Computer-Aided Design 43 (2011) 1370–1379

Contents lists available at SciVerse ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

GPU-accelerated Hausdorff distance computation between dynamic deformable NURBS surfaces

Adarsh Krishnamurthy [a,*], Sara McMains [a], Iddo Hanniel [b]

[a] University of California, Berkeley, Berkeley, USA

[b] Technion, Israel Institute of Technology, Haifa, Israel

ARTICLE INFO

Keywords:
Hausdorff distance
NURBS
GPU
Geometric algorithms

ABSTRACT

We present a parallel GPU-accelerated algorithm for computing the directed Hausdorff distance from one NURBS surface to another, within a bound. We make use of axis-aligned bounding-box hierarchies that bound the NURBS surfaces to accelerate the computations. We dynamically construct as well as traverse the bounding-box hierarchies for the NURBS surfaces using operations that are optimized for the GPU. To compute the Hausdorff distance, we traverse this hierarchy after culling bounding-box pairs that do not contribute to the Hausdorff distance. Our contribution includes two-sided culling tests that can be performed in parallel using the GPU. The culling, based on the minimum and maximum distance ranges between the bounding boxes, eliminates bounding-box pairs from both surfaces that do not contribute to the Hausdorff distance simultaneously. We calculate accuracy bounds for our computed Hausdorff distance based on the curvature of the surfaces. Our algorithm runs in real-time with very small guaranteed error bounds for complex NURBS surfaces. Since we dynamically construct our bounding-box hierarchy, our algorithm can be used to interactively compute the Hausdorff distance for models made of dynamic deformable surfaces.

1. Introduction

The Hausdorff distance is a useful measure of the similarity between geometric objects. There are many applications that benefit from an efficient computation of the Hausdorff distance, including shape matching [1], mesh simplification [2], geometric approximation [3], and penetration depth calculation for physically based animation [4]. As a result, the Hausdorff distance computation has attracted considerable research attention in computer graphics, computational geometry, and geometric modeling. However, most previous work focused on computing the Hausdorff distance for polygons and polygonal meshes.

This paper introduces a GPU-accelerated algorithm for the computation of the Hausdorff distance for NURBS surfaces, which remain the de facto standard for CAD models. Our algorithm runs in real-time with very small guaranteed error bounds for complex NURBS surfaces. Since we do not rely on preprocessing the input, our algorithm can be used to interactively compute the Hausdorff distance for models made of dynamic deformable surfaces.

1.1. Contributions

In this paper, we introduce a GPU-accelerated algorithm for computing the directed Hausdorff distance from one NURBS surface to another. Our main contributions include:

- A GPU-accelerated Hausdorff distance computation for NURBS surfaces. We perform GPU traversal of a bounding-box hierarchy to compute the Hausdorff distance after selectively culling parts of the surface that do not contribute to the Hausdorff distance. To the best of our knowledge, this is the first completely implemented algorithm to compute the Hausdorff distance for pairs of complex curved surfaces without requiring pre-tessellation.
- Novel culling tests that cull bounding-box pairs that do not contribute to the Hausdorff distance based on the range of minimum and maximum distance between the pairs. We perform two culling tests that bound both the minimum and maximum value of the Hausdorff distance for two-sided culling.
- Ability to efficiently compute a tight range for the Hausdorff distance and also the locations on the surfaces where the Hausdorff distance is within this bound.
- Theoretical bounds for the Hausdorff distance computations. We can guarantee user-defined tolerance values based on curvature-based bounds on the NURBS surfaces.
- Interactive computation of the Hausdorff distance between models made of dynamic deformable surfaces.

* Corresponding author. Tel.: +1 510 590 7325.
E-mail addresses: adarshk@berkeley.edu, adkrishnamurthy@ucsd.edu (A. Krishnamurthy), mcmains@me.berkeley.com (S. McMains), ihanniel@technion.ac.il (I. Hanniel).

0010-4485/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2011.08.022

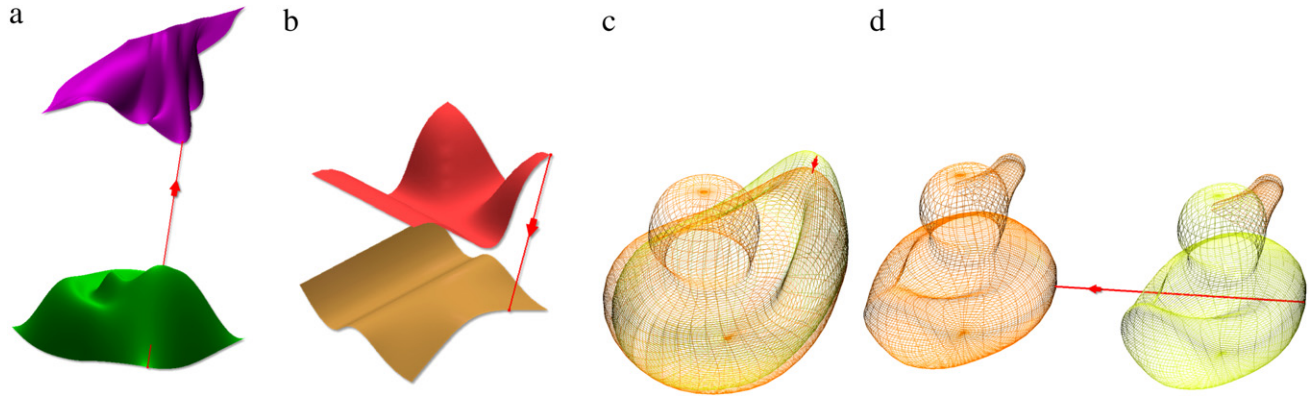


Fig. 1. Computing the Hausdorff distance for NURBS surfaces; the arrows connect the locations on the two models where the Hausdorff distance is achieved. The tail of the arrow indicates the surface over which the minimum distances are maximized.

2. Related work

The Hausdorff distance computation has attracted considerable research attention in computer graphics, computational geometry, and geometric modeling. Atallah [5] developed a linear time algorithm for computing the Hausdorff distance between planar convex polygons. For simple non-convex polygons with m and n vertices, Alt et al. [1] presented an algorithm, which is based on Voronoi diagrams, with $O((n+m)\log(n+m))$ running time.

In $\Re^3$, for polygonal meshes with $O(n)$ triangles, Alt et al. [6] presented theoretical and randomized algorithms. Their deterministic algorithm (a special case of a general theorem they prove for $\Re^d$) runs in $O(n^5)$ asymptotic time. Using a randomized lower envelope algorithm, they arrive at an $O(n^{3+\epsilon})$ randomized algorithm, and with minor additional requirements on the input they achieved a sub-cubic randomized algorithm. These algorithms are based on sophisticated data structures and algorithms and therefore are of a theoretical nature. To the best of our knowledge, they have never been implemented. Recently, Bartoň et al. [7] presented an $O(n^4 \log n)$ deterministic algorithm for computing the precise (up to floating point) Hausdorff distance between polygonal meshes. However, the high runtime complexity makes their implementation too slow (seconds) for large meshes.

Due to the complexity of computing the Hausdorff distance exactly, approximate practical algorithms have been proposed. Llanas [8] proposed two approximate algorithms for non-convex polytopes, based on random covering. Guthe et al. [2] proposed an approximate algorithm for meshes that finds high-distance regions by performing an octree search. These regions are then further subdivided and regions that cannot attain the Hausdorff distance are purged away.

In a recent paper, Tang et al. [4] implemented an approximate algorithm, which is similarly based on a Bounding Volume Hierarchy (BVH) that is computed during preprocessing. Using the BVH, many of the triangles are culled away based on upper and lower bounds that are updated as the algorithm progresses. The triangles that pass the culling are further subdivided until the difference between the upper and lower bound on the Hausdorff distance is smaller than a user-defined tolerance. This results in an approximate distance, which is within the user-defined tolerance of the actual Hausdorff distance between the two meshes. Their implementation is very fast in practice, running at interactive speed for meshes of practical size.

On computing the Hausdorff distance between freeform surfaces, there are fewer previous results. For freeform curves, Alt and Scharf [9,10] presented an algorithm for the Hausdorff distance computation, based on a characterization of the possible points where the distance can be attained. Chen et al. [11] presented an algorithm for computing the Hausdorff distance between two B-Spline curves. The algorithm improves the one from Alt and Scharf [10] by using a pruning technique to save computation time. Kim et al. [12] presented a real-time algorithm for planar curves. The algorithm assumes that the planar curves are approximated by biarcs in a preprocessing stage and makes use of the GPU depth buffer to detect high distance regions efficiently. Recently, Bai et al. [13] have computed an approximate Hausdorff distance between planar free-form curves by approximating the input curves with polylines and then computing the Hausdorff distance between the line segments.

Elber and Grandine [14] extended the results from Alt and Scharf [9] to Hausdorff distance computations, not only between planar curves, but also between curves and surfaces in $\Re^3$. The performance of their algorithm is quite efficient for the case of planar/space curves. However, for the Hausdorff distance between a freeform surface and another curve or surface, the algorithm takes several seconds. To the best of our knowledge, this is the only published previous work that discusses the problem of this paper.

3. Overview

In this section, we first mathematically define the Hausdorff distance between two sets in $\Re^3$. Subsequently, we present an overview of our algorithm highlighting the different steps. We then explain the steps in detail in the following sections.

3.1. Mathematical preliminaries

Definition 1 (*Hausdorff Distance*)**.** Given two compact sets $\mathcal{A}$ and $\mathcal{B}$ in $\Re^3$, the Hausdorff distance from $\mathcal{A}$ to $\mathcal{B}$ is given by Eq. (1), where $\mathbf{d}(.,.)$ is the Euclidean distance in $\Re^3$.

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

Sometimes this anti-symmetric Hausdorff distance is referred to as the "directed" or "one-sided" Hausdorff distance, but we will simply refer to it as the Hausdorff distance for the remainder of this paper.

3.2. Algorithm overview

Our algorithm to find the Hausdorff distance consists of three steps. We first construct bounding-box hierarchies (Section 4) that enclose sub-patches of the NURBS surfaces using the GPU (Fig. 2). If the surfaces are not changing, this step can be performed during initialization. In the second step, we traverse this hierarchy on the GPU, building a virtual 2D array of min–max distance ranges

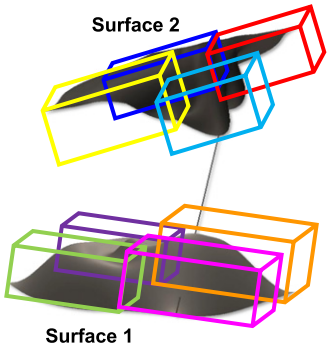


Fig. 2. Surface bounding-boxes constructed at the coarsest level of the hierarchy constructed for two sample input surfaces.

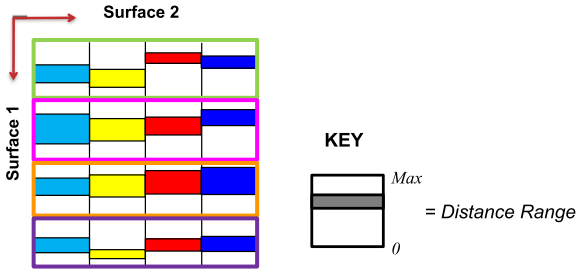


Fig. 3. Virtual 2D representation of the min-max distances between the bounding-box pairs. The colors correspond to the bounding-boxes in Fig. 2.

for pairs of bounding-boxes from the two input surfaces (Fig. 3), starting at the coarsest level of the hierarchy. At each level, we perform two tests to cull pairs that could not contain sub-patches potentially contributing to the Hausdorff distance (Section 5.1). For the first test, we calculate per-row Hausdorff distance maximums and cull ranges with a minimum distance value greater than the corresponding maximum; for the second test, we calculate a global Hausdorff distance minimum and cull rows containing ranges with a maximum distance value less than this minimum. An efficient implementation of this hierarchical culling with virtual array packing is detailed in Section 5.2. Finally, in the last step, we compute the Hausdorff distance between the surface patches that lie inside the potential bounding-box pairs in the finest level of the hierarchy.

4. Hierarchical bounding boxes for NURBS

We employ an axis-aligned bounding-box (AABB) hierarchy for the NURBS surfaces as an accelerating data structure to compute the Hausdorff distance using the GPU. We chose AABBs over a more complex BVH such as spherical shells or oriented bounding boxes (OBBs) [15] because the efficiency of GPU programs can be reduced dramatically with increases in the complexity of the parallel kernels that are used. The individual computational kernels for OBBs are more complex and contain many branching conditions; the GPU typically must wait until the most computationally intensive branch of the kernel in a particular block is completed before proceeding to the next block. In addition, since OBB kernels make use of more temporary registers, the number of computations that can be active ("in flight") simultaneously on the GPU is reduced; it may be difficult to hide the memory access latency in this case. Thus, we found that the advantage provided by tight OBBs might be offset by the increase in complexity of the kernels that use them.

We evaluate the NURBS surface on the GPU in a uniformly sampled regular grid in parametric space and then construct AABBs

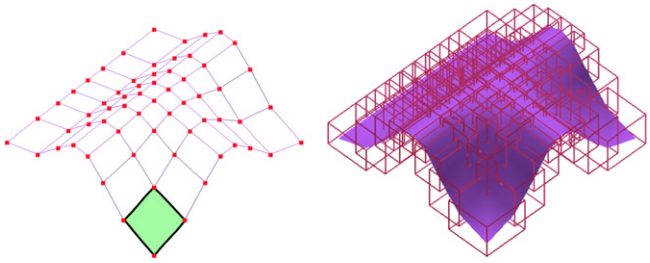


Fig. 4. Surface bounding-boxes constructed from points evaluated on a NURBS surface.

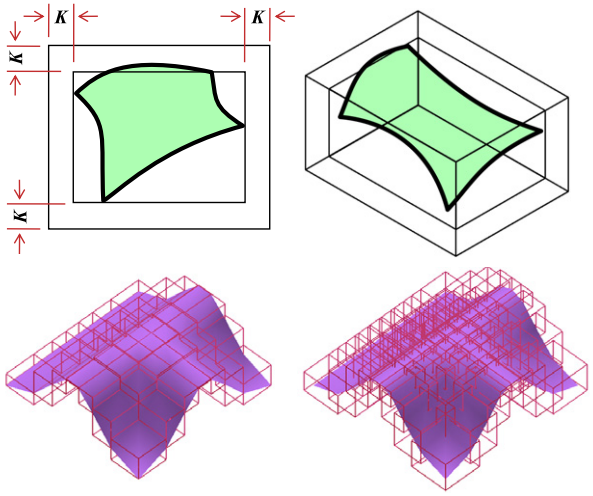


Fig. 5. We expand the AABBs by K in all three dimensions to guarantee that the surface patch is completely enclosed.

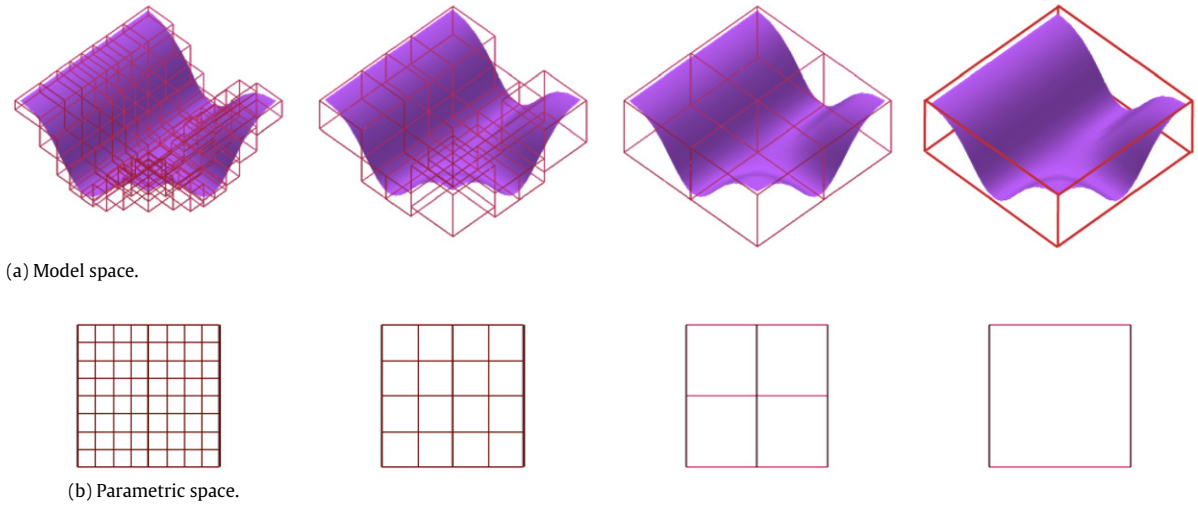
that enclose the surface sub-patches that lie between each group of four adjacent surface points, similar to the method explained in Krishnamurthy et al. [16] (Fig. 4). We used an optimized CUDA implementation of the surface point evaluation to evaluate the NURBS surface. (An alternate GPU NURBS evaluation method such as the one introduced by Kanai [17] could also be used, but that approach requires writing different GPU kernels for different-order surfaces.)

Since the surface patches enclosed by the bounding boxes are curved, some part of the surface might penetrate out of the bounding box if we were to merely use the coordinates of the four evaluation points to construct the bounding boxes (see Fig. 5). To guarantee that the bounding boxes enclose their surface patches, we expand the bounding boxes based on the curvature of the surface patch. Using this method, the bounding boxes automatically become tighter when we evaluate the surface at a finer resolution.

The analytical expression for the factor that can be used to expand the bounding boxes based on the surface curvature was originally given by Filip et al. [18]. They show that if a parametric C^2 surface $\mathbf{S}$ is evaluated at $(n+1) \times (m+1)$ grid points, the deviation of the surface from the piecewise linear approximation cannot exceed a constant K, which is a function of the magnitude of the maximum curvature vector on the surface, as defined by Eqs. (2)–(5). We use this K to derive bounds for our computed Hausdorff distance.

$$K = \frac{1}{8}\left(\frac{1}{n^2}M_1 + \frac{2}{nm}M_2 + \frac{1}{m^2}M_3\right) \tag{2}$$

$$M_1 = \max_{\forall(u,v)} \left\| \frac{\partial^2 \mathbf{S}}{\partial u^2} \right\| \tag{3}$$



(a) Model space.

(b) Parametric space.

Fig. 6. We construct the bounding-box hierarchy by recursively combining four bounding boxes from the finer level to get the next coarser level.

$$M_2 = \max_{\forall(u,v)} \left\| \frac{\partial^2 \mathbf{S}}{\partial u \partial v} \right\| \tag{4}$$

$$M_3 = \max_{\forall(u,v)} \left\| \frac{\partial^2 \mathbf{S}}{\partial v^2} \right\|. \tag{5}$$

Once we construct the bounding boxes at the finest level of the hierarchy, we use a bottom-up approach to construct the BVH (Fig. 6). We construct the bounding-box hierarchy by recursively combining four adjacent bounding boxes at a finer level to get the next coarser level. We repeat this operation until we get a single zeroth-level bounding box that encloses the whole surface. This bottom-up approach to hierarchy construction allows the use of the tighter curvature bounds from the densest sampling throughout the hierarchy. Lauterbach et al. [19,20] have recently developed a GPU algorithm where the BVH is constructed top-down, but a top-down approach is not as attractive for freeforms. Our approach allows using tight bounds since in a top-down method, one can only calculate very loose bounds because estimated local-curvature accuracy can only be refined after more evaluation points are available; thus the initial higher-level bounding boxes cannot be as tight as with a bottom-up approach.

5. Hierarchy traversal on the GPU

Once we have a bounding box hierarchy, we traverse this hierarchy on the GPU by selectively culling bounding-box pairs that cannot contain the surface sub-patches corresponding to the Hausdorff distance. To aid in the culling process, we allocate another array on the GPU that we will refer to as the min–max "distance matrix" that contains the distance ranges between pairs of bounding boxes, calculating the distance ranges in parallel using the GPU-friendly approach we had previously described [16]. If the bounding boxes overlap, we take the minimum distance between them as zero. We store the minimum and maximum distance in the global memory of the GPU, since the sizes of these arrays can be larger than what can fit into shared memory. We then use these min–max distances for computing the Hausdorff distance.

5.1. Culling tests

Bounding box hierarchy culling for Hausdorff distances is more complex than for traditional distance metrics because of the presence of both a minimum and a maximum in the formula

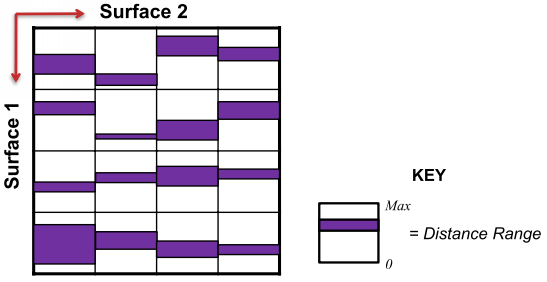
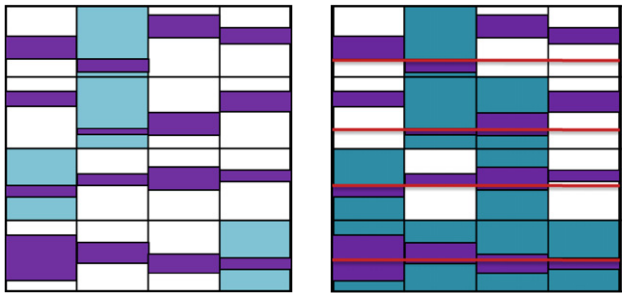


Fig. 7. The minimum and maximum distance ranges for the bounding-box pairs. The range of distance is represented by the colored regions for each pair.

(Eq. (1)), which must be coordinated with the distance ranges for the bounding boxes. We will explain the culling portion of our algorithm using as a concrete example the level 1 (top) of the hierarchy, containing 4 bounding boxes from each surface (as illustrated in Fig. 2). We compute the min–max distances for all 16 pairs of bounding boxes and store them as a 4×4 distance matrix (Fig. 7 for this example). In this matrix, as in Fig. 3, each row corresponds to distance ranges between a *surface* 1 sub-patch's bounding box and the bounding boxes of the different sub-patches of *surface* 2; similarly, each column contains distance ranges between a *surface* 2 sub-patch's bounding box and the bounding boxes of the different sub-patches of *surface* 1. Thus, each entry is the distance range for a pair of bounding boxes. If the entries of the matrix were exact distance values instead of ranges, to find the Hausdorff distance we would only need to find the minimum value for each row and then find the maximum of these minimum values. However, since the entries are ranges, we cull those ranges that cannot contribute to the Hausdorff distance. We apply two culling tests to cull the bounding-box pairs that cannot contribute to the Hausdorff distance.

Keeping in mind that the Hausdorff distance is the maximum of the minimum distances from points on *surface* 1, we first cull based on the bound on the maximum possible value of the minimum distance from each *surface* 1 bounding box, which corresponds to the maximum possible minimum distance per row. Since the actual minimum distance between a pair of bounding boxes can be anywhere in the range pictured, we can only exclude those bounding-box pairs whose range lies completely above the smallest maximum value in a row. To perform this culling, we first find the smallest maximum value in each row as shown by the shaded boxes in Fig. 8(a). We then use this value as a cut-off and cull all the bounding-box pairs whose minimum value is greater



(a) Smallest max distance in each row.

(b) Ranges having min value less than smallest max distance in a row are not culled.

Fig. 8. The first culling test.

than the smallest maximum value in each row. In Fig. 8(b) the candidate bounding-box pairs that are not culled for the example case are shaded.

The second culling test is based on a global bound on the minimum value of the Hausdorff distance. We calculate this bound by first finding the smallest minimum distance in each row (Fig. 9(a). We then find the largest of these smallest minimum distance values as shown in Fig. 9(b). This value gives a lower bound on the Hausdorff distance. We use this bound to find all bounding-box pairs whose range lies completely below this lower bound value (Fig. 9(c)). Such ranges indicate that the *surface* 1 bounding-box corresponding to that row has a smaller minimum distance to *surface* 2 than the Hausdorff distance lower bound calculated in Fig. 9(b), so it cannot contribute to the Hausdorff distance. Thus we can cull *all* pairs in such rows.

We apply both these culling tests simultaneously and only those bounding-box pairs that pass both the tests are finally passed on for subdividing and further testing at the next level of refinement in the hierarchy. Fig. 9(d) shows the result of applying both the culling tests to the example case.

5.2. GPU hierarchy traversal implementation

Our GPU implementation of the hierarchy traversal consists of two steps for each hierarchy level. In the first step, at any given level, we construct the min–max distance matrix consisting of the bounding-box pairs that remain potential candidates for the Hausdorff distance. At finer levels of the hierarchy, there will be too many bounding-box pairs to devote a separate row to each unique bounding-box from *surface* 1. In addition, since more and more pairs will also be culled away, we must pack the remaining pairs into the matrix. In the second step, we apply the culling tests

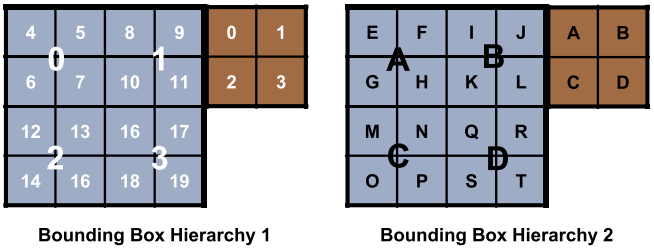
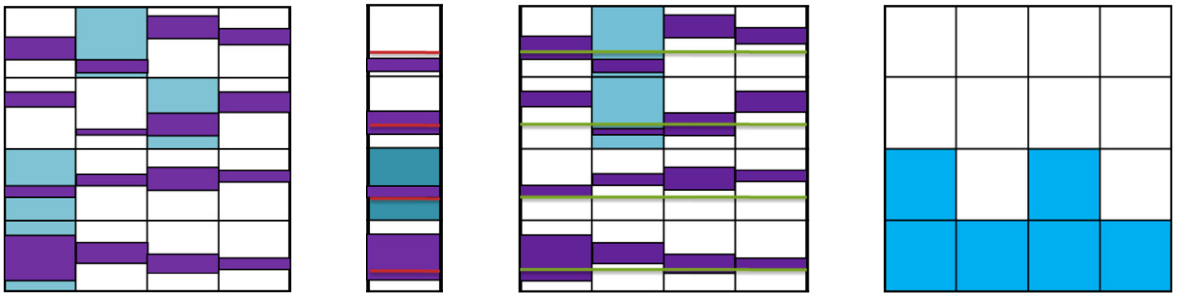


Fig. 10. Two levels of the bounding-box hierarchy stored on the GPU. We refer to this hierarchy while explaining our hierarchy traversal algorithm.

on these pairs. In order to perform the culling tests correctly on the packed min–max distance matrix, we also need to maintain bounding-box labels that indicate which original bounding-box of *surface* 1 defines each distance range, so that distance ranges from the same original *surface* 1 bounding box can continue to be grouped together (the row groupings in Figs. 8 and 9). We allocate an "address" array to aid in both the construction of the packed min–max distance matrix and the culling operation. Finally, we collect all the bounding-box pairs that are not culled and pass them to the next hierarchy level.

We will explain the two steps of the process using an example starting with the min–max distance matrix generation. Fig. 10 is the original bounding box hierarchy from which the min–max distance matrix will be calculated. We start with the bounding-box pairs that are culled in level 1. In Fig. 11(a), the lighter-shaded boxes correspond to the non-culled bounding-box pairs at this level. We collect these pairs and pack them into a square array that serves as an address template to the bounding boxes in the hierarchies of both surfaces to perform the refinement (Fig. 11(b)). Using this information, and knowing the hierarchical layout pattern from Fig. 10, we create the address array for the next level of the hierarchy (Fig. 11(c)). Once we have the address array, we can construct the corresponding min–max distance matrix with the same structure. We compute the min–max range of distances between the bounding-box pairs specified by each entry of the address array.

We now proceed to perform the culling operations using this min–max distance matrix. In order to perform the culling, we need to compute the bounds for the Hausdorff distance (explained in Section 5.1) based on this min–max distance matrix. For applying the first culling test, this is a "segmented reduction" of the min–max distance matrix to compute the smallest maximum distance for each group of pairs corresponding to the same *surface* 1 bounding box. In order to perform this operation on the GPU, we would first need to sort the bounding-box pairs with respect to their *surface* 1 bounding-box labels and perform the segmented



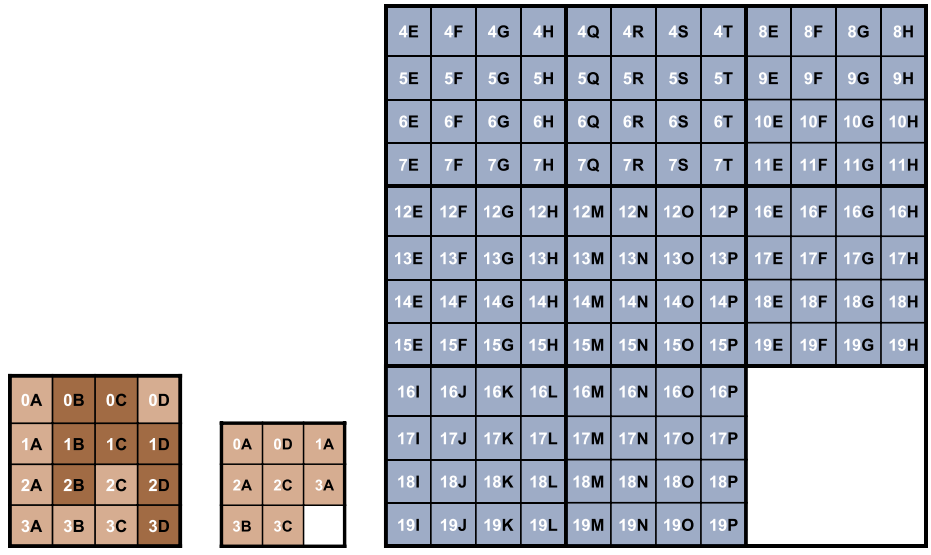
(a) Smallest min distance in each row.

(b) Largest of these smallest min distances.

(c) Ranges with max value less than the largest of the smallest min distances. Their rows are culled.

(d) Entries that pass both culling tests.

Fig. 9. The second culling test. The Hausdorff distance is greater than the green line in (c), so its first two rows are culled. The shaded pairs in (d) pass both this and the previous culling tests.



(a) Level 1. (b) Non-culled pairs. (c) Level 2.

Fig. 11. Address arrays used to aid in the hierarchy traversal. Based on bounding-box pairs not culled in level 1, the level 2 bounding-box pairs are constructed.

reduction using segmented scan [21]. Both these operations are supported by *Thrust*,[1] a CUDA library of parallel algorithms with an interface resembling the C++ Standard Template Library (STL).

In our current implementation, we perform the "segmented reduction" operation on the CPU. We then create a new array with the smallest maximum distance value for each unique *surface* 1 bounding-box label and send it back to the GPU to perform the culling. The disadvantage of performing this step on the CPU is the extra data-transfer step that could potentially be costly. On the other hand, by performing this operation on the CPU, we can avoid the sort operation by maintaining a hash table that is based on the *surface* 1 bounding-box label. We scan through all the min–max distance ranges and maintain the current bounds on the smallest maximum distance for each *surface* 1 bounding-box in this hash table. (Note that each min–max distance range is tagged with a *surface* 1 bounding-box label).

We also compute the global largest of the smallest minimum distance for the second culling operation (Fig. 9(a)–(c)) in a similar manner. We then perform the combined culling operation shown in Fig. 9(d). After the culling operation, we again collect the bounding-box pairs that are not culled and refine them to the next level of the hierarchy (Fig. 12(b)). (Note that by now, the references to *surface* 1 bounding boxes are not only out of order, but they are no longer contiguous; more coherence is lost at each step of refinement.)

Once we reach the finest level of the hierarchy, we get the list of bounding-box pairs that could potentially contribute to the Hausdorff distance. We then find the (approximated) Hausdorff distance between the surface sub-patches contained in these pairs of bounding-boxes by first decomposing the surface sub-patches into two triangles each, using the four evaluated points on each of *surface* 1 and *surface* 2, and then finding the Hausdorff distance between the triangles. We find the minimum distance between the triangles (2 for each sub-patch resulting in 4 different triangles pairs) in each sub-patch pair in the list in parallel using the GPU. However, typically the list will have multiple entries for the same *surface* 1 sub-patch corresponding to pairing it with different *surface* 2 sub-patches. For each *surface* 1 sub-patch that is in the list, we group all its entries to find the minimum of all the calculated triangle minimum distances that correspond to that *surface* 1 sub-patch. To perform this operation, we again make use of a hash table based on *surface* 1 labels (note that these labels now correspond to individual sub-patches since we are in the finest level of the hierarchy). Finally, to find the Hausdorff distance, we find the maximum of these minimum distances using a standard reduction operation.

[1] http://code.google.com/p/thrust.

6. Theoretical bounds for Hausdorff distance

In this section, we make use of the curvature-based bounds for the NURBS surfaces to derive the theoretical bounds for our Hausdorff distance computations. Given a point in space, the error in computing the minimum distance to a linear approximation of a curved surface, instead of to the surface itself, is bounded by the maximum deviation K (Eq. (2)) of the approximation to the surface. We can extend this bound to the Hausdorff distance from one surface to another.

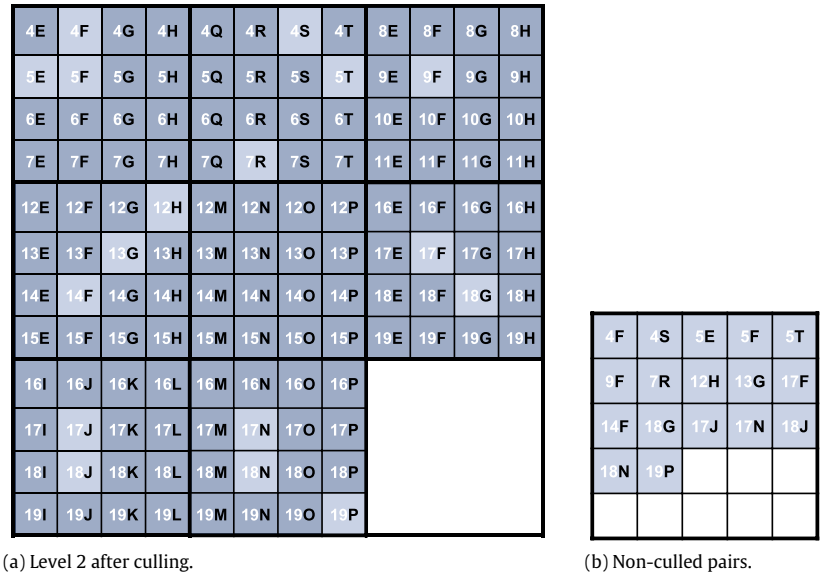
Consider two surface patches $\mathbf{A}$ and $\mathbf{B}$, having bounds K_A and K_B respectively, with respect to their linearized approximations. The surface patches cannot deviate by more than their respective bounds from any triangles, $\triangle_{\mathbf{A}}$ and $\triangle_{\mathbf{B}}$, from their respective linearized approximations (Fig. 13). We shall prove that $|\mathbf{h}(\triangle_{\mathbf{A}}, \triangle_{\mathbf{B}}) - \mathbf{h}(\mathbf{A}, \mathbf{B})| < K_A + K_B$ and therefore the Hausdorff distance found by our algorithm is bounded.

First, given any point $a \in \mathbf{A}$, the difference between $\min_{b\in\mathbf{B}} \mathbf{d}(a, b)$ and $\min_{b\in\triangle_B} \mathbf{d}(a, b)$ is bounded by the value K_B. Hence, $|\mathbf{h}(\mathbf{A}, \triangle_{\mathbf{B}}) - \mathbf{h}(\mathbf{A}, \mathbf{B})| < K_B$.

We now bound the difference in the Hausdorff distances $|\mathbf{h}(\triangle_{\mathbf{A}}, \triangle_{\mathbf{B}}) - \mathbf{h}(\mathbf{A}, \triangle_{\mathbf{B}})|$. Following the notation in Alt et al. [6], we first define a δ-neighborhood (Eq. (6)).

$$nh_\delta(Q) = \{p \in \Re^3 \mid d(p, Q) \leq \delta\}. \tag{6}$$

For a triangle $\triangle$, the $nh_\delta(\triangle)$ is an offset of the triangle, consisting of the union of three δ-radius balls centered at the vertices, three δ-radius cylinders around the edges of the triangle, and a triangular prism of height 2δ containing $\triangle$ in its center ($nh_\delta(\triangle)$ can also be viewed as the swept volume of a δ-ball over $\triangle$). By definition, any point within $nh_\delta(\triangle_B)$ has a minimum distance to



(a) Level 2 after culling. (b) Non-culled pairs.

Fig. 12. Creating the address array for the next finer level based on the non-culled bounding-box pairs in a level.

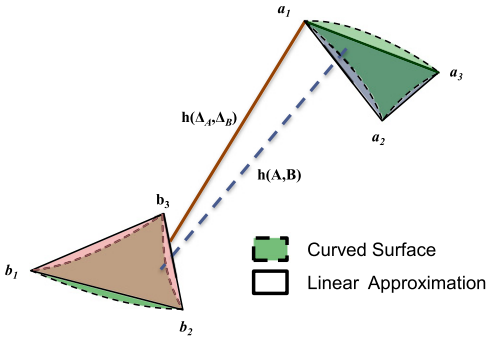


Fig. 13. The Hausdorff distance we compute is within the sum of the distance bounds for each surface.

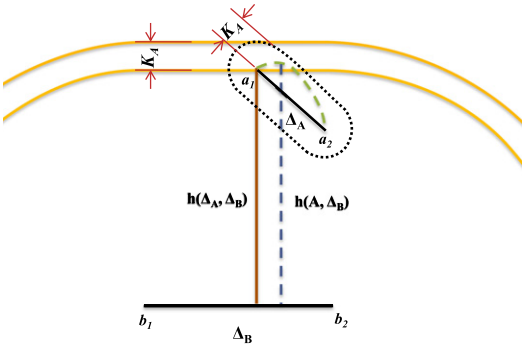


Fig. 14. Figure to prove the bound on Hausdorff distance on surface **A**, $|\mathbf{h}(\triangle_\mathbf{A}, \triangle_\mathbf{B}) - \mathbf{h}(\mathbf{A}, \triangle_\mathbf{B})| < K_A$.

$\triangle_B$ that is smaller than δ and any point on the boundary of $nh_\delta(\triangle_B)$ has a minimum distance exactly δ from $\triangle_B$.

Fig. 14 shows a schematic 2D diagram of the triangles $\triangle_B$ and $\triangle_A$ and surface patch **A** (shown as a dotted green line). We denote by $hd_\triangle = \mathbf{h}(\triangle_\mathbf{A}, \triangle_\mathbf{B})$, the Hausdorff distance between the triangles. The $hd_\triangle$ is always achieved at a vertex of $\triangle_A$, call it a_1 (see Fig. 14). The orange curves in the figure (consisting of arcs and line segments) represent portions of the boundary of $nh_{(hd_\triangle)}(\triangle_B)$ (the inner curve) and $nh_{(hd_\triangle+K_A)}(\triangle_B)$ (the outer curve), i.e., the triangle $\triangle_B$ offset by $hd_\triangle$ and by $(hd_\triangle + K_A)$ respectively.

By definition, triangle $\triangle_A$ is completely contained in $nh_{(hd_\triangle)}(\triangle_B)$ (with a_1 on its boundary). Therefore, $nh_{(hd_\triangle+K_A)}(\triangle_B)$ (which is an offset of $nh_{(hd_\triangle)}(\triangle_B)$ by K_A) contains all of $nh_{(K_A)}(\triangle_A)$ (denoted by dotted lines in Fig. 14). This means that any point inside the neighborhood $nh_{(K_A)}(\triangle_A)$ cannot have distance from $\triangle_B$ that is greater than $(hd_\triangle + K_A)$ and in particular $\mathbf{h}(\mathbf{A}, \triangle_\mathbf{B}) < \mathbf{h}(\triangle_\mathbf{A}, \triangle_\mathbf{B}) + K_A$.

Thus we have:

$$\begin{aligned}
&|\mathbf{h}(\triangle_\mathbf{A}, \triangle_\mathbf{B}) - \mathbf{h}(\mathbf{A}, \mathbf{B})| \\
&\quad = |\mathbf{h}(\triangle_\mathbf{A}, \triangle_\mathbf{B}) - \mathbf{h}(\mathbf{A}, \triangle_\mathbf{B}) + \mathbf{h}(\mathbf{A}, \triangle_\mathbf{B}) - \mathbf{h}(\mathbf{A}, \mathbf{B})| \\
&\quad \leq |\mathbf{h}(\triangle_\mathbf{A}, \triangle_\mathbf{B}) - \mathbf{h}(\mathbf{A}, \triangle_\mathbf{B})| + |\mathbf{h}(\mathbf{A}, \triangle_\mathbf{B}) - \mathbf{h}(\mathbf{A}, \mathbf{B})| \\
&\quad < K_A + K_B,
\end{aligned}$$

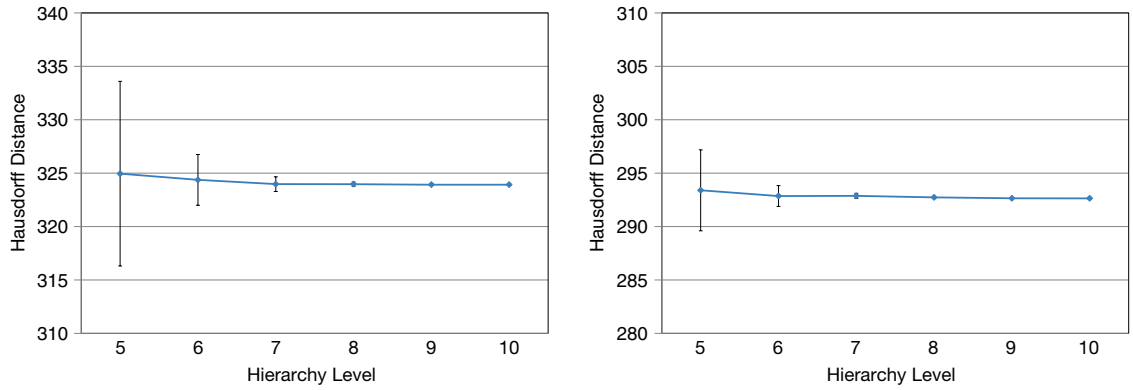
and our bound is established.

7. Results

We tested our Hausdorff distance computation algorithm on a 3.33 GHz Intel CPU equipped with an NVIDIA GeForce GTX480 GPU running CUDA 4.0. Along with the Hausdorff distance, we also compute the theoretical guarantees based on the curvature of the surface. We tested our algorithms on several different NURBS surfaces that varied in complexity, at a large number of positions, since the algorithm is very sensitive to the relative positions of the input. Table 1 lists the specifications for all the NURBS surfaces tested and the short names used to refer to them later. The surface size, given as the maximum dimension of the smallest bounding-box that encloses it, is used to measure the relative accuracy of our computations.

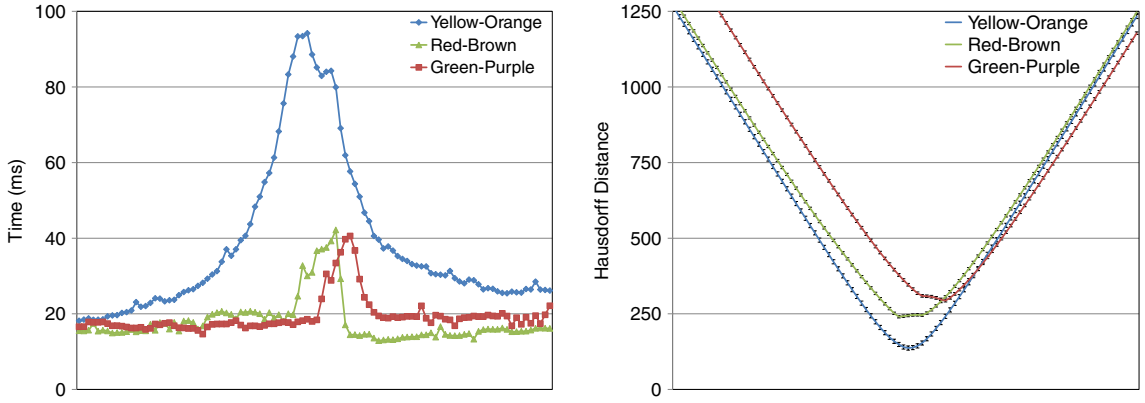
7.1. Culling performance

One of the first measures of the performance of our algorithm is the number of pairs of bounding boxes that are culled using our culling tests. The culling performance that is summarized in Table 2 is for the surface pair (Green–Purple) at positions shown in Fig. 1(a). The first column in Table 2 gives the number of AABB-pairs that would need to be tested in each level to compute the Hausdorff distance without culling. We can see that we cull more than 99.5% from the fifth level of the hierarchy on. This is typical of our culling performance.



(a) Surface pair Green–Purple. (b) Surface pair Red–Brown.

Fig. 15. Accuracy of our Hausdorff distance computation based on the number of levels of hierarchy used for the computation. The error bars represent the guaranteed bounds calculated.



(a) Computation time. (b) Hausdorff distance.

Fig. 16. Computation time for computing the Hausdorff distance between different surface pairs while interactively translating them with keyboard inputs. The surfaces were initially far apart and were moved closer so that they overlapped; they then continued to be moved until they are far apart again on the opposite side. The error bars in (b) correspond to the calculated bounds. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Surfaces used for timing our Hausdorff distance algorithm. The surfaces are shown in Fig. 1.

Surface	Name	Control points	Size
Green surface	Green	298 × 313	501.14
Purple surface	Purple	595 × 97	541.77
Red surface	Red	6 × 6	722.84
Brown surface	Brown	6 × 6	734.85
Yellow-DuckBody	Yellow	14 × 13	535.45
Orange-DuckBody	Orange	14 × 13	552.98

Table 2
The performance of our culling tests based on the level of the hierarchy of the surface. It can be seen that the culling rate improves with more levels of the hierarchy.

Hierarchy level	Hierarchy pairs	Non-culled	Culling (%)
1	16	16	0.000
2	256	256	0.000
3	4096	2 377	41.968
5	1×10^6	3 791	99.638
7	268×10^6	10 803	99.996
9	68×10^9	46 920	99.999

7.2. Accuracy

We make use of our guarantees to compute the accuracy of our Hausdorff distance computations for different surface pairs shown in Fig. 1. The error bars on each plot in Fig. 15 show actual values of our guarantees. We get extremely accurate results (error $< 1\%$ of input model size) even with just six levels of the hierarchy. Table 3 gives the relative Hausdorff distance bound as a percentage of the average size of the two input models at various levels of the hierarchy. The fourth column in Table 3 gives the equivalent number of triangles that are required in a tessellation of the same surfaces to achieve the same accuracy guarantees.

Table 3
The guaranteed bounds of our Hausdorff distance algorithm based on the actual Hausdorff distance as a percentage of the average size of the two input models. The fourth column gives the equivalent number of triangles that would be required in a tessellated model of each input to achieve the same guarantees.

Hierarchy level	Green–Purple accuracy (%)	Yellow–Orange accuracy (%)	Equivalent triangles
5	1.643	3.683	2 048
6	0.451	1.260	8 192
7	0.131	0.368	32 768
8	0.038	0.098	131 072
9	0.009	0.025	524 288
10	0.002	0.006	2 097 152

7.3. Timing results

Fig. 16 shows the computation time and the Hausdorff distance for the three different pairs of rigidly transforming surfaces (Green–Purple, Red–Brown, and Yellow–Orange), interactively moving the surfaces while computing to level 6 of the hierarchy. The timings shown include the time taken to construct the

Table 4
Total time taken in milliseconds to compute the Hausdorff distance for the different pairs of surfaces.

Hierarchy level	Green–Purple			Red–Brown			Yellow–Orange (Ducks)			Yellow–Orange deform		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
5	13.43	7.82	22.85	12.40	9.13	22.81	26.29	15.40	57.89	40.95	36.01	55.13
6	17.86	10.66	38.21	20.03	13.27	44.39	39.39	19.82	93.77	41.07	32.78	73.91
7	27.34	16.41	60.41	37.21	23.10	75.82	72.51	27.17	227.02	79.01	49.71	159.73
8	55.33	29.34	114.21	72.47	37.19	154.22	176.14	45.73	671.69	106.48	55.18	352.41

Table 5
Breakdown of the timing in milliseconds for different stages of our algorithm using a 7-level hierarchy. The overlapping surfaces are timed as shown in Fig. 1(c).

Operation	Overlapping	Far
Evaluation, BVH	5.28	6.17
Min max distance matrix	12.07	7.34
Culling ranges computation	73.23	27.44
Culling	51.82	23.79
Final distance computation	11.02	14.79
Total Time	175.85	96.02

bounding-boxes and the hierarchy at each position, which takes from 5–10 ms at this resolution. This time also includes the time taken to evaluate the second derivative and the bounds for the computation. It can be seen that the average time when the surfaces are far apart is less than 20 ms. However, when the surfaces overlap, the computation time increases due to more unculled AABB-pairs to be tested at each level.

We also timed our algorithm while computing the Hausdorff distances between both rigidly transforming and deforming surfaces while varying the number of levels of hierarchy used in the computations. Table 4 summarizes the results for the surface pairs tested. Since the computation is very sensitive to the relative positions of the two surfaces, we report the average, minimum, and the maximum timings of about 100 different runs. To compute these timings for the first three non-deforming pairs, we initially positioned the surfaces with separation distance approximately equal to their respective sizes, then moved them closer to each other until they overlapped, and finally continued the motion until they were again separated in the opposite direction. For the final deforming pair, the ducks were placed as shown in the movie, with separation distance about equal to their size, and then the control points were manipulated with the mouse. It can be seen that on average, our algorithm takes less than a hundred milliseconds for computing the Hausdorff distance to a very high accuracy.

7.4. Timing breakdown

The time differences between the different steps of our algorithm are affected mainly by culling differences. The computation of the ranges for performing the culling and the actual culling operations dominate the computation time in our current implementation at high resolutions (Table 5). For overlapping surfaces, distance computation takes a larger proportion of the time at low resolutions (5 level BVH) because culling is not very effective. However, as resolution increases (>7 level BVH), since culling often becomes more effective even as its running time increases, this tends to reduce the final distance computation time (occasionally more accurate results at higher resolutions are actually faster to compute!).

7.5. Comparison with previous algorithms

The algorithm by Elber and Grandine [14] was implemented on the IRIT modeling environment[2] using the IRIT multivariate polynomial solver. In their paper, they report running times of several seconds for curve/curve Hausdorff distance in $\Re^3$. For the problem of this paper, the surface/surface Hausdorff distance, there are no implementation examples in that paper, and there is only a partial implementation in IRIT, which computes only the Hausdorff distance attained at antipodal points (the most common case). The partial implementation takes several seconds to compute the Hausdorff distance between two surfaces.

We cannot directly compare our performance with what is to our knowledge the best-performing approximate polyhedral Hausdorff distance software [4], since our algorithm requires no separate preprocessing to tessellate the surface into polygons or for one-time BVH construction. Hence, it can handle deformable models, unlike algorithms such as [22,4]. However, we can perform a rough comparison of performance on non-deforming models computed to similar accuracy as reported in the latter paper as follows.

Tang et al. [4] computed results on a 2.6 GHz Intel CPU for tessellated models with absolute error bounds of 10^{-4} on model sizes that appear to be about 0.4 or 0.5, for an error of 0.02%–0.03% of model size. This is on top of tessellation error, which probably contributed more to the overall error given how many triangles were there in their input. The most challenging case they reported was for the Hausdorff distance between 16.7K and 69.7K triangle bunny models; this corresponds to the number of triangles in tessellated surfaces computed between hierarchy levels 6 & 7 and 7 & 8 respectively in our system. Positioning these surfaces with high overlap, their two-sided Hausdorff distance computation took 948 ms (not including pre-processing time to construct the BVH). Our worst-case timing computed at hierarchy level 7 was 227 ms, including BVH construction, for the one-sided Hausdorff distance for the severely overlapped Yellow–Orange DuckBody models (Fig. 1(c)).

8. Limitations and future work

Our implementation of the hierarchy traversal is not optimized for performance. We have not compared the performance of our hybrid CPU–GPU implementation with a pure GPU implementation, which could avoid reading back the min–max distance matrix by using a segmented scan on the GPU.

Another limitation of our algorithm is that our error bounds will be large when the NURBS surface has a very high curvature. While it is true that higher curvature increases error, our bottom-up BVH construction allows the use of the local error derived at the finest level of detail throughout the hierarchy, and thus this effect is not very large in practice. For a top down approach, this would have been a more significant limitation. However, if the input parameterization is also very non-uniform, combined with high local curvature, the error bounds will be very large. In such cases, the program could use the error bounds to flag the result as potentially ambiguous.

One potential application of our interactive algorithm is to geometric surface optimization problems. For example, if the desired final surface shape is known and the effect of forces on the input surface shape has been extensively modeled but not in an invertible manner, the question is how to find the initial

[2] www.cs.technion.ac.il/irit.

surface shape that yields the final desired surface shape. For this application, measuring shape similarity with our algorithm rather than a Hausdorff distance algorithm that operates on tessellated meshes is preferred since the surface to be optimized is best represented using a high-level representation (such as control points in the case of NURBS) instead of a tessellated mesh. This is because the latter has far more degrees of freedom than optimal for the optimization. Our algorithm could efficiently evaluate the Hausdorff distance between the dynamic surface being optimized and the desired surface, with the Hausdorff distance functioning as the objective function to minimize. Finally, for directing the optimization, our algorithm could find all the surface locations within the error bounds where the "bounded" Hausdorff distance may be attained (unlike Hausdorff distance between meshes, which will typically only find a single location), which should significantly accelerate convergence.

9. Conclusions

We have presented the first practical, fully implemented algorithm to compute the Hausdorff distance from one NURBS surface to another, within computed bounds. Because our algorithm has no separate preprocessing, it can compute the Hausdorff distance between dynamically deforming surfaces. Our method provides guaranteed bounds for the computations that can be used to match user-defined tolerance values. The user can also loosen the tolerance to increase interactivity or tighten it when accuracy rather than performance is critical. Our implementation is fast enough to calculate the two-sided Hausdorff distance interactively for most practical cases. Our implementation is both more accurate and has better performance than previous algorithms for computing Hausdorff distances between non-deforming, pre-tessellated models.

Acknowledgments

The models used in the paper were downloaded from 3D content central [23]. We would like to thank Wei Li for his comments. We would also like to thank the reviewers for their valuable suggestions. This material is based upon work supported in part by UC Discovery under Grant No. DIG07-10224, and the National Science Foundation under CAREER Award No. 0547675.

Appendix. Supplementary data

Supplementary material related to this article can be found online at doi:10.1016/j.cad.2011.08.022.

References

[1] Alt H, Behrends B, Blömer J. Approximate matching of polygonal shapes. Annals of Mathematics and Artificial Intelligence 1995;13(3–4):251–65.
[2] Guthe M, Borodin P, Klein R. Fast and accurate Hausdorff distance calculation between meshes. Journal of WSCG 2005;13:41–8.
[3] Varadhan G, Manocha D. Accurate Minkowski sum approximation of polyhedral models. Graphical Models 2006;68(4):343–55.
[4] Tang M, Lee M, Kim YJ. Interactive Hausdorff distance computation for general polygonal models. ACM Transactions on Graphics 2009;28: 74:1–74:9.
[5] Atallah MJ. A linear time algorithm for the Hausdorff distance between convex polygons. Information Processing Letters 1983;17(4):207–9.
[6] Alt H, Braß P, Godau M, Knauer C, Wenk C. Computing the Hausdorff distance of geometric patterns and shapes. In: Aronov B, Basu S, Pach J, Sharir M, editors. Discrete and computational geometry. The Goodman–Pollack festschrift. Algorithms and combinatorics, vol. 25. Springer; 2003. p. 65–76.
[7] Bartoň M, Hanniel I, Elber G, Kim M-S. Precise Hausdorff distance computation between polygonal meshes. Computer Aided Geometric Design 2010;27: 580–91.
[8] Llanas B. Efficient computation of the Hausdorff distance between polytopes by exterior random covering. Computational Optimization and Applications 2005;30:161–94.
[9] Alt H, Scharf L. Computing the Hausdorff distance between curved objects. In: Proceedings of the 20th European workshop on computational geometry. 2004. p. 233–6.
[10] Alt H, Scharf L. Computing the Hausdorff distance between curved objects. International Journal of Computational Geometry and Applications 2008; 18(4):307–20.
[11] Chen X-D, Ma W, Xu G, Paul J-C. Computing the Hausdorff distance between two B-spline curves. Computer Aided Design 2010;42:1197–206.
[12] Kim Y-J, Oh Y-T, Yoon S-H, Kim M-S, Elber G. Precise Hausdorff distance computation for planar freeform curves using biarcs and depth buffer. The Visual Computer 2010;26:1007–16.
[13] Bai Y-B, Yong J-H, Liu C-Y, Liu X-M, Meng Y. Polyline approach for approximating Hausdorff distance between planar free-form curves. Computer Aided Design 2011;43:687–98.
[14] Elber G, Grandine T. Hausdorff and minimal distances between parametric freeforms in R2 and R3. In: Proceedings of the 5th international conference on advances in geometric modeling and processing. Springer-Verlag; 2008. p. 191–204.
[15] Krishnan S, Gopi M, Lin M, Manocha D, Pattekar A. Rapid and accurate contact determination between spline models using shelltrees. Computer Graphics Forum 1998;17(3):315–26.
[16] Krishnamurthy A, McMains S, Haller K. Accelerating geometric queries using the GPU. In: SIAM/ACM joint conference on geometric and physical modeling. ACM; 2009. p. 199–210.
[17] Kanai T. Fragment-based evaluation of Non-Uniform B-spline surfaces on GPUs. Computer-Aided Design and Applications 2007;4(3):287–94.
[18] Filip D, Magedson R, Markot R. Surface algorithms using bounds on derivatives. Computer Aided Geometric Design 1987;3(4):295–311.
[19] Lauterbach C, Garland M, Sengupta S, Luebke D, Manocha D. Fast BVH construction on GPUs. Computer Graphics Forum 2009;28(2):375–84.
[20] Lauterbach C, Mo Q, Manocha D. Proximity: hierarchical GPU-based operations for collision and distance queries. Computer Graphics Forum 2010;29(2): 419–28.
[21] Sengupta S, Harris M, Zhang Y, Owens JD. Scan primitives for GPU computing. In: Symposium on graphics hardware. ACM, Eurographics Association; 2007. p. 97–106.
[22] Guthe M, Balázs A, Klein R. GPU-based trimming and tessellation of NURBS and T-spline surfaces. ACM Transactions on Graphics 2005;24(3):1016–23.
[23] 3D Content Central, 2009. http://www.3dcontentcentral.com.