larger size of L for each triangle survived after the traversal stage, and then the larger the size of L is, the more triangle-triangle distance evaluations [KOY*10] are required for upper bound estimation.

6. Conclusion

In this paper, we accelerate the Hausdorff distance computation method proposed in [TLK09] by replacing the triangle based upper bound estimation by a four-point strategy. Such a small change leads to surprising speedup, but can be explained by checking the tightness and computational cost.

Our BVH based method is slower than the grid based one [KYKK19] when two meshes are very close to each other and the grid size is properly chosen. After all, [KYKK19] is specially designed for such cases. However, their method also involves bound estimation, and integrating our idea with their method may bring better performance.

As many previous methods, our method is based on floating-point representation. Therefore, it makes the predication (e.g. $\mathbf{t}.upper \geq \underline{h}$ in Algorithm 3) depending on the roundoff of floating-point coordinates, and leads to inaccurate result.

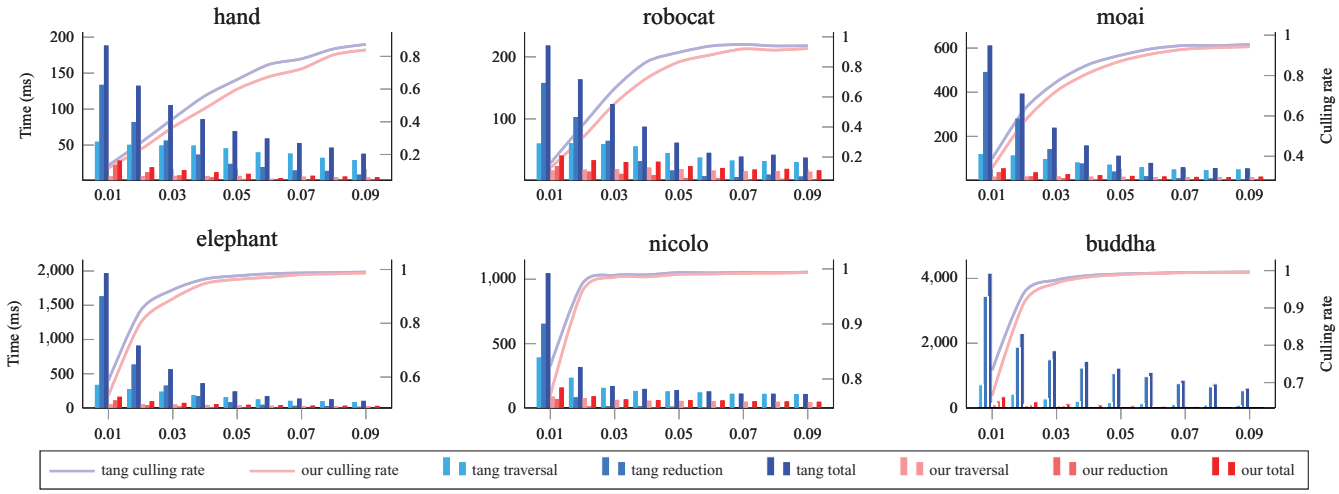

Figure 8: *Experiment results of different placements. The numbers on horizontal axis indicate the distance parameter d. The bar charts show the time consumption in traversal stage, subdivision stage and the whole algorithm at different placements. The line charts indicate the culling rate after the traversal stage.*

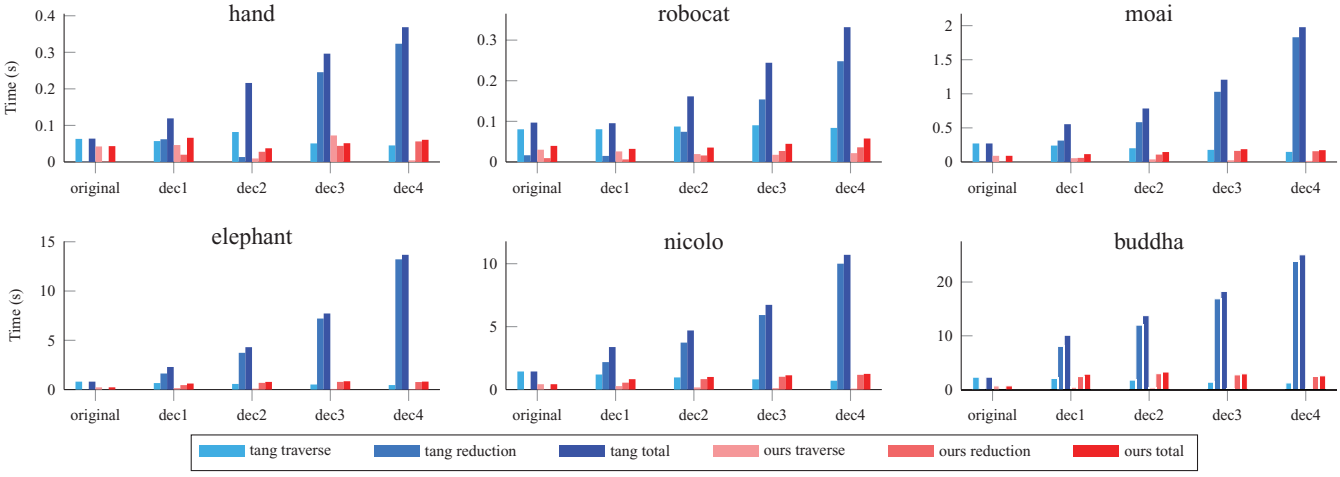

Figure 9: *Experiment results on meshes with different resolutions. We calculate the Hausdorff distance from each decimated mesh to the original one at the same pose. The horizontal axis represents a series of meshes, from the finest to the coarsest (original, dec, dec2, dec3, dec4), while the vertical axis represents the time of computation, measured in seconds.*

More detailed understanding about the algorithm behaviour is important. We noticed that the speedup ratio on TetWild dataset is much higher than the one on Blender dataset, but cannot explain it clearly. The overall speed up ratio is mainly affected by three factors: distance calculation, size of L and the tightness of upper bound. Our method uses point-based distance instead of triangle-based one used by [TLK09], which makes a single distance calculation about 10 times faster (see Figure 2). Since we used four-point estimation strategy, the size of L in our method is no more than 4, while the size of L in [TLK09] varies significantly and is generally larger than 4. The last factor, tightness of upper bound, works against our method. Its specific impact depends on input models, which is difficult to analyse. If one can get a mathematical model to relate the overall timing, tightness of bound and cost of bound estimation and so on, there is the possibility of developing more economic strategy.

The code can be downloaded from https://github.com/ZJUCADGeoSim/Hausdorff. We hope it is not only a tool for Hausdorff distance computation, but also a baseline for further research.

Acknowledgements

We would like to thank the anonymous reviewers and editors for their constructive comments and suggestions. This work was supported by National Key R&D Program of China (No. 2020AAA0108901), NSFC (No. 61732016, 62032011), and

Zhejiang Provincial Science and Technology Program in China under Grant 2021C01108.

Appendix A: Traverse Algorithm

It begins with the root node of model A's BVH tree. When visiting a node, say node N, it firstly estimates the upper bound for the node (according to Equation 4). If node N's upper bound is smaller than the lower bound $\underline{h}$, node N can be pruned (according to Lemma 3.2). If N is a leaf node, there is a list of triangles stored in it. Then, we push into heap H the triangles whose upper bounds are above the current lower bound $\underline{h}$, while ignoring the triangles whose upper bounds are below $\underline{h}$. Whenever there is a greater lower bound in the triangles, the current lower bound $\underline{h}$ will be updated by the greater one (by Lemma 3.1). If N is not a leaf node, the traverse algorithm will recursively visit the child nodes. One can find that, during the traversal, the lower bound $\underline{h}$ never goes downward.

Function NodeBound in line 1 of Algorithm 2 computes the upper distance bound for node N.

Function TriBound in line 6 of Algorithm 2 computes both the lower bound and the upper bound for triangle $\triangle$.

Appendix B: Reduction Algorithm

In Algorithm 3, we initialize the global lower and upper bound $\underline{h}$ and $\overline{h}$ by the lower bound calculated in Algorithm 2 and the upper bound of the top element in H. In every iteration, we take the triangle with the largest upper bound from the heap.

References

[ABB95] Alt H., Behrends B., Blömer J.: Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence 13*, 3-4 (1995), 251–265.

[ABCO*03] Alexa M., Behr J., Cohen-Or D., Fleishman S., Levin D., Silva C. T.: Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9*, 1 (2003), 3–15.

[ABG*03] Alt H., Brass P., Godau M., Knauer C., Wenk C.: Computing the Hausdorff distance of geometric patterns and shapes. In *Discrete and Computational Geometry*. Springer, Berlin, Germany, 2003, pp. 65–76.

[ASCE02] Aspert N., Santa-Cruz D., Ebrahimi T.: Mesh: Measuring errors between surfaces using theHausdorff distance. In *Proceedings of IEEE International Conference on Multimedia and Expo* (2002), vol. 1, IEEE, pp. 705–708.

[Ata83] Atallah M. J.: A linear time algorithm for the Hausdorff distance between convex polygons. *Information Processing Letters 17*, 4 (Nov 1983), 207–209.

[Ber97] Bergen G. v. d.: Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools 2*, 4 (1997), 1–13.

[BF05] Borouchaki H., Frey P.: Simplification of surface mesh using Hausdorff envelope. *Computer Methods in Applied Mechanics and Engineering 194*, 48 (2005), 4864–4884. Unstructured Mesh Generation.

[BHEK10] Bartoň M., Hanniel I., Elber G., Kim M.-S.: Precise Hausdorff distance computation between polygonal meshes. *Computer Aided Geometric Design 27*, 8 (2010), 580–591.

[CHWH17] Chen Y., He F., Wu Y., Hou N.: A local start search algorithm to compute exact Hausdorff distance for arbitrary point sets. *Pattern Recognition 67* (Jul 2017), 139–148.

[CMXP10] Chen X.-D., Ma W., Xu G., Paul J.-C.: Computing the hausdorff distance between two b-spline curves. *Computer-Aided Design 42*, 12 (2010), 1197–1206.

[CRS98] Cignoni P., Rocchini C., Scopigno R.: Metro: measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (1998), 167–174.

[DD09] Deza M. M., Deza E.: Encyclopedia of distances. In *Encyclopedia of Distances*. Springer, Berlin, Germany, 2009, pp. 1–583.

[DJ94] Dubuisson M.-P., Jain A. K.: A modified hausdorff distance for object matching. In *Proceedings of 12th international conference on pattern recognition* (1994), vol. 1, IEEE, pp. 566–568.

[EG08] Elber G., Grandine T.: Hausdorff and minimal distances between parametric freeforms in $\mathbb{R}^2$ and $\mathbb{R}^3$. Springer-Verlag.

[Eri04] Ericson C.: *Real-Time Collision Detection*. CRC Press, Inc., USA, 2004.

[GBK05] Guthe M., Borodin P., Klein R.: Fast and accurate hausdorff distance calculation between meshes. *Journal of WSCG 13*, 2 (Feb. 2005), 41–48.

[HKM12] Hanniel I., Krishnamurthy A., McMains S.: Computing the hausdorff distance between nurbs surfaces using numerical iteration on the gpu. *Graphical Models 74*, 4 (2012), 255–264.

[HKR93] Huttenlocher D. P., Klanderman G. A., Rucklidge W. J.: Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*, 9 (1993), 850–863.

[HZG*18] Hu Y., Zhou Q., Gao X., Jacobson A., Zorin D., Panozzo D.: Tetrahedral meshing in the wild. *ACM Transactions on Graphics 37*, 4 (July 2018).

[JKF01] Jesorsky O., Kirchberg K. J., Frischholz R. W.: Robust face detection using the Hausdorff distance. In *Audio- and Video-Based Biometric Person Authentication* (Berlin, Germany, Aug 2001), Springer, pp. 90–95.

[KKYK18] Kang Y., Kyung M.-H., Yoon S.-H., Kim M.-S.: Fast and robust hausdorff distance computation from triangle mesh to

quad mesh in near-zero cases. *Computer Aided Geometric Design 62* (May 2018), 91–103.

[KMH11] Krishnamurthy A., McMains S., Hanniel I.: Gpu-accelerated Hausdorffff distance computation between dynamic deformable nurbs surfaces. *Computer-Aided Design 43*, 11 (2011), 1370–1379.

[KOY*10] Kim Y.-J., Oh Y.-T., Yoon S.-H., Kim M.-S., Elber G.: Precise Hausdorffff distance computation for planar freeform curves using biarcs and depth buffer. *The Visual Computer 26*, 6-8 (2010), 1007–1016.

[KOY*13] Kim Y.-J., Oh Y.-T., Yoon S.-H., Kim M.-S., Elber G.: Efficient hausdorff distance computation for freeform geometric models in close proximity. *Computer-Aided Design 45*, 2 (2013), 270–276.

[KYKK19] Kang Y., Yoon S.-H., Kyung M.-H., Kim M.-S.: Fast and robust computation of the Hausdorffff distance between triangle mesh and quad mesh for near-zero cases. *Computers & Graphics 81* (Jun 2019), 61–72.

[LGLM00] Larsen E., Gottschalk S., Lin M. C., Manocha D.: Fast distance queries with rectangular swept sphere volumes. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)* (2000), vol. 4, IEEE, pp. 3719–3726.

[LRC*03] Luebke D., Reddy M., Cohen J. D., Varshney A., Watson B., Huebner R.: *Level of detail for 3D graphics*. Morgan Kaufmann, 2003.

[Mem07] Memoli F.: On the use of Gromov-Hausdorff distances for shape comparison. In *Eurographics Symposium on Point-Based Graphics* (2007), Botsch M., Pajarola R., Chen B., Zwicker M., (Eds.), The Eurographics Association.

[NJS11] Nutanong S., Jacox E. H., Samet H.: An incremental Hausdorffff distance calculation algorithm. *Proceedings of the VLDB Endowment 4*, 8 (2011), 506–517.

[PTMH05] Papadias D., Tao Y., Mouratidis K., Hui C. K.: Aggregate nearest neighbor queries in spatial databases. *ACM Transactions on Database Systems (TODS) 30*, 2 (2005), 529–576.

[Str07] Straub R.: Exact Computation of the Hausdorff Distance between Triangular Meshes. In *EG Short Papers* (2007), Cignoni P., Sochor J., (Eds.), The Eurographics Association.

[TH15] Taha A. A., Hanbury A.: An efficient algorithm for calculating the exact Hausdorffff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 37*, 11 (2015), 2153–2163.

[TLK09] Tang M., Lee M., Kim Y. J.: Interactive Hausdorffff distance computation for general polygonal models. *ACM Transactions on Graphics (TOG) 28*, 3 (July 2009).

[ZHH*17] Zhang D., He F., Han S., Zou L., Wu Y., Chen Y.: An efficient approach to directly compute the exact Hausdorffff distance for 3D point sets. *Integrated Computer-Aided Engineering 24*, 3 (2017), 261–277.

[ZJ16] Zhou Q., Jacobson A.: Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).

[ZPYW18] Zhang J., Pang J., Yu J., Wang P.: An efficient assembly retrieval method based on Hausdorffff distance. *Robotics and Computer-Integrated Manufacturing 51* (Jun 2018), 103–111.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Data S1

Data S2