Interactive Hausdorff Distance Computation for General Polygonal Models

Min Tang* Minkyoung Lee† Young J. Kim‡
Ewha Womans University, Seoul, Korea
http://graphics.ewha.ac.kr/HDIST

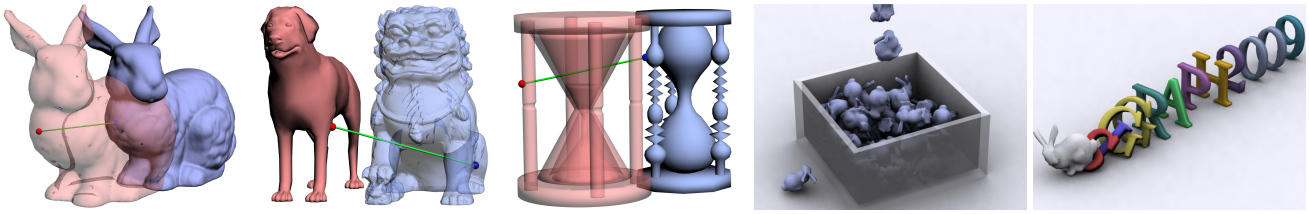


Figure 1: ***Interactive Hausdorff Distance Computation.*** *Our algorithm can compute Hausdorff distance between complicated models at interactive rates (the first three figures). Here, the green line denotes the Hausdorff distance. This algorithm can also be used to find penetration depth (PD) for physically-based animation (the last two figures). It takes only a few milli-seconds to run on average.*

Abstract

We present a simple algorithm to compute the Hausdorff distance between complicated, polygonal models at interactive rates. The algorithm requires no assumptions about the underlying topology and geometry. To avoid the high computational and implementation complexity of exact Hausdorff distance calculation, we approximate the Hausdorff distance within a user-specified error bound. The main ingredient of our approximation algorithm is a novel polygon subdivision scheme, called *Voronoi subdivision*, combined with culling between the models based on bounding volume hierarchy (BVH). This *cross-culling* method relies on tight yet simple computation of bounds on the Hausdorff distance, and it discards unnecessary polygon pairs from each of the input models alternatively based on the distance bounds. This algorithm can approximate the Hausdorff distance between polygonal models consisting of tens of thousands triangles with a small error bound in real-time, and outperforms the existing algorithm by more than an order of magnitude. We apply our Hausdorff distance algorithm to the measurement of shape similarity, and the computation of penetration depth for physically-based animation. In particular, the penetration depth computation using Hausdorff distance runs at highly interactive rates for complicated dynamics scene.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Keywords: Hausdorff Distance; Penetration Depth; Shape Similarity; Dynamics Simulation.

1 Introduction

*tangmin@ewha.ac.kr
†minkyounglee@ewhain.net
‡kimy@ewha.ac.kr

Computing a distance measure between geometric models is an important problem in diverse fields including computer graphics, computer games, virtual environment, geometric modeling, and robotics. Various types of distance measures have been extensively investigated and efficient algorithms have been proposed over the past two decades [Lin and Manocha 2003]. In particular, because of their practical importance, fast and reliable algorithms to compute Euclidean distance (also known as *separation distance*) have been proposed by many researchers, and are considered as a solved problem for polygonal models in $\mathbb{R}^3$ [Larsen et al. 2000; Ehmann and Lin 2001]. In contrast, distance measures appropriate for quantifying the similarity between two polygonal models, such as Hausdorff distance, have been relatively poorly studied, even though there are many graphics and computer vision applications that would benefit from such measures. These include shape matching [Alt and Guibas 2000], mesh simplification [Luebke et al. 2002; Lopez and Reisner 2008], geometric modeling [Varadhan and Manocha 2004], model rendering [Alexa et al. 2003], image registration and recognition [Zitová and Flusser 2003; Huttenlocher et al. 1993b], and face detection [Jesorsky et al. 2001]. However, due to the high computational complexity and difficult implementation of proposed approaches, very few algorithms exist to compute Hausdorff distance for polygonal models in $\mathbb{R}^3$. Thus, many applications circumvent this problem: for instance, by using conservative approximations [Cohen et al. 1996] or by employing other measures [Garland and Heckbert 1997].

Intuitively speaking, the Hausdorff distance between two models is the maximum deviation between them. For polygonal models in $\mathbb{R}^3$ with $O(n)$ polygons, the expected time taken by the best-known algorithm to exactly evaluate the Hausdorff distance is $O(n^{3+\epsilon})$, where $\epsilon > 0$ [Alt et al. 2003]. Moreover, this algorithm requires the computation of a lower envelope of a set of non-linear algebraic surfaces in $\mathbb{R}^3$, which is prone to degeneracies and numerical error, making it quite challenging to implement for high values of n [Kettner et al. 2008]. This is hardly a practical algorithm.

MAIN RESULTS We present a fast and simple algorithm to compute the Hausdorff distance between complicated polygonal models at interactive rates. Our algorithm makes no assumptions about the underlying topology and geometry of the models. To the best of our knowledge, this is the first real-time algorithm that can calculate Hausdorff distances between complicated polygonal models. Avoiding the complexity of exact evaluation of Hausdorff distance, our algorithm approximates the distance within a user-specified error bound. It does this by calculating tight upper and lower bounds to the exact Hausdorff distance value, and then it refines these

bounds by polygon subdivision until the error bound is obtained. We also prove inclusion properties related to Hausdorff distance measures, and utilize these properties to perform efficient bounding volume hierarchy (BVH) culling on the input models. Thus, our algorithm is able to calculate Hausdorff distance for *polygon-soup models* consisting of tens of thousands of triangles in real-time. As an application of our algorithm, we show how it can rapidly calculate a similarity between polygonal models for shape analysis, and we also show how to compute penetration depth (PD) efficiently for physically-based animation. We define the PD as the point of maximal mutual interpenetration of the two models, which can be formulated as the Hausdorff distance of the intersection of the two models. In practice, we show that our PD algorithm takes only a few milli-seconds to run for complicated scene.

2 Previous Work

Since the seminal work by [Atallah 1983], different algorithms for calculating Hausdorff distances have been proposed in the literature, in particular by the computational geometry community. In this section, we briefly survey work which is directly relevant to ours, and refer readers to [Alt and Guibas 2000] for a more extensive survey of the field. Broadly, the literature on Hausdorff distance algorithms can be classified based on their objectives and the types of models dealt with as follows:

Point-Sets Given two point-sets with n and m points respectively, a brute-force algorithm to compute the Hausdorff distance requires $O(nm)$ time. For $\mathbb{R}^2$, [Alt et al. 1995] presented a method based on the Voronoi diagram which requires $O((n+m)\log(n+m))$ running time. For $\mathbb{R}^3$, [Alt et al. 2003] proposed a randomized algorithm with $O((n+m+(nm)^{\frac{3}{4}})\log(n+m))$ expected time.

Polytopes and Polygonal Models For $\mathbb{R}^2$, [Atallah 1983] presented a linear-time algorithm for convex polygons. For simple, non-convex polygons with n and m vertices, [Alt et al. 1995] presented an $O((n+m)\log(n+m))$-time algorithm based on an observation that the Hausdorff distance can only be realized at the points of intersection between Voronoi boundary surfaces and the polygon edges. In $\mathbb{R}^3$, for polygon-soup models with n triangles, [Godau 1998; Alt et al. 2003] presented deterministic and randomized algorithms that run respectively in $O(n^5)$ and $O(n^{3+\epsilon})$, where $\epsilon > 0$. [Llanas 2005] proposed an algorithm using random covering, and also demonstrated implementation results for simple, convex ellipsoids. Recently, more practical algorithms have been put forward, but all these algorithms only approximate the Hausdorff distance due to the complexity of the exact computation. [Guthe et al. 2005] use polygon subdivision to approximate the solution within an error bound. [Cignoni et al. 1998; Aspert et al. 2002] sample a polygonal surface to approximate the distance, but no sampling analysis is provided. However, the performance of these algorithms is still too slow for interactive applications: they require tens of seconds or more for models of practical size.

Hausdorff Distance under Motion An important variation of the Hausdorff distance problem is that of finding the minimal Hausdorff distance when one of the models is allowed to move. This problem is known as 'geometric matching' under the Hausdorff distance metric. For $\mathbb{R}^2$, [Huttenlocher et al. 1992; Huttenlocher et al. 1993a] presented methods based on upper envelopes of a Voronoi diagram, and dynamic Voronoi diagrams which can find matches under translation and rigid motions. [Chew et al. 1997] proposed methods of matching points and line segments under rigid motions using parametric searching. [Agarwal et al. 2003] also used a parametric searching technique to match points, balls and unions of balls under translation in $\mathbb{R}^3$. However, none of these algorithms have been implemented in practice. The complexities of different matching problems in $\mathbb{R}^2$ are summarized by [Rucklidge 1996]. [Goodrich et al. 1999] implemented approximate matching for point-sets under rigid motions in $\mathbb{R}^2$ and $\mathbb{R}^3$ using approximate nearest-neighborhood search. No matching algorithm is known to have been implemented for polygon-soup models in $\mathbb{R}^3$.

3 Preliminaries and Overview

In this section, we present some preliminary concepts and theorems related to our Hausdorff distance algorithm before presenting the algorithm itself.

3.1 Problem Formulation

We define our problem of Hausdorff distance computation as follows:

DEFINITION 1
Given two compact sets $\mathcal{A}$ and $\mathcal{B}$ in $\mathbb{R}^3$, the one-sided Hausdorff distance[1] *from $\mathcal{A}$ to $\mathcal{B}$ is defined as:*

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

where $d(\cdot,\cdot)$ denotes the Euclidean distance operator in $\mathbb{R}^3$. Then, the two-sided Hausdorff distance between $\mathcal{A}$ and $\mathcal{B}$ is defined as:

$$H(\mathcal{A},\mathcal{B}) \equiv \max\left(h(\mathcal{A},\mathcal{B}), h(\mathcal{B},\mathcal{A})\right). \tag{2}$$

From the above definition, one can trivially derive the following theorem for polygonal models:

THEOREM 1
If $\mathcal{A}$ and $\mathcal{B}$ are polygonal models, and $\Delta^{\mathcal{A}}$ denotes a triangle in $\mathcal{A}$, then

$$h(\mathcal{A},\mathcal{B}) = \max_{\Delta^{\mathcal{A}}\in\mathcal{A}} \left(h(\Delta^{\mathcal{A}},\mathcal{B}) \right). \tag{3}$$

From Theorem 1, computing $h(\mathcal{A},\mathcal{B})$ boils down to computing $h(\Delta^{\mathcal{A}},\mathcal{B})$. Now we will present lemmas related to the bounds of the Hausdorff distance metric, which will plays a vital role in cross-culling in Sec. 4.

LEMMA 1
Given compact sets $\mathcal{A},\mathcal{A}',\mathcal{B},\mathcal{B}'$, with $\mathcal{A}\subseteq\mathcal{A}'$ and $\mathcal{B}\subseteq\mathcal{B}'$, the following inequalities hold [Tang and Kim 2009]:

$$\begin{aligned} h(\mathcal{A},\mathcal{B}') &\le h(\mathcal{A},\mathcal{B}) \\ h(\mathcal{A},\mathcal{B}) &\le h(\mathcal{A}',\mathcal{B}). \end{aligned} \tag{4}$$

Based on the above lemma, we present a simple way to compute the upper and lower bounds of the one-sided Hausdorff distance between polygonal models.

LEMMA 2
Let $\mathbf{v}_i^{\mathcal{A}}(i=1,2,3)$ represent one of the three vertices of a triangle $\Delta^{\mathcal{A}}\in\mathcal{A}$. Then, the upper and lower bounds of $h\left(\Delta^{\mathcal{A}},B\right)$ can be obtained as follows [Tang and Kim 2009]:

$$\begin{aligned} \overline{h}(\Delta^{\mathcal{A}},\mathcal{B}) &= \min_{\Delta_{\mathcal{B}}\in\mathcal{B}} \left(h\left(\Delta^{\mathcal{A}},\Delta^{\mathcal{B}}\right)\right) \\ \underline{h}(\Delta^{\mathcal{A}},\mathcal{B}) &= \max_{i=1,2,3} \left(d\left(\mathbf{v}_i^{\mathcal{A}},\mathcal{B}\right)\right) \end{aligned} \tag{5}$$

[1] In most places, where it is clear, we mean one-sided Hausdorff distance when we refer to Hausdorff distance.

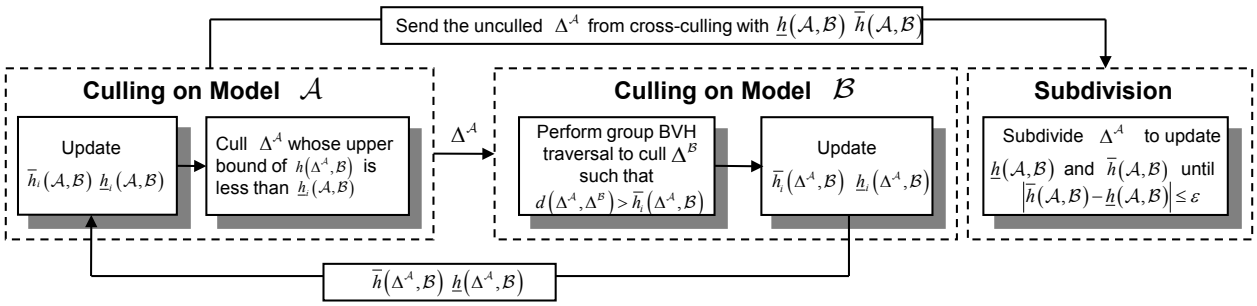


Figure 2: ***Hausdorff Distance Computation Algorithm.***

Notice that the lower bound $\underline{h}(\Delta^{\mathcal{A}},\mathcal{B})$ (or $d(\mathbf{v}_i^{\mathcal{A}},\mathcal{B})$) can easily be obtained by using a known proximity package such as the PQP library [Larsen et al. 2000], with minor modifications, since this computation only requires Euclidean distance from points to objects. Moreover, since $h\left(\Delta^{\mathcal{A}},\Delta^{\mathcal{B}}\right) = \max_i \left(d\left(\mathbf{v}_i^{\mathcal{A}},\Delta^{\mathcal{B}}\right)\right)$, where $\mathbf{v}_i^{\mathcal{A}}$ is a vertex in $\Delta^{\mathcal{A}}$ [Atallah 1983], the upper bound $\overline{h}(\Delta^{\mathcal{A}},\mathcal{B})$ can also be obtained simply.

THEOREM 2
We have the upper bound $\overline{h}$ and the lower bound $\underline{h}$ of $h(\mathcal{A},\mathcal{B})$ as follows:

$$\begin{aligned}\overline{h}(\mathcal{A},\mathcal{B}) &= \max_{\Delta^{\mathcal{A}}\in\mathcal{A}}\left(\overline{h}(\Delta^{\mathcal{A}},\mathcal{B})\right)\\ \underline{h}(\mathcal{A},\mathcal{B}) &= \max_{\Delta^{\mathcal{A}}\in\mathcal{A}}\left(\underline{h}(\Delta^{\mathcal{A}},\mathcal{B})\right)\end{aligned} \tag{6}$$

Proof The result follows from Theorem 1 and Lemma 2. □

3.2 Overview

Based on Theorem 1, the main step of our algorithm is to calculate $h(\Delta_i^{\mathcal{A}},\mathcal{B})$ for each $\Delta_i^{\mathcal{A}} \in \mathcal{A}, 1 \le i \le |A|$ and then to maximize it. However, computing $h(\Delta_i^{\mathcal{A}},\mathcal{B})$ exactly is expensive and is avoided by our algorithm. We therefore proceed by making the significant observation that some triangle $\Delta_j^{\mathcal{A}}$ may not contribute to the final value of $h(\mathcal{A},\mathcal{B})$, if there exists another triangle $\Delta_i^{\mathcal{A}}$ such that $\overline{h}(\Delta_j^{\mathcal{A}},\mathcal{B}) < \underline{h}(\Delta_i^{\mathcal{A}},\mathcal{B})$. More precisely, let us call $\underline{h}_i(\mathcal{A},\mathcal{B})$ as the *current* lower bound of $\underline{h}(\mathcal{A},\mathcal{B})$ after the first i triangles in $\mathcal{A}$ have been considered such that:

$$\underline{h}_i(\mathcal{A},\mathcal{B}) \equiv \max_{1\le j\le i} \underline{h}(\Delta_j^{\mathcal{A}},\mathcal{B}). \tag{7}$$

Then, when we consider the $(i+1)$th triangle $\Delta_{i+1}^{\mathcal{A}}$, we compute its upper bound $\overline{h}(\Delta_{i+1}^{\mathcal{A}},\mathcal{B})$ and compare it against $\underline{h}_i(\mathcal{A},\mathcal{B})$. If $\overline{h}(\Delta_{i+1}^{\mathcal{A}},\mathcal{B}) < \underline{h}_i(\mathcal{A},\mathcal{B})$, then we can safely cull $\Delta_{i+1}^{\mathcal{A}}$; otherwise, we calculate a lower bound $\underline{h}(\Delta_{i+1}^{\mathcal{A}},\mathcal{B})$, and update the current lower bound, and also keep the triangle $\Delta_{i+1}^{\mathcal{A}}$ to calculate $h(\Delta_{i+1}^{\mathcal{A}},\mathcal{B})$ more precisely. We call this process *culling on model* $\mathcal{A}$ (more details are provided in Sec. 4.1).

Further, when computing an upper bound $\overline{h}(\Delta^{\mathcal{A}},\mathcal{B})$, it is not necessary to consider $h(\Delta^{\mathcal{A}},\Delta^{\mathcal{B}})$ for all triangles $\Delta^{\mathcal{B}}$ in $\mathcal{B}$, since we are only interested in the minimum of $h(\Delta^{\mathcal{A}},\Delta^{\mathcal{B}})$ over all $\Delta^{\mathcal{B}} \in \mathcal{B}$. Similar reasoning can be applied to the lower bound computation. This step of *culling on model* $\mathcal{B}$ will be explained in Sec. 4.2.

After completing this cross-culling on $\mathcal{A}$ and $\mathcal{B}$, we have a list of triangles $\Delta^{\mathcal{A}}$ which survived the cull, as well as their individual upper and lower bounds $\overline{h}(\Delta^{\mathcal{A}},\mathcal{B}), \underline{h}(\Delta^{\mathcal{A}},\mathcal{B})$. We can also compute the upper and lower bounds $\overline{h}(\mathcal{A},\mathcal{B}), \underline{h}(\mathcal{A},\mathcal{B})$ of $h(\mathcal{A},\mathcal{B})$ from $\overline{h}(\Delta^{\mathcal{A}},\mathcal{B}), \underline{h}(\Delta^{\mathcal{A}},\mathcal{B})$, as shown in Theorem 2. Then, if there is a triangle $\Delta^{\mathcal{A}}$ such that $\overline{h}(\Delta^{\mathcal{A}},\mathcal{B}) < \underline{h}(\mathcal{A},\mathcal{B})$, it cannot contribute to $h(\mathcal{A},\mathcal{B})$, and we can further remove these triangles. For the remaining triangles $\Delta^{\mathcal{A}}$, we need to compute their Hausdorff distances within the user-specified error bound ε. We use a novel Voronoi subdivision method to evaluate $h(\mathcal{A},\mathcal{B})$ accurately (more details are given in Sec. 5). In summary, our algorithm to compute $h(\mathcal{A},\mathcal{B})$ consists of three steps as illustrated in Fig.2: (1) culling on model $\mathcal{A}$, (2) culling on model $\mathcal{B}$, and (3) polygon subdivision. We present the notation used throughout this paper in Table 1.

Notation	Meaning
$h(\mathcal{A},\mathcal{B})$	the one-sided Hausdorff distance from $\mathcal{A}$ to $\mathcal{B}$
$H(\mathcal{A},\mathcal{B})$	the two-sided Hausdorff distance between $\mathcal{A}$ and $\mathcal{B}$
$\overline{h}(\mathcal{A},\mathcal{B}), \underline{h}(\mathcal{A},\mathcal{B})$	the upper and lower bounds on $h(\mathcal{A},\mathcal{B})$
$\Delta^{\mathcal{A}}, \mathbf{v}^{\mathcal{A}}$	a triangle in model $\mathcal{A}$ and a vertex of $\Delta^{\mathcal{A}}$
$\overline{h}_i(\mathcal{A},\mathcal{B})$ $\underline{h}_i(\mathcal{A},\mathcal{B})$	the current upper and lower bounds of $h(\mathcal{A},\mathcal{B})$ after considering the triangles $\Delta_1^{\mathcal{A}}, \Delta_2^{\mathcal{A}} \cdots \Delta_i^{\mathcal{A}}$ in $\mathcal{A}$ (Eq.7)
$\overline{h}_i(\Delta^{\mathcal{A}},\mathcal{B})$ $\underline{h}_i(\Delta^{\mathcal{A}},\mathcal{B})$	the current upper and lower bounds of $h(\Delta^{\mathcal{A}},\mathcal{B})$ after considering the triangles $\Delta_1^{\mathcal{B}}, \Delta_2^{\mathcal{B}} \cdots \Delta_i^{\mathcal{B}}$ in $\mathcal{B}$ (Eq.12,13)
$d(\mathcal{A},\mathcal{B})$	the Euclidean distance between $\mathcal{A}$ and $\mathcal{B}$
$\mathtt{BV}(\Delta^{\mathcal{A}})$	a bounding volume that contains $\Delta^{\mathcal{A}}$

Table 1: ***Notation.***

4 Cross-Culling

The purpose of cross-culling is to conservatively to find a set of triangles $\Delta^{\mathcal{A}}$ in $\mathcal{A}$ that do not contribute to $h(\mathcal{A},\mathcal{B})$ (culling on $\mathcal{A}$), and a set of triangles $\Delta^{\mathcal{B}}$ in $\mathcal{B}$ that do not contribute to $h(\Delta^{\mathcal{A}},\mathcal{B})$ (culling on $\mathcal{B}$), where $\Delta^{\mathcal{A}}$ are the triangles that have survived the cull on $\mathcal{A}$.

4.1 Culling on Model $\mathcal{A}$

As explained in the previous section, some triangles $\Delta_j^{\mathcal{A}}$ need not be considered in the computation of $h(\Delta_j^{\mathcal{A}},B)$ if its upper bound $\overline{h}(\Delta_j^{\mathcal{A}},\mathcal{B})$ is less than the current global lower bound $\underline{h}_i(\mathcal{A},\mathcal{B})$. We can extend this culling to a set of triangles using a bounding volume.

Let $\mathtt{BV}(\Delta_j^{\mathcal{A}})$ be a volume that bounds a set of triangles $\Delta_j^{\mathcal{A}}$ (i.e. $\Delta_j^{\mathcal{A}} \subset \mathtt{BV}(\Delta_j^{\mathcal{A}})$). Using Lemma 1, we get

$$h(\Delta_j^{\mathcal{A}},\mathcal{B}) \le h\left(\mathtt{BV}(\Delta_j^{\mathcal{A}}),\mathcal{B}\right) \le h\left(\mathtt{BV}(\Delta_j^{\mathcal{A}}),\mathbf{q}\right), \tag{8}$$

where $\mathbf{q}$ is some point in $\mathcal{B}$. Thus, if $h(\mathtt{BV}(\Delta_j^{\mathcal{A}}),\mathbf{q})$ is less than the current lower bound $\underline{h}_i(\mathcal{A},\mathcal{B})$, we can cull all the triangles $\Delta_j^{\mathcal{A}}$ that are included in the bounding volume $\mathtt{BV}(\Delta_j^{\mathcal{A}})$. To be even more effective, this culling procedure can be executed hierarchically using

a standard bounding volume hierarchy (BVH) such as swept sphere volumes (SSV) [Larsen et al. 2000].

In fact, our choice of bounding volume is SSV. The SSV consists of PSS, LSS, and RSS which are respectively defined as the Minkowski sums of a point, a line, and a rectangle with a sphere (e.g. Fig.3). The reason for our choice of SSV is twofold. First, using SSV, one can efficiently calculate the Euclidean distances between polygon-soup models, and this operation is frequently called for in our Hausdorff distance computation. Second, Eq.8 can be evaluated simply. More precisely, we set $\mathbf{q}$ to be the point closest to $\mathtt{BV}(\Delta_j^{\mathcal{A}})$ in $\mathcal{B}$, and then $h(\mathtt{BV}(\Delta_j^{\mathcal{A}}), \mathbf{q})$ can be obtained simply by considering the Euclidean distance between $\mathbf{q}$ and the generator vertices that comprise the SSV. In the case of a rectangle swept sphere (RSS), for example, $h(\mathtt{BV}(\Delta_j^{\mathcal{A}}), \mathbf{q})$ is obtained as follows:

$$h\left(\mathtt{BV}(\Delta_j^{\mathcal{A}}), \mathbf{q}\right) = \max_{\mathbf{c}_i} \left(d\left(\mathbf{c}_i, \mathbf{q}\right)\right) + r, \tag{9}$$

where $\mathbf{c}_i, i = 1, \cdots, 4$ are the four vertices of the generator primitive rectangle of the RSS, and r is the radius of the swept sphere used to construct the RSS, as shown in Fig.3.

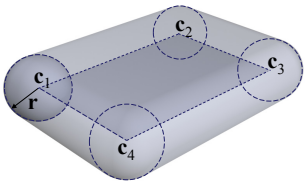


Figure 3: ***Rectangle Swept Sphere.***

The pseudo-code for this procedure is Alg.1. Once we have filtered out the triangles that have no chance of contributing to $h(\mathcal{A}, \mathcal{B})$, we attempt to find tight upper and lower bounds on the remaining triangles using Lemma 2. In the next section, we explain how we can calculate these bounds efficiently.

4.2 Culling on Model $\mathcal{B}$

The lower and upper bounds of $h(\Delta^{\mathcal{A}}, \mathcal{B})$ in Lemma 2 can be reformulated as follows:

$$\overline{h}\left(\Delta^{\mathcal{A}}, \mathcal{B}\right) = \min_{\Delta_{\mathcal{B}} \in \mathcal{B}} \left(\max_{\mathbf{v}^{\mathcal{A}} \in \Delta^{\mathcal{A}}} \left(d\left(\mathbf{v}^{\mathcal{A}}, \Delta^{\mathcal{B}}\right)\right)\right) \tag{10}$$

$$\underline{h}\left(\Delta^{\mathcal{A}}, \mathcal{B}\right) = \max_{\mathbf{v}^{\mathcal{A}} \in \Delta^{\mathcal{A}}} \left(\min_{\Delta^{\mathcal{B}} \in \mathcal{B}} \left(d\left(\mathbf{v}^{\mathcal{A}}, \Delta^{\mathcal{B}}\right)\right)\right). \tag{11}$$

Similarly, as in Sec. 3.2, we can define the *current* upper bound $\overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$ of $\overline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$, after considering the first i triangles in $\mathcal{B}$, as follows:

$$\overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B}) \equiv \min_{1 \le j \le i} \left(\max_{\mathbf{v}^{\mathcal{A}} \in \Delta^{\mathcal{A}}} \left(d\left(\mathbf{v}^{\mathcal{A}}, \Delta_j^{\mathcal{B}}\right)\right)\right). \tag{12}$$

As can be seen from Eq.10, we need to reduce $\overline{h}_i\left(\Delta^{\mathcal{A}}, \mathcal{B}\right)$ to get $\overline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$. Then the following inequality holds for a set of triangles $\Delta_j^{\mathcal{B}}$ in $\mathcal{B}$ and for its bounding volume $\mathtt{BV}(\Delta_j^{\mathcal{B}})$:

$$d\left(\Delta^{\mathcal{A}}, \mathtt{BV}\left(\Delta_j^{\mathcal{B}}\right)\right) \le d\left(\Delta^{\mathcal{A}}, \Delta_j^{\mathcal{B}}\right) \le \max_{\mathbf{v}^{\mathcal{A}} \in \Delta^{\mathcal{A}}} \left(d\left(\mathbf{v}^{\mathcal{A}}, \Delta_j^{\mathcal{B}}\right)\right).$$

Suppose $d(\Delta^{\mathcal{A}}, \mathtt{BV}(\Delta_j^{\mathcal{B}})) > \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$. Then, $\max_{\mathbf{v}^{\mathcal{A}} \in \Delta^{\mathcal{A}}}(d(\mathbf{v}^{\mathcal{A}}, \Delta_j^{\mathcal{B}})) > \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$. Thus, neither $\Delta_j^{\mathcal{B}}$ nor any triangle contained in $\mathtt{BV}(\Delta_j^{\mathcal{B}})$ can realize $\overline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$. Therefore, all the triangles in $\mathtt{BV}(\Delta_j^{\mathcal{B}})$ (including $\Delta_j^{\mathcal{B}}$) can be culled away.

Algorithm 1 CullingOnA
Input: $n_{\mathcal{A}}$: BVH node in $\mathcal{A}$, $n_{\mathcal{B}}.root$: BVH root node of $\mathcal{B}$, $\underline{h}_i(\mathcal{A}, \mathcal{B})$.
Output: $\overline{h}_i\left(\mathcal{A}, \mathcal{B}\right)$ and $\underline{h}_i\left(\mathcal{A}, \mathcal{B}\right)$.

if $n_{\mathcal{A}}$ is a leaf node **then**
  $\{\Delta^{\mathcal{A}}$:= the triangle included in $n_{\mathcal{A}}\}$
  $\left(\underline{h}(\Delta^{\mathcal{A}}, \mathcal{B}), \overline{h}(\Delta^{\mathcal{A}}, \mathcal{B})\right)$
  =**CullingOnB**$(\Delta^{\mathcal{A}}, n_{\mathcal{B}}.root, \infty, (\infty, \infty, \infty))$;
  if $\underline{h}(\Delta^{\mathcal{A}}, \mathcal{B}) > \underline{h}_i(\mathcal{A}, \mathcal{B})$ **then**
    $\underline{h}_i(\mathcal{A}, \mathcal{B}) = \underline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$;
  end if
  if $\overline{h}(\Delta^{\mathcal{A}}, \mathcal{B}) > \overline{h}_i(\mathcal{A}, \mathcal{B})$ **then**
    $\overline{h}_i(\mathcal{A}, \mathcal{B}) = \overline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$;
  end if
  return $\left(\overline{h}_i\left(\mathcal{A}, \mathcal{B}\right), \underline{h}_i\left(\mathcal{A}, \mathcal{B}\right)\right)$;
else
  $d_1 = h(n_{\mathcal{A}}.leftchild, \mathbf{q})$; {Using Eq.9}
  $d_2 = h(n_{\mathcal{A}}.rightchild, \mathbf{q})$; {Using Eq.9}
  if $d_1 > \underline{h}_i(\mathcal{A}, \mathcal{B})$ **then**
    $\left(\overline{h}_i\left(\mathcal{A}, \mathcal{B}\right), \underline{h}_i\left(\mathcal{A}, \mathcal{B}\right)\right)$
    =**CullingOnA**$(n_{\mathcal{A}}.leftchild, n_{\mathcal{B}}.root, \underline{h}_i(\mathcal{A}, \mathcal{B}))$;
  end if
  if $d_2 > \underline{h}_i(\mathcal{A}, \mathcal{B})$ **then**
    $\left(\overline{h}_i\left(\mathcal{A}, \mathcal{B}\right), \underline{h}_i\left(\mathcal{A}, \mathcal{B}\right)\right)$
    =**CullingOnA**$(n_{\mathcal{A}}.rightchild, n_{\mathcal{B}}.root, \underline{h}_i(\mathcal{A}, \mathcal{B}))$;
  end if
  return $\left(\overline{h}_i\left(\mathcal{A}, \mathcal{B}\right), \underline{h}_i\left(\mathcal{A}, \mathcal{B}\right)\right)$;
end if

Now, in Eq.11, to obtain $\underline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$, let us express the *current* lower bound $\underline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$ of $\underline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$, after considering ith triangle in $\mathcal{B}$ as follows:

$$\underline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B}) \equiv \max_{\mathbf{v}^{\mathcal{A}} \in \Delta^{\mathcal{A}}} \left(\min_{1 \le j \le i} \left(d\left(\mathbf{v}^{\mathcal{A}}, \Delta_j^{\mathcal{B}}\right)\right)\right). \tag{13}$$

We also need to reduce the value of $\underline{h}_i\left(\Delta^{\mathcal{A}}, \mathcal{B}\right)$ for a given triangle $\Delta^{\mathcal{A}}$ and its vertices $\mathbf{v}^{\mathcal{A}}$. Suppose $d\left(\Delta^{\mathcal{A}}, \mathtt{BV}\left(\Delta_j^{\mathcal{B}}\right)\right) > \overline{h}_i\left(\Delta^{\mathcal{A}}, \mathcal{B}\right)$, then $d\left(\Delta^{\mathcal{A}}, \mathtt{BV}\left(\Delta_j^{\mathcal{B}}\right)\right) > \underline{h}_i\left(\Delta^{\mathcal{A}}, \mathcal{B}\right)$ since $\underline{h}_i\left(\Delta^{\mathcal{A}}, \mathcal{B}\right) \le \overline{h}_i\left(\Delta^{\mathcal{A}}, \mathcal{B}\right)$. Thus, neither $\Delta_j^{\mathcal{B}}$ nor any triangle included in $\mathtt{BV}\left(\Delta_j^{\mathcal{B}}\right)$ can contribute to $\underline{h}\left(\Delta^{\mathcal{A}}, \mathcal{B}\right)$; so these triangles can be culled away.

In summary, for a given $\Delta^{\mathcal{A}}$, the inequality $d\left(\Delta^{\mathcal{A}}, \mathtt{BV}\left(\Delta_j^{\mathcal{B}}\right)\right) > \overline{h}_i\left(\Delta^{\mathcal{A}}, \mathcal{B}\right)$ is our culling condition on model $\mathcal{B}$ to find a set of triangles that contribute neither to $\overline{h}\left(\Delta^{\mathcal{A}}, \mathcal{B}\right)$ nor to $\underline{h}\left(\Delta^{\mathcal{A}}, \mathcal{B}\right)$.

4.2.1 Group Traversal

To compute $\overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$, for a given triangle $\Delta^{\mathcal{A}}$, the culling technique above requires the calculation of $d(\mathbf{v}^{\mathcal{A}}, \mathcal{B})$ for all three vertices $\mathbf{v}^{\mathcal{A}}$ in $\Delta^{\mathcal{A}}$, which in turn requires us to find the closest triangle in $\mathcal{B}$ to $\mathbf{v}^{\mathcal{A}}$, which we call $\Delta^{\mathcal{B}}$ (see Eq.12). However, repeated computation of $d(\mathbf{v}^{\mathcal{A}}, \mathcal{B})$ for different vertices of the same triangle $\Delta^{\mathcal{A}}$ is wasteful when several vertices in $\mathcal{A}$ are all closest to a set of nearby triangles in $\mathcal{B}$, since finding $d(\mathbf{v}^{\mathcal{A}}, \mathcal{B})$ requires a separate, top-down BVH traversal, as illustrated in Fig.4-(a). In order to speed up this computation, we partition and cluster the triangles in $\mathcal{B}$, and enclose each cluster with a bounding volume; in

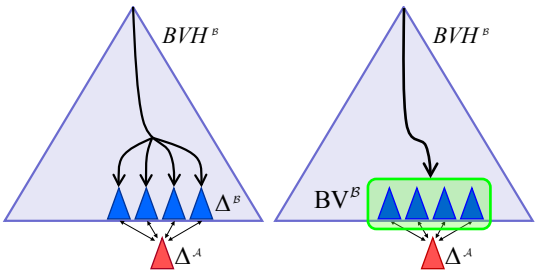
our implementation, we always put four or fewer triangles into one leaf-level bounding volume. Then, we perform a closest-distance query between $\Delta^{\mathcal{A}}$ and the clustered bounding volumes. Once we have found the minimal distance from $\Delta^{\mathcal{A}}$ to a leaf-level bounding volume $\mathtt{BV}^{\mathcal{B}}$, and if it is less than $\overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$, we test all possible pairwise combinations of $\forall \mathbf{v}^{\mathcal{A}} \in \Delta^{\mathcal{A}}$ and $\forall \Delta^{\mathcal{B}} \subset \mathtt{BV}^{\mathcal{B}}$ to find $d(\mathbf{v}^{\mathcal{A}}, \mathcal{B})$, as shown in Fig.4-(b). This group traversal concept is reminiscent of the ray-packet idea that uses BVH for ray-tracing [Wald et al. 2007]. The pseudo-code for culling on model $\mathcal{B}$ with group traversal is Alg.2.

Algorithm 2 CullingOnB
Input: $\Delta^{\mathcal{A}}$, $n_{\mathcal{B}}$: BVH node in $\mathcal{B}$, $\overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$, $\underline{\mathbf{d}}$: temp. vector for $\underline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$.
Output: $\underline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$, $\overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$.

{Group Traversal}
if # of triangles in $n_{\mathcal{B}} \leq 4$ **then**
  {Using Eq.10}
  $d_1 = \min_{\Delta^{\mathcal{B}} \in n_{\mathcal{B}}} \left(\max_{\mathbf{v}^{\mathcal{A}} \in \Delta^{\mathcal{A}}} \left(d\left(\mathbf{v}^{\mathcal{A}}, \Delta^{\mathcal{B}}\right)\right)\right)$;
  if $d_1 < \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$ **then**
    $\overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B}) = d_1$;
  end if
  {Using Eq.11}
  for all $\mathbf{v}^{\mathcal{A}} \in \mathcal{A}$ **do**
    $\mathbf{d}.\mathbf{v}^{\mathcal{A}} = \min_{\Delta^{\mathcal{B}} \in n_{\mathcal{B}}} \left(d\left(\mathbf{v}^{\mathcal{A}}, \Delta^{\mathcal{B}}\right)\right)$;
    if $\mathbf{d}.\mathbf{v}^{\mathcal{A}} < \underline{\mathbf{d}}.\mathbf{v}^{\mathcal{A}}$ **then**
      $\underline{\mathbf{d}}.\mathbf{v}^{\mathcal{A}} = \mathbf{d}.\mathbf{v}^{\mathcal{A}}$;
    end if
  end for
  $\underline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B}) = \max_{\mathbf{v}^{\mathcal{A}} \in \Delta^{\mathcal{A}}} \left(\underline{\mathbf{d}}.\mathbf{v}^{\mathcal{A}}\right)$;
  return $\left(\underline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B}), \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})\right)$;
else
  $d_1 = d(\Delta_{\mathcal{A}}, n_{\mathcal{B}}.leftchild)$;
  $d_2 = d(\Delta_{\mathcal{A}}, n_{\mathcal{B}}.rightchild)$;
  if $d_1 \leq \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$ **then**
    $\left(\underline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B}), \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})\right)$
    =**CullingOnB**$(\Delta_{\mathcal{A}}, n_{\mathcal{B}}.leftchild, \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B}), \underline{\mathbf{d}})$;
  end if
  if $d_2 \leq \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})$ **then**
    $\left(\underline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B}), \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})\right)$
    =**CullingOnB**$(\Delta_{\mathcal{A}}, n_{\mathcal{B}}.rightchild, \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B}), \underline{\mathbf{d}})$;
  end if
  return $\left(\underline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B}), \overline{h}_i(\Delta^{\mathcal{A}}, \mathcal{B})\right)$;
end if

5 Subdivision

Now we need to compute the Hausdorff distance $h(\Delta^{\mathcal{A}}, \mathcal{B})$ of those triangles $\Delta^{\mathcal{A}}$ in $\mathcal{A}$ which survived the cross-culling. In other words, if $\overline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$ for a triangle $\Delta^{\mathcal{A}}$ is greater than the global lower bound $\underline{h}(\mathcal{A}, \mathcal{B})$ in Eq.6, we need to evaluate $h(\Delta^{\mathcal{A}}, \mathcal{B})$. However, as we have already mentioned, since exact evaluation of $h(\Delta^{\mathcal{A}}, \mathcal{B})$ is costly, we approximate it within a user-specified error bound ε. The main idea is that we subdivide $\Delta^{\mathcal{A}}$ until the difference between the upper and lower bounds of the Hausdorff distance of $\Delta^{\mathcal{A}}$ is less than ε (i.e. $|\overline{h}(\Delta^{\mathcal{A}}, \mathcal{B}) - \underline{h}(\Delta^{\mathcal{A}}, \mathcal{B})| \leq \varepsilon$) or until the exact Hausdorff distance is obtained.



(a) Normal Traversal (b) Group Traversal

Figure 4: ***Group Traversal for BVH.*** *To compute $d(\mathbf{v}^{\mathcal{A}}, \mathcal{B})$ for all $\mathbf{v}^{\mathcal{A}} \in \Delta^{\mathcal{A}}$, (a) we can perform individual, top-down BVH traversal, or (b) a single top-down BVH group traversal for $\Delta^{\mathcal{A}}$ and the bounding volume $\mathtt{BV}^{\mathcal{B}}$.*

5.1 Algorithm

When we subdivide a triangle $\Delta^{\mathcal{A}}$ into four sub-triangles (a set of three triangles $\Delta_s^{\mathcal{A}}$ plus a single triangle $\Delta_c^{\mathcal{A}}$) as illustrated in Fig.5-(a), we can apply the culling and bound computation techniques, already explained in Sec. 4.1 and 4.2, to the sub-triangles. And, we can further optimize this process by utilizing the proximity information obtained during the bound computation for $\Delta^{\mathcal{A}}$, since we expect quite a high coherence between the bounds of $\Delta^{\mathcal{A}}$ and the triangles $\Delta_s^{\mathcal{A}}$ and $\Delta_c^{\mathcal{A}}$. The subdivision process is executed as follows:

1. Given $\Delta^{\mathcal{A}}$ and its upper bound $\overline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$, we first find a set of triangles $\mathtt{L}$ in $\mathcal{B}$ such that $\forall \Delta^{\mathcal{B}} \in \mathtt{L}, d\left(\Delta^{\mathcal{A}}, \Delta^{\mathcal{B}}\right) \leq \overline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$ using the BVH traversal. Note that the size of $\mathtt{L}$ is very small since $\overline{h}(\Delta^{\mathcal{A}}, \mathcal{B}) \simeq d(\Delta^{\mathcal{A}}, \mathcal{B})$.

2. The triangle $\Delta^{\mathcal{A}}$ is subdivided along each edge into the four sub-triangles $\Delta_s^{\mathcal{A}}$ and $\Delta_c^{\mathcal{A}}$ as illustrated in Fig.5-(a) (more details are in Sec. 5.2).

3. The upper and lower bounds of $\Delta_s^{\mathcal{A}}$ could be computed using Eqs.10 and 11; however, we only use the set $\mathtt{L}$ instead of the entire set $\mathcal{B}$ as explained below, and this process is quite efficient since $|\mathtt{L}| << |\mathcal{B}|$. Lemma 3 validates this process.

   - We first compute the bounds of the central triangle $\Delta_c^{\mathcal{A}}$ as follows:

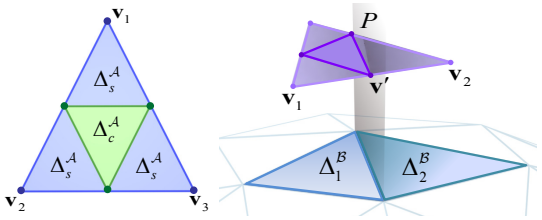
$$\underline{h}(\Delta_c^{\mathcal{A}}, \mathcal{B}) = \max_{\mathbf{v}^{\mathcal{A}} \in \Delta_c^{\mathcal{A}}} \left(\min_{\Delta^{\mathcal{B}} \in \mathtt{L}} \left(d\left(\mathbf{v}^{\mathcal{A}}, \Delta^{\mathcal{B}}\right)\right)\right)$$
$$\overline{h}(\Delta_c^{\mathcal{A}}, \mathcal{B}) = \min_{\Delta^{\mathcal{B}} \in \mathtt{L}} \left(\max_{\mathbf{v}^{\mathcal{A}} \in \Delta_c^{\mathcal{A}}} \left(d\left(\mathbf{v}^{\mathcal{A}}, \Delta^{\mathcal{B}}\right)\right)\right).$$

   - The bounds of the other three sub-triangles $\Delta_s^{\mathcal{A}}$ are obtained in a similar way to the above from the set $\mathtt{L}$. Note that the closest distance $d(\mathbf{v}^{\mathcal{A}}, \mathcal{B})$ from the vertices $\mathbf{v}^{\mathcal{A}}$ of the triangle $\Delta^{\mathcal{A}}$ to the model $\mathcal{B}$ can be reused for the bounds of the sub-triangles $\Delta_s^{\mathcal{A}}$, since the vertices $\mathbf{v}^{\mathcal{A}}$ are shared by $\Delta^{\mathcal{A}}$ and $\Delta_s^{\mathcal{A}}$.

4. We repeat steps of 2 and 3 until one of the following conditions is met:

   - If the triangles closest to all the vertices in $\Delta^{\mathcal{A}}$ are the same, then the subdivision terminates for $\Delta^{\mathcal{A}}$, since the upper and lower bounds are identical: i.e. $\underline{h}(\Delta^{\mathcal{A}}, \mathcal{B}) = \overline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$. This termination condition is also used by [Guthe et al. 2005]. In this case, we have the exact Hausdorff distance $h(\Delta^{\mathcal{A}}, \mathcal{B})$.

- The difference between the lower and upper bounds becomes less than the user-defined error bound: i.e. $|\overline{h}(\Delta^{\mathcal{A}}, \mathcal{B}) - \underline{h}(\Delta^{\mathcal{A}}, \mathcal{B})| \leq \varepsilon$.
- (*Optional*) If the models $\mathcal{A}$ and $\mathcal{B}$ are both closed, we check whether $\Delta^{\mathcal{A}}$ is enclosed by $\mathcal{B}$ when translated by $d(\mathbf{v}^{\mathcal{A}}, \mathcal{B}) + \varepsilon$ as described in Lemma 4 below and in Fig.6. If so, the user-defined accuracy ε has already been achieved by $h(\Delta^{\mathcal{A}}, \mathcal{B})$, and thus no more subdivision is required.



(a) Subdivided Triangle (b) Voronoi Subdivision

Figure 5: ***Voronoi Subdivision.*** *(a) The triangle $\Delta^{\mathcal{A}}$ is subdivided into four sub-triangles. (b) A new vertex $\mathbf{v}'$ on an edge $\overline{\mathbf{v}_1\mathbf{v}_2}$ is the intersection of the bisector surface P of $\Delta_1^{\mathcal{B}}, \Delta_2^{\mathcal{B}}$ and $\overline{\mathbf{v}_1\mathbf{v}_2}$.*

LEMMA 3
Given the subdivided triangle $\Delta_s^{\mathcal{A}} \subseteq \Delta^{\mathcal{A}}$, if $\overline{h}\left(\Delta_s^{\mathcal{A}}, \mathcal{B}\right) = d(\mathbf{v}_u^{\mathcal{A}}, \Delta_u^{\mathcal{B}})$ and $\underline{h}\left(\Delta_s^{\mathcal{A}}, \mathcal{B}\right) = d(\mathbf{v}_l^{\mathcal{A}}, \Delta_l^{\mathcal{B}})$ for some $\mathbf{v}_u^{\mathcal{A}}, \mathbf{v}_l^{\mathcal{A}} \in \Delta_s^{\mathcal{A}}$, then $\{\Delta_u^{\mathcal{B}}, \Delta_l^{\mathcal{B}}\} \subseteq \mathtt{L}$, where $\mathtt{L}$ is obtained in Step 1 [Tang and Kim 2009].

LEMMA 4
Let $\mathbf{v}^{\mathcal{A}}$ and $\Delta^{\mathcal{B}}$ respectively be the vertex in $\mathcal{A}$ and the triangle in $\mathcal{B}$ that realize $\underline{h}(\Delta^{\mathcal{A}}, \mathcal{B})$: i.e. $\underline{h}(\Delta^{\mathcal{A}}, \mathcal{B}) = d(\mathbf{v}^{\mathcal{A}}, \mathcal{B})$. Further, let us refer to the closest direction vector from $\mathbf{v}^{\mathcal{A}}$ to $\Delta^{\mathcal{B}}$ as $\mathbf{d}$, such that $||\mathbf{d}|| = d(\mathbf{v}^{\mathcal{A}}, \mathcal{B})$. Then, when we translate $\Delta^{\mathcal{A}}$ by $||\mathbf{d}|| + \varepsilon$ along $\mathbf{d}$, and if $\Delta^{\mathcal{A}}$ is completely enclosed by the model $\mathcal{B}$, the following inequality holds:

$$\forall \mathbf{p} \in \Delta_{\mathcal{A}},\ d(\mathbf{p}, \mathcal{B}) \leq \underline{h}\left(\Delta^{\mathcal{A}}, \mathcal{B}\right) + \varepsilon. \tag{14}$$

Thus, $h\left(\Delta^{\mathcal{A}}, \mathcal{B}\right) = \max_{\mathbf{p} \in \Delta_{\mathcal{A}}} d(\mathbf{p}, \mathcal{B}) \leq \underline{h}\left(\Delta^{\mathcal{A}}, \mathcal{B}\right) + \varepsilon$. [Tang and Kim 2009].

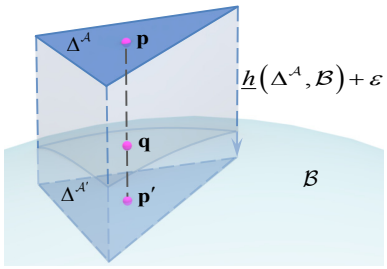


Figure 6: ***Termination Condition for Closed Models.*** *$\Delta^{\mathcal{A}}$ is enclosed by $\mathcal{B}$ when $\Delta^{\mathcal{A}}$ is translated by $\underline{h}(\Delta^{\mathcal{A}}, \mathcal{B}) + \varepsilon$.*

5.2 Voronoi Subdivision

One simple way to subdivide $\Delta^{\mathcal{A}}$ in Step 3 of the previous section is simply to bisect each edge in $\Delta^{\mathcal{A}}$ until the termination condition is met, as proposed by [Guthe et al. 2005]. However, one can employ a more efficient method with the aim of terminating the subdivision early. The main observation here is that when all the vertices in $\Delta^{\mathcal{A}}$ are projected into the same triangle in $\mathcal{B}$, no more subdivision is necessary. Thus, we want to add a new vertex along an edge such that this new vertex is projected into the same triangle as the previous vertices are projected.

More specifically, as shown in Fig.5-(b), let $\overline{\mathbf{v}_1\mathbf{v}_2}$ be an edge with end-vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ that needs to be subdivided, since $\mathbf{v}_1$ and $\mathbf{v}_2$ are projected into different triangles, namely $\Delta_1^{\mathcal{B}}$ and $\Delta_2^{\mathcal{B}}$, respectively. Then, we want to add a new vertex $\mathbf{v}'$ on the edge $\overline{\mathbf{v}_1\mathbf{v}_2}$ such that $\mathbf{v}'$ is equidistant from $\Delta_1^{\mathcal{B}}$ and $\Delta_2^{\mathcal{B}}$; i.e. $d(\mathbf{v}', \Delta_1^{\mathcal{B}}) = d(\mathbf{v}', \Delta_2^{\mathcal{B}})$. In other words, we want to find an intersection point between the bisector surface P (or Voronoi boundary) of $\Delta_1^{\mathcal{B}}$ and $\Delta_2^{\mathcal{B}}$ and the edge $\overline{\mathbf{v}_1\mathbf{v}_2}$. Since computing a bisector surface for triangles is relatively expensive, we conservatively find a vertex $\mathbf{v}'$ that is equidistant from the planes embedding $\Delta_1^{\mathcal{B}}$ and $\Delta_2^{\mathcal{B}}$, by solving a simple linear equation. We find this method works well in practice.

6 Two-sided Hausdorff Distance

So far, we have described an efficient method to compute $h(\mathcal{A}, \mathcal{B})$. A simple way to compute the two-sided Hausdorff distance $H(\mathcal{A}, \mathcal{B})$ is to calculate $h(\mathcal{A}, \mathcal{B})$ and $h(\mathcal{B}, \mathcal{A})$ independently and take their maximum. However, one can find $H(\mathcal{A}, \mathcal{B})$ more efficiently by exploiting the inter-dependency between the computation of $h(\mathcal{A}, \mathcal{B})$ and that of $h(\mathcal{B}, \mathcal{A})$. More specifically, we first compute $h(\mathcal{A}, \mathcal{B})$ using the technique explained in Secs. 4 and 5, and then use the following observations to accelerate the computation of $h(\mathcal{B}, \mathcal{A})$:

- Since $h(\mathcal{A}, \mathcal{B}) \leq H(\mathcal{A}, \mathcal{B})$, we can use $h(\mathcal{A}, \mathcal{B})$ to initialize the current lower bound $\underline{h}_0(\mathcal{B}, \mathcal{A})$ of $h(\mathcal{B}, \mathcal{A})$. This gives much better culling than the trivial lower bound $\underline{h}_0(\mathcal{B}, \mathcal{A}) = 0$ for the first cull of Sec. 4.1.
- In computing $h(\mathcal{A}, \mathcal{B})$, the second cull of Sec. 4.2 requires finding many closest triangles in $\mathcal{A}$ and $\mathcal{B}$. We can cache these pairs and reuse them in the second cull of $h(\mathcal{B}, \mathcal{A})$.

7 Results and Discussions

We will now present some results obtained with our algorithm, and compare its performance against other existing algorithm. We also explain how we can apply our algorithm to the computation of penetration depth. Finally, we explain some of the limitations of our algorithm.

7.1 Implementation and Benchmarks

We implemented our Hausdorff distance algorithm using C++ on a Windows XP PC, equipped with an Intel 2.6GHz CPU and 2.7GB of memory. Our implementation has not been fully optimized. We used modified code from the public-domain proximity library PQP [Larsen et al. 2000] for point/object distance calculations, and use the CGAL library[2] of graph data-structures for computing penetration depths. We benchmarked our Hausdorff distance algorithm using models of different complexities, ranging from 2.2K to 1.3M triangles, as shown in Fig.7. More specifically, we placed two models in space, initially separated by approximately three times their longest dimension. As we translated one of the models toward the other, we computed the two-sided Hausdorff distance between them, measured the timings and averaged them. Throughout the entire experiments, we set the user-specified bound ε as 10^{-4}. We repeated this test for different combinations of the models, as shown in Figs.1 and 8. The timing statistics as well as cross-culling

[2] `http://www.cgal.org`

Figure 7: ***Benchmarking Models.*** *From left to right (with triangle counts): Bunny1 (16.7K), Bunny2 (69.7K), Bird1 (2.2K), Bird2 (6.2K), Fish1 (11K), Fish2 (36.5K), Puma1 (8.2K), Puma2 (7.1K), Watch1 (3.8K), Watch2 (38.1K), Lion (1.3M), Dog (3.2K)*

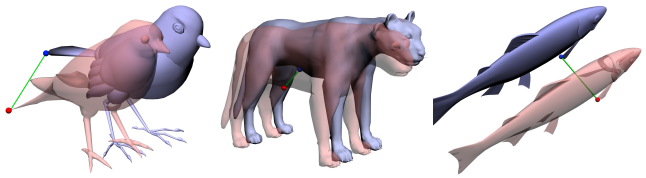


Figure 8: ***Hausdorff Distance Computation Results.*** *The green line denotes the Hausdorff distance result.*

rates are given in Table 2. As can be seen from the table, our algorithm takes only a fraction of a second, even for very complicated models, and cross-culling is effective, with a rate of over 99% for most benchmarks. This means that fewer than 1% of triangles need to be subdivided. We also observed that the maximum number of Voronoi subdivision is typically less than ten, and is sensitive to ε. In Fig.9, we show the timing of each step of our algorithm.

Benchmark	Timing ($msec$)			Culling Rates (%)	
	Avg.	Min.	Max.	On $\mathcal{A}$	On $\mathcal{B}$
Bird1/Bird2	2.57	0.10	6.37	99.27	99.42
Watch1/Watch2	6.92	1.50	31.31	99.93	97.60
Puma1/Puma2	6.10	1.63	21.28	99.18	98.43
Bunny1/Bunny2	21.95	0.18	948.40	97.12	99.77
Fish1/Fish2	18.77	1.24	69.22	99.59	99.60
Dog/Lion	84.31	0.04	670.60	98.03	99.99

Table 2: ***Performance of the Two-sided Hausdorff Distance Computation.***

7.2 Comparative Performance

We compared the performance of our algorithm against that due to [Guthe et al. 2005], which is known to be the fastest practical algorithm for polygonal models. For this comparison, we overlapped two static models such that their two-sided Hausdorff distance is less than 3% of the longest dimension of the models, and ran both algorithms. [Guthe et al. 2005]'s algorithm and ours share some similarities, but there are the following major differences:

- [Guthe et al. 2005] use octree data structures for proximity computations, whereas we rely on BVH. Thus our algorithm consumes much less memory for a similar result. Moreover, the current implementation of [Guthe et al. 2005]'s algorithm does not allow for arbitrary rigid transformation of objects.
- Our algorithm provides simple yet effective bounds on the Hausdorff distance.
- Our algorithm uses a novel Voronoi subdivision technique so that it produces fewer polygon subdivisions than [Guthe et al. 2005].

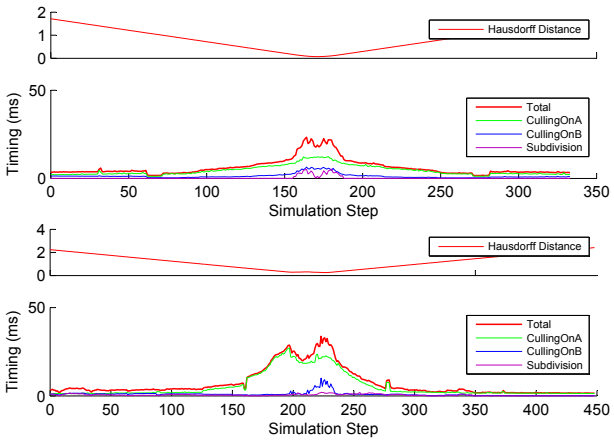


Figure 9: ***Timing Profiles.*** *Timings for culling on $\mathcal{A}$, culling on $\mathcal{B}$, and subdivision steps in our algorithm, as well as two-sided Hausdorff distance results, for the Puma1/Puma2 models (top two graphs) and Watch1/Watch2 models (bottom two graphs).*

As a result of these differences, our algorithm outperforms [Guthe et al. 2005] by more than an order of magnitude, as shown in Fig.10.

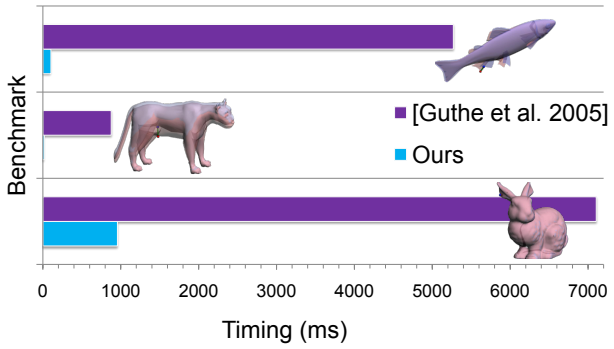


Figure 10: ***Performance Comparisons.*** *The performance was measured for severely overlapped models.*

7.3 Application to Penetration Depth Computation

A penetration depth (PD) measures the extent of the interpenetration of two objects overlapping in space [Kim et al. 2002]. A typical application of PD is the simulation of rigid and articulated-body dynamics for physically-based animation. Here, PD can be used to compute penalty forces in a penalty-based system [Moore and Wilhelms 1988] or to calculate collision impulses in an impulse-based system [Guendelman et al. 2003]. Moreover, in constraint-based

dynamics, in which the non-penetration constraint must be strictly imposed, PD is employed to reposition an object to satisfy that constraint [Redon 2004].

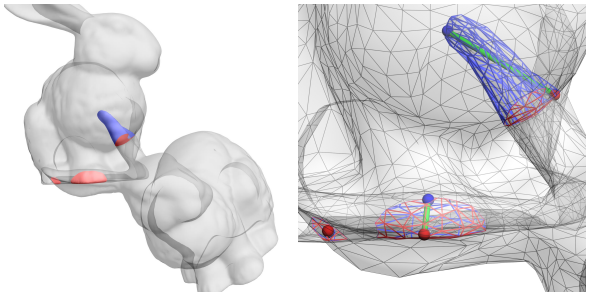
Various definitions of penetration depth are to be found in the literature [Zhang et al. 2007]. In many physically-based animation applications, PD is often defined as the point of deepest interpenetration of $\mathcal{A}$ and $\mathcal{B}$ [Guendelman et al. 2003]. We call this *pointwise* PD (or simply PD) and we use this definition in our work. As most known definitions of PD require inside/outside classifications with respect to the given models, the input models $\mathcal{A}$ and $\mathcal{B}$ must be closed manifolds. Note that this version of pointwise PD is much simpler than the configuration space-based formulation of [Kim et al. 2002; Zhang et al. 2007] in terms of computational and implementation complexity. Practical implementation of pointwise PD is based on the use of distance fields [Fisher and Lin 2001; Guendelman et al. 2003]. However, due to the high memory consumption of distance fields, some game physics engines no longer use this type of implementation [NVIDIA 2009]. Our PD algorithm uses a more compact geometric representation based on BVH and the Hausdorff distance computation.

PD Algorithm Given two overlapping, closed models $\mathcal{A}$ and $\mathcal{B}$, our PD algorithm consists of two stages as follows:

1. Compute the volume $\mathcal{V}$ of the intersection of $\mathcal{A}$ and $\mathcal{B}$. Then the portions of the boundary surfaces of $\mathcal{A}$ and $\mathcal{B}$ that belong to $\mathcal{V}$ are denoted as $\partial\mathcal{V}_\mathcal{A}$ and $\partial\mathcal{V}_\mathcal{B}$, respectively: see Fig.11-(a).

2. Compute the two-sided Hausdorff distance $H(\partial\mathcal{V}_\mathcal{A}, \partial\mathcal{V}_\mathcal{B})$ and report it as the pointwise PD of $\mathcal{A}$ and $\mathcal{B}$: see Fig.11-(b).

Step 1 of the above algorithm can be implemented by using an existing collision detection algorithm such as the one in the PQP library to find intersection curves. Then, using the intersecting curves as seed points, a simple flood-filling algorithm can be used to find the vertices and triangles of $\mathcal{A}$ and $\mathcal{B}$ that are inside $\mathcal{V}$. A similar approach has been proposed by [Hippmann 2004] for Step 1.



(a) Intersection volume (b) PD results

Figure 11: ***Penetration Depth Computation.*** *The surfaces of intersection volume $\mathcal{V}$ are highlighted as blue ($\partial\mathcal{V}_\mathcal{A}$) and red ($\partial\mathcal{V}_\mathcal{B}$) in (a) and (b). In this case, there are three disjoint, intersection volumes. For each intersection volume, the PD result $H(\partial\mathcal{V}_\mathcal{A}, \partial\mathcal{V}_\mathcal{B})$ is displayed as a green line and the deepest penetrating points are shown as blue and red spheres in (b).*

Performance We measured the performance of our PD algorithm on two complicated dynamic scenes: Alphabet Domino and Falling Bunnies (see Fig.1 and the supplemental video). For the Alphabet Domino scene, we physically simulated the dominos consisting of 12 alphabet characters, each composed of around 200 triangles, and a bunny with 1K triangles. For Falling Bunnies, we dropped 50 bunnies, each consisting of 1K triangles, on top of each other. In both cases, we used the Havok™ software[3] to simulate the dynamics. Then, we plugged our algorithm into the dynamic sequences and measured its average performance. The Alphabet Domino and 50 Falling Bunnies benchmarks respectively required only 0.97 msec and 3.88 msec on average for the PD computations.

7.4 Limitations

There are a few limitations to our algorithm: it is not exact, even though it can compute Hausdorff distance within a user-specified accuracy and its performance is sensitive to the relative configuration of objects. More specifically, it takes more time as the Hausdorff distance becomes smaller. This phenomenon has been also observed by [Llanas 2005]. Our penetration depth algorithm is limited to closed models with well-defined insides and outsides.

8 Conclusions

We have presented an efficient algorithms to compute the Hausdorff distance between polygonal models. The main idea of our algorithm is the use of cross-culling on the input models using tight distance bounds and Voronoi polygon subdivision. Our algorithm shows interactive performance for models of practical size. We have also applied our Hausdorff distance algorithm to computing penetration depth for physically-based animation.

Future Work We would like to extend our algorithm to handle dynamically changing models such as deformable or breaking objects. The bound calculation presented in the paper would remain valid for such models, but dynamic BVH updating would be required to implement an efficient culling procedure. Devising an incremental Hausdorff distance algorithm for a highly coherent environment would be an interesting project. And it would also be highly desirable to be able to apply our algorithm to the computation of Hausdorff distance under motion. We are also interested in extending our PD framework to deformable and polygon-soup models. Applying the PD algorithm to robot motion planning would also be worthwhile.

Acknowledgements

This research was supported in part by the IT R&D program of MKE/MCST/IITA (2008-F-033-02, Development of Real-time Physics Simulation Engine for e-Entertainment) and the KRF grant (KRF-2007-331-D00400). We thank Michael Guthe for sharing his Hausdorff distance codes with us.

References

AGARWAL, P. K., S.HAR-PELED, SHARIR, M., AND WANG, Y. 2003. Hausdorff distance under translation for points, disks, and balls. In *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, 282–291.

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2003. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9*, 1, 3–15.

ALT, H., AND GUIBAS, L. J. 2000. Discrete geometric shapes: Matching, interpolation, and approximation. In *Handbook of*

[3] http://www.havok.com/

Computational Geometry, J.-R. Sack and J. Urrutia, Eds. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 121–153.

ALT, H., BEHRENDS, B., AND BLÖMER, J. 1995. Approximate matching of polygonal shapes. *Ann. Math. Artif. Intell. 13*, 251–266.

ALT, H., BRASS, P., GODAU, M., KNAUER, C., AND WENK, C. 2003. Computing the Hausdorff distance of geometric patterns and shapes. In *Discrete and Computational Geometry.*, vol. 25. 65–76.

ASPERT, N., SANTA-CRUZ, D., AND EBRAHIMI, T. 2002. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, 705 – 708.

ATALLAH, M. J. 1983. A linear time algorithm for the Hausdorff distance between convex polygons. *Inf. Process. Lett. 17*, 207–209.

CHEW, L. P., GOODRICH, M. T., HUTTENLOCHER, D. P., KEDEM, K., KLEINBERG, J. M., AND KRAVETS, D. 1997. Geometric pattern matching under Euclidean motion. *Computational Geometry: Theory and Applications 7*, 113–124.

CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 1998. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2, 167–174.

COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., BROOKS, F., AND WRIGHT, W. 1996. Simplification envelopes. In *Proc. of ACM Siggraph'96*, 119–128.

EHMANN, S., AND LIN, M. C. 2001. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001) 20*, 3, 500–510.

FISHER, S., AND LIN, M. C. 2001. Fast penetration depth estimation for elastic bodies using deformed distance fields. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 2001*.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proc. of ACM SIGGRAPH*, 209–216.

GODAU, M. 1998. *On the complexity of measuring the similarity between geometric objects in higher dimensions*. PhD thesis, Freie Universität.

GOODRICH, M. T., MITCHELL, J. S. B., AND ORLETSKY, M. W. 1999. Approximate geometric pattern matching under rigid motions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 21*, 37–9.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Nonconvex rigid bodies with stacking. In *ACM SIGGRAPH*, ACM, New York, NY, USA, 871–878.

GUTHE, M., BORODIN, P., AND KLEIN, R. 2005. Fast and accurate Hausdorff distance calculation between meshes. In *International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, 41–48.

HIPPMANN, G. 2004. An algorithm for compliant contact between complexly shaped bodies. *Multibody System Dynamics 12*, 4.

HUTTENLOCHER, D. P., KEDEM, K., AND KLEINBERG, J. M. 1992. On dynamic Voronoi diagrams and the minimum Hausdorff distance for point sets under Euclidean motion in the plane. In *ACM Symposium on Computational Geometry*, 110–119.

HUTTENLOCHER, D. P., KEDEM, K., AND SHARIR, M. 1993. The upper envelope of Voronoi surfaces and its applications. *Discrete and Computational Geometry 9*, 267–291.

HUTTENLOCHER, D. P., KLANDERMAN, G. A., AND RUCKLIDGE, W. J. 1993. Comparing images using the Hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell. 15*, 850–863.

JESORSKY, O., KIRCHBERG, K., FRISCHHOLZ, R., ET AL. 2001. Robust face detection using the Hausdorff distance. *Lecture Notes in Computer Science*, 90–95.

KETTNER, L., MEHLHORN, K., PION, S., SCHIRRA, S., AND YAP, C. 2008. Classroom examples of robustness problems in geometric computations. *Comput. Geom. Theory Appl. 40*, 1, 61–78.

KIM, Y. J., OTADUY, M. A., LIN, M. C., AND MANOCHA, D. 2002. Fast penetration depth computation for physically-based animation. *Proc. of ACM Symposium on Computer Animation*.

LARSEN, E., GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 2000. Distance queries with rectangular swept sphere volumes. *Proc. of IEEE Int. Conference on Robotics and Automation*.

LIN, M., AND MANOCHA, D. 2003. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*.

LLANAS, B. 2005. Efficient computation of the Hausdorff distance between polytopes by exterior random covering. *Comput. Optim. Appl. 30*, 2, 161–194.

LOPEZ, M. A., AND REISNER, S. 2008. Hausdorff approximation of 3D convex polytopes. *Inf. Process. Lett. 107*, 2, 76–82.

LUEBKE, D., WATSON, B., COHEN, J., REDDY, M., AND VARSHNEY, A. 2002. *Level of detail for 3D graphics*. Elsevier Science Inc. New York, NY, USA.

MOORE, M., AND WILHELMS, J. 1988. Collision detection and response for computer animation. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, J. Dill, Ed., vol. 22, 289–298.

NVIDIA. 2009. *PhysX*. http://www.nvidia.com.

REDON, S. 2004. Fast continuous collision detection and handling for desktop virtual prototyping. *Virtual Real. 8*, 1, 63–70.

RUCKLIDGE, W. J. 1996. Lower bounds for the complexity of the graph of the Hausdorff distance as a function of transformation. *Discrete and Computational Geometry 16*, 2.

TANG, M., AND KIM, Y. J. 2009. Deriving upper and lower bounds of Hausdorff distance for polygonal models. Tech. rep. *CSE-TR-2009-01*, Ewha Womans University, Seoul, Korea.

VARADHAN, G., AND MANOCHA, D. 2004. Accurate Minkowski sum approximation of polyhedral models. In *Pacific Graphics*, 392–401.

WALD, I., BOULOS, S., AND SHIRLEY, P. 2007. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph. 26*, 1, 6.

ZHANG, L., KIM, Y. J., VARADHAN, G., AND MANOCHA, D. 2007. Generalized penetration depth computation. *Computer-Aided Design 39*, 8, 625 – 638.

ZITOVÁ, B., AND FLUSSER, J. 2003. Image registration methods: a survey. *Image and Vision Computing 21*, 11, 977–1000.