Graphical Models 74 (2012) 255–264

Contents lists available at SciVerse ScienceDirect

Graphical Models

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

Computing the Hausdorff distance between NURBS surfaces using numerical iteration on the GPU ☆

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

[a] Technion, Israel Institute of Technology, Haifa, Israel
[b] University of California, Berkeley, Berkeley, USA

ARTICLE INFO

Article history:
Received 5 March 2012
Received in revised form 30 April 2012
Accepted 4 May 2012
Available online 18 May 2012

Keywords:
Numerical iterations
Hausdorff distance
NURBS
GPU
Geometric algorithms

ABSTRACT

We present a GPU algorithm for computing the directed Hausdorff distance between two NURBS surfaces. The algorithm is based on sampling of one surface, and performing numerical iterations on the GPU to compute the minimal distance from each sample to the other surface. An error analysis for the Hausdorff distance computations is performed, based on bounds on the NURBS surfaces. We compare a CUDA implementation of our algorithm to existing methods, demonstrating that the new method addresses limitations of previous hierarchical culling methods such as the sensitivity to the position of the inputs.

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 [11], geometric approximation [27], and penetration depth calculation for physically based animation [25]. As a result, 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.

In a previous paper [18] we introduced a GPU-accelerated algorithm for computing the directed Hausdorff distance between NURBS surfaces, which remain the de facto standard for CAD models. That algorithm was based on GPU traversal of a bounding-box hierarchy and culling pairs of bounding boxes that could not contribute to the Hausdorff distance. In this paper we introduce a new approach that also exploits the parallelism of the GPU, but in a different manner, without building the bounding box hierarchy that is the foundation of the previous algorithm. Our new algorithm is based on sampling of one surface, and performing numerical iterations on the GPU to compute the minimal distance from each sample to the other surface. We compare our new algorithm to the previous one, both theoretically and empirically, and show its advantages over existing methods.

1.1. Contributions

In this paper, we provide a GPU algorithm to compute the directed Hausdorff distance between two NURBS surfaces. Our main contributions include:

- A novel GPU-accelerated directed Hausdorff distance computation between NURBS surfaces. We sample the first surface and perform numerical iterations on the GPU to compute the minimal distance

☆ This paper has been recommended for acceptance by Jarek Rossignac.
* Corresponding author.
E-mail addresses: ihanniel@tx.technion.ac.il (I. Hanniel), adarshk@berkeley.edu (A. Krishnamurthy), mcmains@me.berkeley.com (S. McMains).

1524-0703/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.gmod.2012.05.002

from each sample to the other surface. We then find the maximal of these minimal distances, which is the Hausdorff distance.

- Error analysis for the Hausdorff distance computations based on the computation of bounds on the first NURBS surface, and on the user-defined numerical iteration tolerance.
- Implementation of our method and comparison with existing methods, demonstrating that the new method is relatively insensitive to the relative position of the inputs, and handles the overlaps, near-offsets, and offsets that are failure cases for hierarchical culling methods with ease.

1.2. Mathematical preliminaries

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

$$h(\mathcal{A},\mathcal{B}) = \max_{\vec{a}\in\mathcal{A}}\left(\min_{\vec{b}\in\mathcal{B}} d(\vec{a},\vec{b})\right) \quad (1)$$

Definition 2 (*Hausdorff distance*). The Hausdorff distance between $\mathcal{A}$ and $\mathcal{B}$ is given by:

$$H(\mathcal{A},\mathcal{B}) = \max\left(h(\mathcal{A},\mathcal{B}), h(\mathcal{B},\mathcal{A})\right) \quad (2)$$

2. Related work

The Hausdorff distance computation has attracted considerable research attention in computer graphics, computational geometry and geometric modeling [1,6].

For polygonal meshes in $\Re^3$ with $O(n)$ triangles, Alt et al. [2] presented theoretical and randomized algorithms. 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. 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.

Due to the complexity of computing the Hausdorff distance exactly, approximate practical algorithms have been proposed [8,5,19,11]. Tang et al. [25] implemented an approximate algorithm that is based on a Bounding Volume Hierarchy (BVH) that is computed in a preprocess. Their implementation is very fast in practice, running at interactive speed for meshes of practical size.

On the computation of the Hausdorff distance between freeform surfaces, there are fewer previous results. For freeform curves, Alt and Scharf [3,4] presented an algorithm for Hausdorff distance computation based on a characterization of the possible points where the distance can be attained. Elber and Grandine [10] extended the results from Ref. [3] to Hausdorff distance computations 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.

In our recent paper [18], we introduced a GPU algorithm that computes the Hausdorff distance between NURBS surfaces. Interactive speeds are obtained by performing GPU traversal of a bounding-box hierarchy and selectively culling pairs of bounding boxes that could not contribute to the Hausdorff distance. To the best of our knowledge, this was the first completely implemented algorithm to compute the Hausdorff distance for pairs of complex curved surfaces without requiring pre-tessellation. Recently Kim et al. [15] presented a compact representation for the BVH of NURBS surfaces using Coons patches, which could possibly be used to construct a BVH-based algorithm for computing the Hausdorff distance between NURBS surfaces.

The algorithm described in this paper is based on computing the projection of points from one surface onto another surface. The problem of projecting a point onto a freeform curve or surface has been studied in the past. Most solutions are based on iterative methods that converge to the projected point. In Piegl and Tiller [22, 6.1], the problem is formulated as minimizing the distance from a point P to the surface; bivariate Newton–Raphson iteration is used to solve the minimization problem.

One of the main problems with numerical iteration schemes is to find a good initial value that reduces the number of iterations of the algorithm and increase its stability. In [23], the projection is computed by subdividing the surface into very flat patches. Each patch can then be approximated by a quadrilateral, onto which the point is

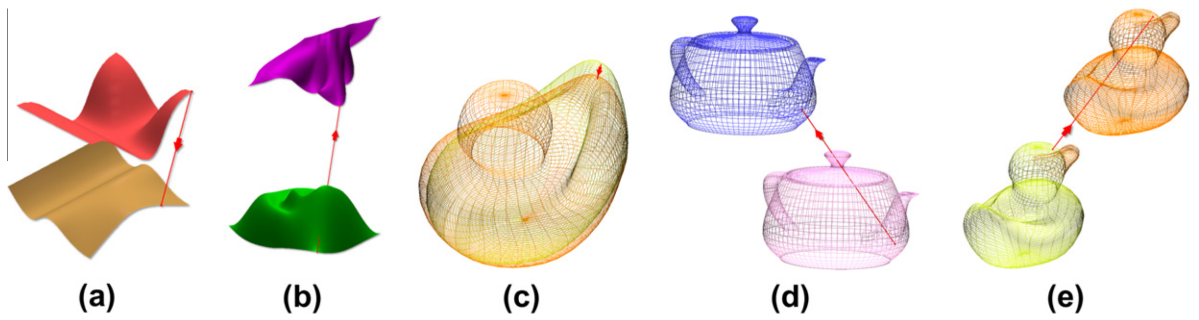


Fig. 1. Computing the Hausdorff distance for NURBS surfaces; the arrows connect the locations on the two models where the Hausdorff distance is achieved. The tail of the arrow indicates the surface over which the minimum distances are maximized.

projected. Ma and Hewitt [20] present an accurate and efficient method to project a point onto a NURBS surface by first subdividing the NURBS surface into a set of Bézier patches. They then find approximate candidate points, based on the Bézier control polygon, and use them as initial starting points for a numerical iteration. Hu and Wallner [14] improve the iteration convergence rate of previous methods by using a second-order iteration scheme.

A similar problem arises when rendering NURBS surfaces directly on the GPU using ray tracing. There, a ray-surface intersection needs to be computed for all pixels on the screen. Both the point-surface projection problem and the ray-surface intersection problem require computing the roots of a set of multivariate polynomial equations. Pabst et al. [21] presented a GPU-based NURBS ray casting implementation, based on a bi-variate Newton iteration method performed on the GPU. In order to achieve convergence to the root, they subdivide the surface into sub-patches in a preprocessing step and compute the convex hull of each sub-patch. The convex hull is then used as a bounding volume for rendering and utilized to find a good initial value for the numerical iteration.

3. Algorithm description

Computing the directed Hausdorff distance from surface $\mathcal{A}$ to $\mathcal{B}$ conceptually requires computing the minimal distance value from every point on $\mathcal{A}$ to $\mathcal{B}$ and then finding the maximal of these values. Therefore, a solution to the problem could be to sample a set of points on $\mathcal{A}$ and for each point $a \in \mathcal{A}$, compute its minimal distance from $\mathcal{B}$. Indeed, using sampling of the surfaces has been proposed in the past (in the context of meshes) [8,5]. However, sampling both surfaces and comparing the distance between every point-pair can be prohibitively expensive. Therefore, such methods construct spatial subdivisions to make the comparison more efficient.

In the present work, we explore designing a sampling-based algorithm that can exploit the GPU's tremendous computing power when brought to bear on massively data-parallel operations. Our solution was inspired by both the work on ray-casting directly on the GPU [21] and by the projection methods described above [20]. We use a sampling approach to sample one surface, but implement a numerical method to project the sampled points onto the second surface and utilize the parallelism of the GPU to implement these steps efficiently. We then find the maximal of the resulting point-surface distances, which is the Hausdorff distance.

Our algorithm can therefore be described as follows:

1. In a preprocessing step subdivide surface $\mathcal{B}$ into relatively flat Bézier patches.
2. On the GPU, compute the sampled points on surface $\mathcal{A}$ using the method described in [17].
3. For each of the Bézier patches $\mathcal{B}_i$ do:
   - For all sampled points $a \in \mathcal{A}$ perform in parallel on the GPU: Project a onto $\mathcal{B}_i$. If the distance to the projected point is smaller than the current minimal distance from a, store the point and the distance value as the current minimal distance of that point a.
4. Find the maximal value over all the minimal distance values attained.

3.1. Algorithm design and implementation choices

The first step of the algorithm above, subdividing the surface into relatively flat patches, is naturally recursive. Therefore we implemented it on the CPU. Computing and projecting each sampled point (Steps (2) and (3) of the algorithm), on the other hand, are arithmetically intense operations that are not dependent on the other sampled points, and thus well suited for the strength of the GPU. Furthermore, a CPU implementation would be infeasible due to the large number of sample points.

We chose to implement our method using Bézier surfaces, since their structure enables evaluation code with more uniform behavior and less branching, which is preferable for a GPU implementation. We chose to evaluate the Bézier surfaces using the DeCasteljau algorithm, rather than with direct evaluation of the basis functions. As noted in [21], the fact that the DeCasteljau algorithm is robust and enables computing the partial derivatives with little additional overhead makes it more suitable for our needs. We use these partial derivatives in the projection method described in Section 3.2. The projection method was chosen because it does not require second derivatives, the computation of which may require data access patterns that are not as well suited to the GPU, though there is a trade-off here in the speed of convergence.

Comparing the distance of the projected point and updating the current minimal distance value is done using a ping pong access pattern. The updated values are copied to output memory, which in turn becomes the input for the distance comparison of the next Bézier patch $\mathcal{B}_i$.

After the sampled points have been projected in parallel onto surface $\mathcal{B}$, we have to find the maximal value over the distances corresponding to each sampled point. We found that if this step was performed on the CPU, it dominated the running time if the sampling resolution was high; therefore we used the Thrust library [26], which implements an efficient CUDA reduction function for finding a maximal value in GPU memory.

3.2. Projecting points onto a Bézier patch using the GPU

Given a point a and a Bézier patch $\mathcal{B}(u, v)$ that is C^1 and regular, we wish to project a onto $\mathcal{B}$, i.e., we want to find the point (and uv-values) on $\mathcal{B}$ for which the distance between a and $\mathcal{B}$ is minimal. This is attained either on a boundary curve or when the vector $(\mathcal{B}(u, v) - a)$ is perpendicular to the tangent at $\mathcal{B}(u, v)$. Thus, it is a solution to the following set of equations (see [22]):

$$\begin{aligned} &\mathcal{B}(u, v) - a, \quad \mathcal{B}_u(u, v) = 0 \\ &\mathcal{B}(u, v) - a, \quad \mathcal{B}_v(u, v) = 0 \end{aligned} \tag{3}$$

Our projection method is based on solving this set of equations using a numerical iteration scheme, which we implement on the GPU. It is a geometric first order method described by Hu and Wallner [14] for surfaces and by Hoschek [13] for curves, which can also be viewed as a gradient descent for minimizing the function $||\mathcal{B}(u,v) - a||$. We compute the tangent plane of $\mathcal{B}$ at (u,v) and project point a onto the plane. The plane is defined by the partial derivatives of $\mathcal{B}$ and therefore can be parameterized by two variables Δ_u and Δ_v, which correspond to the parameters along the partial derivatives $\mathcal{B}_u$ and $\mathcal{B}_v$. That is, for a given (u,v) the tangent plane is defined by:

$$Pl(u,v) = \mathcal{B}(u,v) + \mathcal{B}_u(u,v)\Delta_u + \mathcal{B}_v(u,v)\Delta_v$$

Once the point is projected onto Pl, Δ_u and Δ_v can then be computed as the barycentric coordinates of the triangle defined by the point $\mathcal{B}(u,v)$ and the vectors $\mathcal{B}_u$ and $\mathcal{B}_v$.

Δ_u and Δ_v are then added to (u,v) to get the new (u,v) values for the next iteration. The process stops when a predefined termination criterion is met. One possible criterion can be to stop the process when Δ_u and Δ_v are smaller than some uv-tolerance. However, in our implementation, we can compute the distance from the point to the projection. Therefore, we chose to stop the process when the difference between the computed distances in consecutive iterations is below a user-defined tolerance E_B. This enables us to have control over the theoretical lower bound on the error of our computation (see Section 4.1.1). In the implementation there is also a predefined maximum iteration number to avoid infinite loops.

One of the main problems with iterative schemes is to find a good initial value that will enable fast convergence to the desired solution. In our algorithm we use the control mesh of the Bézier patch to get an initial (u,v) value. We go over the control mesh and find the control point that has the minimal distance from the projected point a. We use the (u,v) values corresponding to the indices of this point as initial values. Namely, if P_{ij} is the closest point on the control mesh, and the degree of the surface is D_u, D_v respectively (i.e., $0 \leqslant i \leqslant D_u$, $0 \leqslant j \leqslant D_v$) then we use: $(u,v) = (i/D_u, j/D_v)$ as an initial value for the iteration.

Fig. 2 visualizes the projection of a set of sampled points from the pictured planar surface onto a NURBS surface.

3.3. Subdividing a surface into flat Bézier patches

Our point-projection method is an iterative procedure that converges to a point on the surface whose tangent plane is orthogonal to the projection vector (see Eq. (3)). However, on an oscillating surface there can be more than one point that satisfies this condition. Therefore, point-projection algorithms such as Ma and Hewitt [20] first subdivide the target surface into flat Bézier patches.

The procedure subdivides the surface into Bézier patches and checks each patch for some flatness criterion. If the criterion is met the subdivision stops and the resulting patch is stored, otherwise the Bézier patch is adaptively subdivided until the criterion is met.

Different flatness criteria can be implemented. Ma and Hewitt [20] used a criterion based on the control mesh consisting of simple convex polygons in both the u and v directions. Piegl and Tiller [23] computed an approximating plane to the four corners and considered a patch to be flat if all the control points were within a user-defined tolerance of the plane.

In this work we used a method based on the normal field of the Bézier patch. We consider a Bézier patch to be flat if the angle spanned by its normal field is within some user-defined tolerance. The normal field of a freeform surface $S(u,v)$ is defined as $S_u(u,v) \times S_v(u,v)$, which is itself a freeform surface of higher degree (see [24,9]). A bound on the normal field surface can therefore give us a bound on the normal angle-span. For example, once the normal field surface is computed, we can bound the normal directions by a cone that contains all of its control points (similarly to what was done in [12]). However, computing the normal field surface is relatively expensive and therefore, in our implementation we chose to sample normals on the patch and use the angle-bound on the samples as a flatness criterion. We found this method to work well in practice.

4. Theoretical bound for hausdorff distance

In this section, we derive a theoretical bound for our Hausdorff distance computations. In fact we will give a bound that is not symmetric. We show that the Hausdorff distance computed with our method cannot deviate from the true Hausdorff distance by more than some predetermined bounds K_A and K_B, where K_A is a bound on a distance from any point on surface $\mathcal{A}$ to its nearest sample point, and K_B is a bound on the deviation of a computed projected distance of a given point to the actual projected distance. As will be shown, this bound is not symmetric. Denoting by *computed_HD* the Hausdorff distance computed by our method, and by *HD* the true Hausdorff distance, we prove below that:

$$-K_B \leqslant HD - computed\_HD \leqslant K_A$$

Then in Section 4.2 we will discuss how we compute the bound K_A for a given sample set, and how the bound K_B can be derived under certain assumptions from our iterative method. Given these bounds on the surface points, we now prove the upper and lower bounds on the Hausdorff distance computed by our method.

4.1. Lower and upper bounds

Given the sampled set of points on surface $\mathcal{A}$, for any point $\mathcal{A}(u,v)$ there exists a nearby point $\hat{a}$ such that:

$$||\mathcal{A}(u,v) - \hat{a}|| < K_A$$

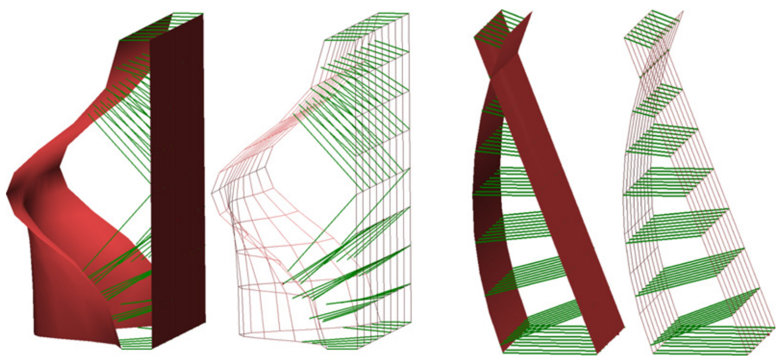
The iterative method projects a point a onto the surface $\mathcal{B}$ and stops at a point $\hat{b}$ where the computed distance from a to $\hat{b}$, by the definition of K_B, satisfies:

$$|dist(a,\mathcal{B}) - computed\_dist(a,\mathcal{B})| \leqslant K_B$$

Let HD be the true Hausdorff distance between surfaces $\mathcal{A}$ and $\mathcal{B}$; assume it is attained at a point $a^* \in \mathcal{A}$.

4.1.1. Bound from below

First, we bound from below by noticing that the Hausdorff distance $HD(samples, \mathcal{B})$ between the sample points of $\mathcal{A}$ and the surface $\mathcal{B}$ must be smaller (or equal) to HD (since the sampled points are themselves points of



(a) *Non-intersecting Surfaces* **(b)** *Intersecting Surfaces*

Fig. 2. Projecting a set of points (sampled from the plane surface) onto another surface. The green lines correspond to the minimal distance vector from the sampled points to the surface. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$\mathcal{A}$, and HD is the maximal distance from $\mathcal{A}$ to $\mathcal{B}$, therefore in the worst case HD would be attained on a sample point).

Fig. 3 visualizes the logic that follows. Let $\hat{a}$ be the sampled point on $\mathcal{A}$ of *computed_HD*, then by the definition of K_B we get:

$$computed\_HD = computed\_dist(\hat{a}, \mathcal{B}) \leqslant dist(\hat{a}, \mathcal{B}) + K_B$$

Since the Hausdorff distance from the samples is the greatest of the sample-points distances, we get:

$$\leqslant HD(samples, \mathcal{B}) + K_B$$

And since the Hausdorff distance is maximal over all of $\mathcal{A}$ including sampled points, we finally get:

$$\leqslant HD + K_B$$

Thus:

$$-K_B \leqslant HD - computed\_HD$$

4.1.2. Bound from above

Let $a^* \in \mathcal{A}$ be the point where HD is attained (see Fig. 4). There exists a nearby sample point $\hat{a}$ such that $||\hat{a} - a^*|| \leqslant K_A$. The computed distance from such a point $\hat{a}$ to $\mathcal{B}$ is to a point that is also on $\mathcal{B}$; let us denote this point $\hat{b}$.

$$HD = dist(a^*, \mathcal{B})$$

Since HD is by definition a minimal distance (from a^*) to $\mathcal{B}$, it is, in particular, smaller than $dist(a^*, \hat{b})$ and therefore by the triangle inequality:

$$\leqslant dist(\hat{a}, a^*) + computed\_dist(\hat{a}, \mathcal{B})$$

$$\leqslant K_A + computed\_dist(\hat{a}, \mathcal{B})$$

Now, since the computed HD is larger than all other computed distances and in particular from $computed\_dist(\hat{a}, \mathcal{B})$, we get:

$$\leqslant K_A + computed\_HD$$

Therefore:

$$HD - computed\_HD \leqslant K_A$$

Combining the lower and upper bounds we finally get:

$$-K_B \leqslant HD - computed\_HD \leqslant K_A$$

4.2. Computing K_A and K_B

As mentioned in Section 3.2, in our implementation, we stop the process when the difference between the com-

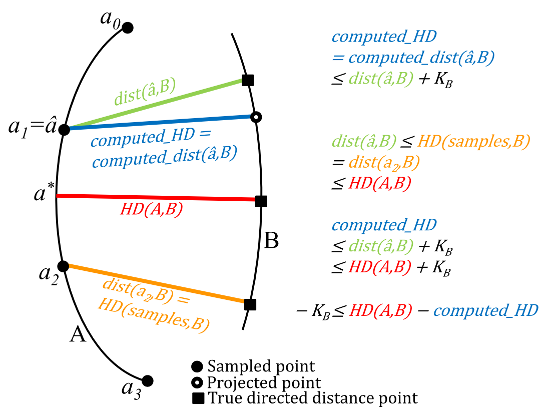

a_0
$a_1=\hat{a}$
a^*
a_2
a_3
A
B
$dist(\hat{a},B)$
computed_HD = computed_dist(â,B)
HD(A,B)
$dist(a_2,B)$ = HD(samples,B)
computed_HD = computed_dist(â,B) ≤ dist(â,B) + K_B
dist(â,B) ≤ HD(samples,B) = $dist(a_2,B)$ ≤ HD(A,B)
computed_HD ≤ dist(â,B) + K_B ≤ HD(A,B) + K_B
$-K_B$ ≤ HD(A,B) − computed_HD
● Sampled point
○ Projected point
■ True directed distance point

Fig. 3. Bound from below.

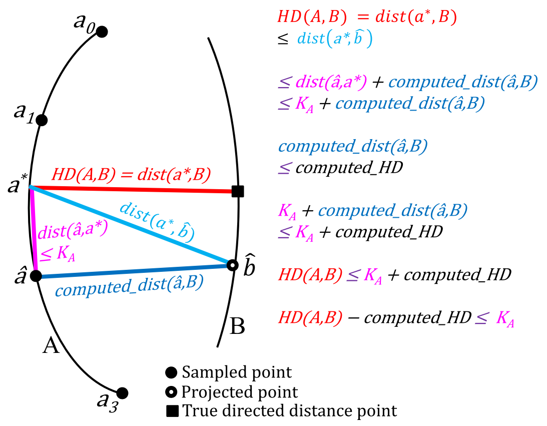

a_0
a_1
a^*
$\hat{a}$
a_3
$\hat{b}$
A
B
HD(A,B) = dist(a*,B)
dist(a*,$\hat{b}$)
dist(â,a*) ≤ K_A
computed_dist(â,B)
HD(A,B) = dist(a*,B) ≤ dist(a*,$\hat{b}$)
≤ dist(â,a*) + computed_dist(â,B)
≤ K_A + computed_dist(â,B)
computed_dist(â,B) ≤ computed_HD
K_A + computed_dist(â,B) ≤ K_A + computed_HD
HD(A,B) ≤ K_A + computed_HD
HD(A,B) − computed_HD ≤ K_A
● Sampled point
○ Projected point
■ True directed distance point

Fig. 4. Bound from above.

puted distances in consecutive iterations is below a user-defined tolerance E_B. Assuming that the convergence of our method satisfies:

$$|dist_{i+1} - dist_i| \leqslant \alpha |dist_i - dist_{i-1}|$$

for some $0 < \alpha < 1$, we can bound the deviation from the actual projected distance. It can be shown that under this assumption, if $|dist_{i+1} - dist_i| < E_B$ then $|dist_{i+1} - dist^*| < \frac{\alpha}{1-\alpha}E_B$, where $dist^*$ is the actual projected distance (i.e., the limit of the iterative method). This is shown by:

$$\begin{aligned}
&|dist_{i+1} - dist^*| = |dist_{i+1} - dist_{i+2} + dist_{i+2} - \ldots - dist^*| \\
&\leqslant |dist_{i+1} - dist_{i+2}| + |dist_{i+2} - dist_{i+3}| + \ldots \\
&\leqslant \alpha|dist_i - dist_{i+1}| + \alpha^2|dist_i - dist_{i+1}| \\
&+ \alpha^3|dist_i - dist_{i+1}| \ldots \leqslant \frac{\alpha}{1-\alpha}E_B
\end{aligned}$$

Thus, $\frac{\alpha}{1-\alpha}E_B$ can serve as our K_B bound (e.g., for $\alpha = 0.5, K_B = E_B$). In our implementation we used an E_B that was smaller than K_A, which enables a relatively large α value.

K_A, on the other hand, is a bound on the maximal distance a surface point can have from its nearest sample point. Given the sample distances Δ_u, Δ_v in the uv-domain, such a bound could be derived using partial derivatives of the surface and the mean value theorem. However, such a bound is not tight since it relies on a global bound of the derivatives over the whole uv-domain.

Therefore, we use a different scheme, which relies on axis aligned bounding boxes (AABBs) that enclose the surface sub-patches lying between each group of four adjacent surface points. 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. To guarantee that the bounding boxes enclose their surface patches, we expand the bounding boxes based on the curvature of the surface patch. Our AABB construction follows that of Krishnamurthy et al. [18].

Once we have a set of tight bounding boxes that enclose all sub-patches of the surface, we are guaranteed that all points of the surface are enclosed within at least one bounding box. Thus, the maximal distance from any point on the surface to its nearest sample point cannot exceed the maximal radius of the bounding boxes (i.e., the length of the maximal diagonal divided by two). K_A is therefore half the length of the maximal diagonal of the bounding boxes. This tighter bound requires more computation but is only computed once.

4.2.1. Conservativeness of bounds in practice

The K_A bound discussed in Section 4.2 can be relatively large. This is because it is defined as the distance to a sampled point and is therefore on the order of the sampling distance. For example, for a square unit plane sampled uniformly by an $n \times n$ grid, K_A is $\frac{\sqrt{2}}{2}\frac{1}{n}$. Thus, the bound on the distance from a point on the surface to its nearest sample point is of a first order. In contrast, in [18] the bound was of second order, since the distance there was measured to a piecewise linear approximation of the surface.

However, our use of the bound in our derivation of the upper bound above is conservative. We use it above within a triangle inequality but in fact the inequality $dist(a^*, \hat{b}) \leqslant dist(\hat{a}, a^*) + computed\_dist(\hat{a}, \hat{b})$ also depends on the angle between $a^*, \hat{a}$, and $\hat{b}$; equality is attained only if the three points are collinear. Indeed our experiments show that our computed Hausdorff distance is in most cases within the second-order bounds computed in [18] (see Section 5.1).

Still, there can be cases where the difference between the Hausdorff distance computed with our method and the true Hausdorff distance is close to the theoretical K_A bound. Consider, for example, surface $\mathcal{B}$ being a cylinder with its axis along the z-axis, and surface $\mathcal{A}$ a planar patch in the xy-plane (see Fig. 5a). The Hausdorff distance in this case is attained at the origin since any point on the plane that is not the origin has a smaller projection distance to the cylinder (this is a case where the Hausdorff distance is attained on an intersection of surface $\mathcal{A}$ with the self-bisector of surface $\mathcal{B}$, one of the cases discussed in [10]). If we translate the planar patch so that the origin is in between four sample points, the computed Hausdorff distance will be attained on one of the sample points surrounding the origin. Fig. 5 demonstrates this case. The true Hausdorff distance for the configuration in Fig. 5 is approximately 320, and the theoretical K_A bound is $8\sqrt{2} \simeq 11.3$ (in our implementation, the K_A bound we compute is larger since we inflate the bounding box). The Hausdorff distance computed with our method is 310.42, which is indeed off by close to the K_A bound.

5. Results

5.1. Single surface timing results

We first computed the Hausdorff distances for the three surface pairs shown in Fig. 1a–c, using both our new Numerical Iteration (NI) method as well as the previous hierarchical bounding-box (BB) method [18] for comparison. Details of the test surfaces are shown in Table 1. 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. The sampling frequency on surface A for method NI is set to be *identical* to the sampling frequency for the highest level-of-detail bounding boxes in method BB, 64×64 samples per patch. This sampling rate determines the K_A and BB bounds. For the green-purple and yellow-orange pairs, surface B was subdivided into Bézier patches based on the original control point distribution. For the red-brown pair that has so few control points, an angle span flatness tolerance of 25° was used for the subdivision. We used a limit of five iterations or a difference between successive distance calculations (E_B) of less than .01 as the termination criteria.

Fig. 6 shows the computation time and Fig. 7 shows the corresponding Hausdorff distances at each position. It can be seen that the numerical iteration method has an approximately constant computation time irrespective of

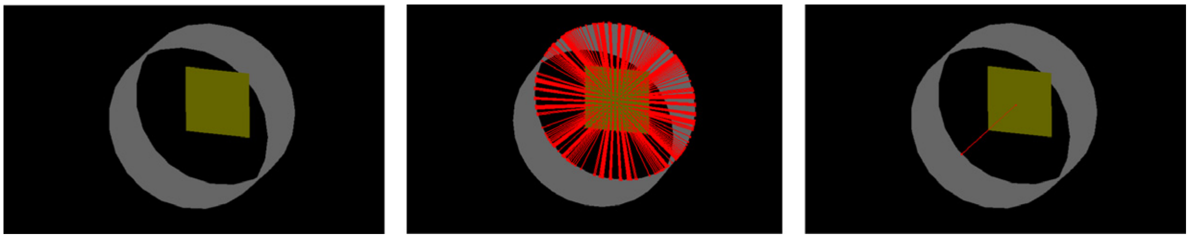

(a) *A cylinder with a plane patch intersecting its axis, the Hausdorff distance should be attained at the intersection of the axis and the patch.* **(b)** *The projection of sampled points onto the cylinder. The cylinder is located with its axis passing between sample points.* **(c)** *The Hausdorff distance attained by our method. It is attained at* $(8, 8, 0)$*, which is one of the points surrounding the origin.*

Fig. 5. Demonstration of a configuration where the computed Hausdorff distance almost achieves the theoretical K_A bound. The true Hausdorff distance is approximately 320, and the theoretical K_A bound is $8\sqrt{2} \simeq 11.3$. The Hausdorff distance computed with our method is 310.42.

the positions of the surfaces, unlike the BB method. The computation times are comparable between the two methods except for the green–purple surface pair, for which the NI method is slower because the surfaces have a large number of knots (approximately the number of control points) and hence they are split into many Bézier patches.

Fig. 8 shows the difference between the computed Hausdorff distance using our new iterative method and the computed Hausdorff distance using the bounding-box method at double the sampling rate, as an approximation of the error of the new method. (Unfortunately, we cannot compute the BB results at yet higher sampling rates for greater accuracy in the error calculation because of the memory limitations that cause this previous method's failure.) The graph shows that in practice, the differences are small and almost all of the values are within the bounding-box distance bounds, and within the K_A bounds.

5.2. Models with multiple surfaces

We also compared the computation time for computing the Hausdorff distances between two similar objects made of multiple surfaces, using the same stopping criteria. We used two objects, the Duck (Fig. 1d) and the Utah Teapot (Fig. 1e), made of 3 and 4 NURBS surfaces respectively. We used different flatness tolerances to split the NURBS surfaces into Bézier patches. "No" flatness tolerance indicates using only control points for splitting; the "loose" flatness tolerance was 45°; the "tight" flatness tolerance was 25°. As expected, a tighter flatness tolerance leads to the NURBS surface being split into more Bézier patches, which increases the computation time (Fig. 9). Note that the bounding-box method fails when the two Duck objects are really close to each other and hence the gaps in the computation time at these positions. (The failure is caused by running out of memory in cases where hierarchical culling fails to cull a sufficient number of bounding box pairs; for higher sampling rates this becomes increasingly problematic.) In addition, the BB method's computation time varies by a large value based on the relative positions of the two objects. The new method is both more consistent in its timing and more robust.

Table 1
Surfaces used for timing our Hausdorff distance algorithm. The surfaces are shown in Fig. 1. Size refers to the length of the bounding box diagonal.

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

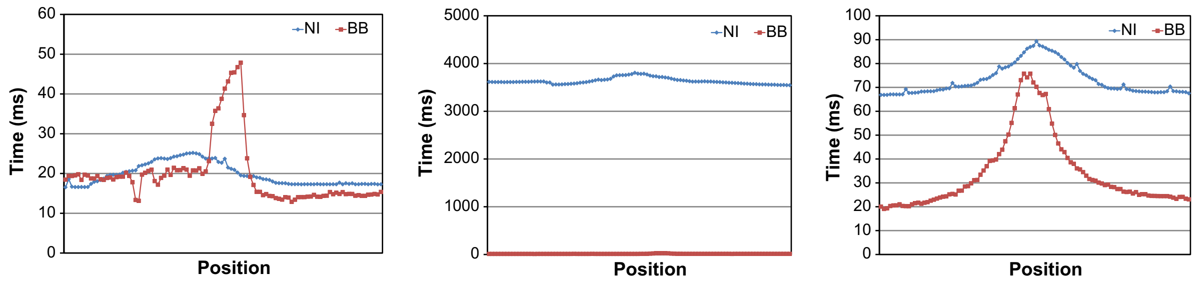
Fig. 10 shows the Hausdorff distances for the relative positions of the two objects corresponding to Fig. 9. Fig. 11 compares the difference in the computed Hausdorff distance between our new iterative method and the bounding-box method at different flatness tolerances.

There are several variables that influence the performance of our implementation and there are trade-offs between them. We have demonstrated some of them here, e.g., the influence of a tighter flatness tolerance, which increases the running time. Others include the influence of a higher sampling density, or of a higher value for the maximal number of iterations or of the user-specified K_B tolerance. We plan to investigate further these trade-offs with different input characteristics.

6. Discussion and future work

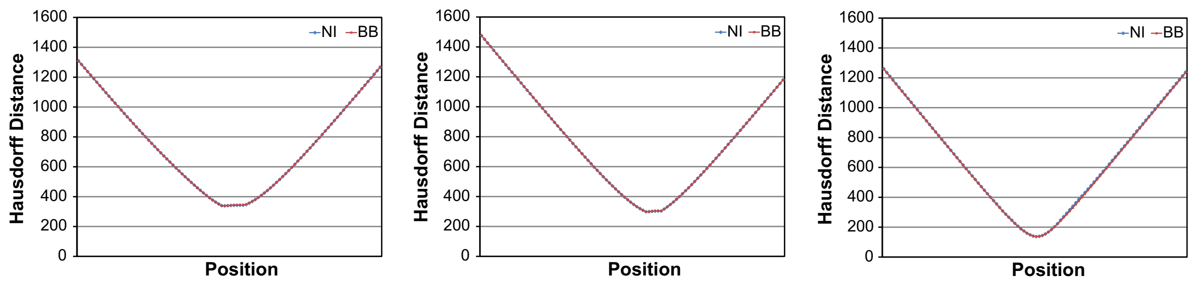
The results in Section 5 show that our new method has advantages and limitations compared to previous methods. One limitation of our implementation is its poor performance when a destination surface contains many knots and therefore must be subdivided into correspondingly many Bézier patches during the conversion process, even before applying the flatness criteria. For knots in relatively flat regions of the original surfaces, this will be needlessly costly in terms of running time, with no corresponding gain in accuracy (see Fig. 7b). The implementation decision to use Bézier patches in the GPU kernel and not NURBS patches was based on the fact that Bézier patches have a more regular structure that seemed a better match for the strength of the GPU.

However, in cases where the original NURBS surface has many knots, this results in a large number of Bézier patches. This is especially common in CAD models that were designed by spline interpolation (such as the surfaces in Fig. 7b). Future implementations should test the use of



(a) *Red-Brown* **(b)** *Green-Purple* **(c)** *Yellow-Orange*

Fig. 6. Comparison of the computation time between our method NI (Numerical Iterations) vs. the previous method BB (Bounding Box Hierarchy). Positions are matched with the corresponding graphs below.



(a) *Red-Brown* **(b)** *Green-Purple* **(c)** *Yellow-Orange*

Fig. 7. Comparison of the Hausdorff distance computed by our method NI (Numerical Iterations) and the previous method BB (Bounding Box Hierarchy).

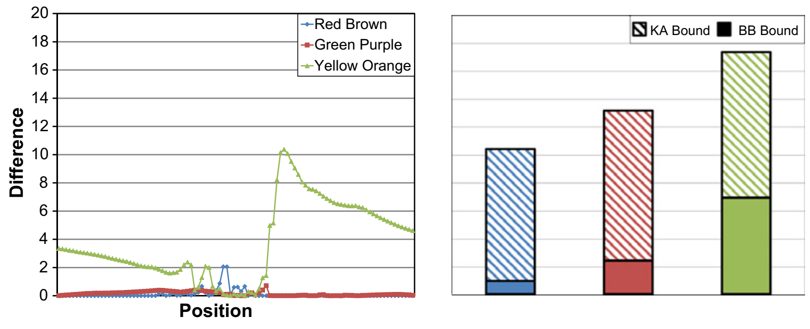


Fig. 8. Difference in the computed Hausdorff distance between the iterative and the bounding-box method. The respective bounds are shown on the right.

NURBS patches in the GPU kernel (i.e., splitting the surfaces into relatively flat NURBS patches regardless of the knot positions), and compare their performance with the current implementation.

Another approach that may mitigate this limitation is to implement a more sophisticated flatness-split criterion. Currently we check for the normal angle span and if it exceeds some value we split at the center parameter in u or v. In more sophisticated splitting criterion could be implemented, for example based on the curvature distribution on the patch.

In our current implementation we perform a uniform sample of points from surface $\mathcal{A}$ in the uv-space. A question that arises is the possibility of investing more preprocessing time in order to achieve a better distribution of points in Euclidean space. An improved distribution should achieve tighter bounds on the computed distance. A future direction in this regard can be to try and sample the points in some approximation of arc-length (or an analogous concept for surfaces).

In Section 4, we prove theoretical bounds on the exactness of our computation. As we discuss in Section 4.2.1, and as shown by our experimental results, these bounds are conservative in most but not all cases. A more exact bound might take into account the angle between the normals at projected points. However, computing such a bound for every projected point has an unwanted overhead. Proving such a theoretical bound and addressing computing it efficiently are a challenge that we leave for future work.

The method described in this paper is based on a numerical iteration scheme and therefore suffers from well-

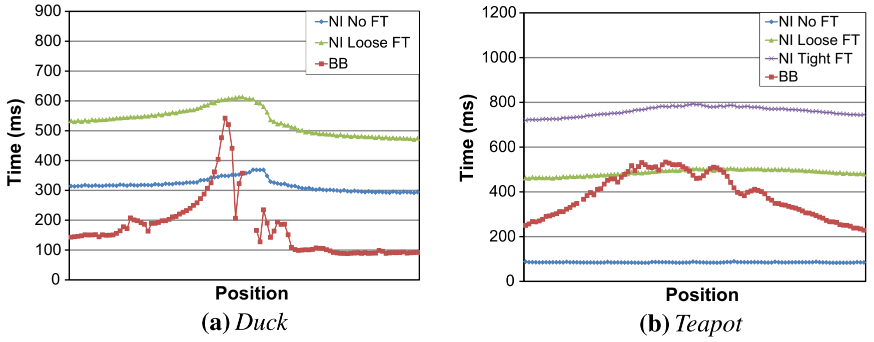


Fig. 9. Comparison of the computation time using different flatness tolerances for splitting the NURBS surfaces to Bezier patches. The computation time of the bounding-box method is also shown. Note the gap in (a), which is caused by a failure of the bounding-box method when the Duck objects are very close to each other.

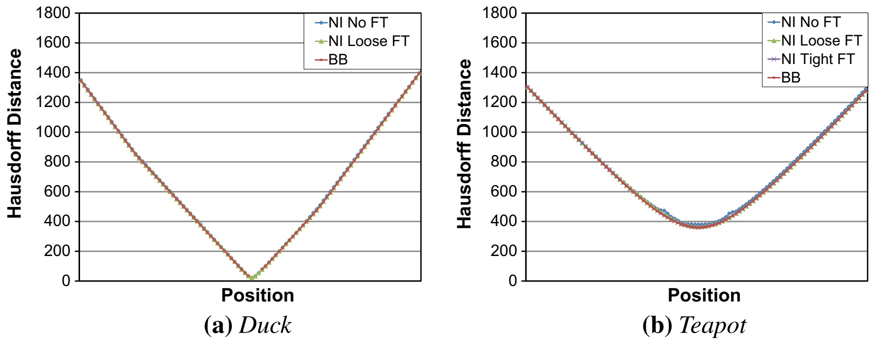


Fig. 10. Comparison of the computed Hausdorff distance values computed for the cases described in Fig. 9. Note the gap caused by the BB failure.

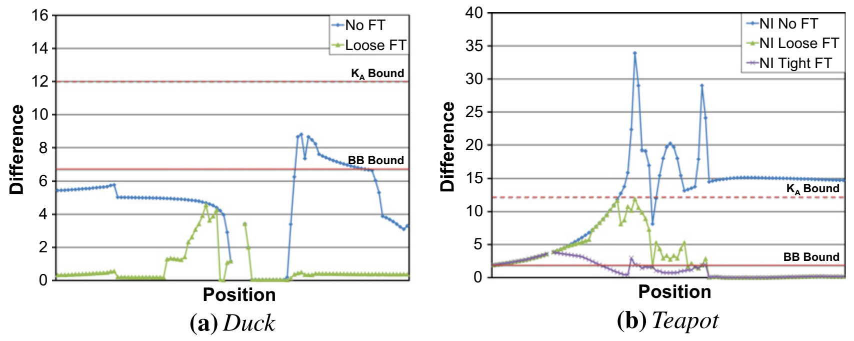


Fig. 11. Comparison of the difference between Hausdorff distances computed using different flatness tolerances with the bounding-box method.

known problems of numerical iterations. Particularly, numerical analysis texts contain examples of functions for which such methods do not converge (see for example, [16, Chapter 3.2]). In our context, this problem must be investigated further, because without convergence, the K_B bound cannot be guaranteed. The direction of the research into this problem can be in theoretical conditions for convergence, or in identifying cases for which the iterations fail to converge and handling them (e.g., by using a smaller flatness tolerance).

7. Conclusions

We have presented a novel algorithm, targeting the strengths of the GPU, for computing the directed Hausdorff distance between two NURBS surfaces. Unlike previous algorithms, which are generally based on BVH and linear approximation, our algorithm uses a new paradigm of parallel numerical iterations on the GPU. As discussed in Section 5, this approach has advantages and limitations compared to previous methods.

The big advantage over BVH methods is that unlike them, its running time does not increase as the two surfaces get closer. This is clearly demonstrated in Figs. 6 and 9, where the bounding box method [18] timing results peak as the Hausdorff distance decreases, whereas the runtime of our new method remains relatively constant (a similar peak behavior was recorded for the BVH method in [25]). Furthermore, BVH methods tend to "explode" in degenerate configurations (e.g., almost overlapping surfaces or between surfaces that are offsets of each other). This happened, for example, in the case shown in Fig. 9a. This advantage is even more meaningful, since many of the applications of computing the Hausdorff distance involve such near degenerate configurations. For example, measuring shape similarity between a model and its simplified geometry can contain such a configuration with a small Hausdorff distance.

We believe that the advantages of the approach described in this paper make it very suitable for applications requiring the computation of the Hausdorff distance. Furthermore, the idea of parallel numerical iterations on the GPU may be of use in other geometric problems involving freeform curves and surfaces.

Acknowledgments

We would 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. The first author gratefully acknowledges research support from the Lowengart Research Fund.

References

[1] H. Alt, B. Behrends, Approximate matching of polygonal shapes, Annals of Mathematics and Artificial Intelligence 13 (3–4) (1995) 251–265.

[2] H. Alt, P. Braß, M. Godau, C. Knauer, C. Wenk, Computing the Hausdorff distance of geometric patterns and shapes, in: Discrete and Computational Geometry, The Goodman–Pollack Festschrift, Algorithms and Combinatorics, 2003, pp. 65–76.

[3] H. Alt, L. Scharf, Computing the Hausdorff distance between curved objects, in: Proceedings of the 20th European Workshop on Computational Geometry, 2004, pp. 233–236.

[4] H. Alt, L. Scharf, Computing the Hausdorff distance between curved objects, International Journal of Computational Geometry and Applications 18 (4) (2008) 307–320.

[5] N. Aspert, D. Santa-Cruz, T. Ebrahimi, MESH: measuring errors between surfaces using the Hausdorff distance, in: Proceedings of the IEEE International Conference on Multimedia and Expo, vol. I, 2002, pp. 705 – 708. <http://mesh.epfl.ch>.

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

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

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

[9] G. Elber, E. Cohen, Second-order surface analysis using hybrid symbolic and numeric operators, ACM Transactions on Graphics 12 (2) (1993) 160–178.

[10] G. Elber, T. Grandine, 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, pp. 191–204.

[11] M. Guthe, P. Borodin, R. Klein, Fast and accurate Hausdorff distance calculation between meshes, Journal of WSCG 13 (2005) 41–48.

[12] I. Hanniel, G. Elber, Subdivision termination criteria in subdivision multivariate solvers using dual hyperplanes representations, CAD Journal 39 (5) (2007) 369–378 (Special issue, selected papers of Geometric Modeling and Processing, 2006).

[13] J. Hoschek, Approximate conversion of spline curves, Computer Aided Geometric Design 4 (172) (1987) 59–66.

[14] S.-M. Hu, J. Wallner, A second order algorithm for orthogonal projection onto curves and surfaces, Computer Aided Geometric Design 22 (2005) 251–260.

[15] Y.-J. Kim, Y.-T. Oh, S.-H. Yoon, M.-S. Kim, G. Elber, Coons BVH for freeform geometric models, ACM Transactions on Graphics (SIGGRAPH Asia) 30 (2011).

[16] D. Kincaid, W. Cheney, Numerical Analysis: Mathematics of Scientific Computing, second ed., Brooks/Cole Publishing Co., 1996.

[17] A. Krishnamurthy, R. Khardekar, S. McMains, Optimized GPU evaluation of arbitrary degree NURBS curves and surfaces, Computer Aided Design 41 (12) (2009) 971–980.

[18] A. Krishnamurthy, S. McMains, I. Hanniel, GPU-accelerated Hausdorff distance computation between dynamic deformable NURBS surfaces, SIAM/ACM Joint Conference on Geometric and Physical Modeling, ACM, 2011.

[19] B. Llanas, Efficient computation of the Hausdorff distance between polytopes by exterior random covering, Computational Optimization and Applications 30 (2005) 161–194.

[20] Y. Ma, W.T. Hewitt, Point inversion and projection for NURBS curve and surface: control polygon approach, Computer Aided Geometric Design 20 (2) (2003) 79–99.

[21] H. Pabst, J. Springer, A. Schollmeyer, R. Lenhardt, C. Lessig, B. Froehlich, Ray casting of trimmed NURBS surfaces on the GPU, in: IEEE Symposium on Interactive Ray Tracing, 2006, pp. 151–160.

[22] L.A. Piegl, W. Tiller, The NURBS Book, second ed., Springer, 1997.

[23] L.A. Piegl, W. Tiller, Parametrization for surface fitting in reverse engineering, Computer Aided Design 33 (8) (2001) 593–603.

[24] T.W. Sederberg, R.J. Meyers, Loop detection in surface patch intersections, Computer Aided Geometric Design 5 (2) (1988) 161–171.

[25] M. Tang, M. Lee, Y.J. Kim, Interactive Hausdorff distance computation for general polygonal models, ACM Transactions on Graphics 28 (2009) 74:1–74:9.

[26] Thrust, 2011. <http://code.google.com/p/thrust>.

[27] G. Varadhan, D. Manocha, Accurate Minkowski sum approximation of polyhedral models, Graphical Models 68 (4) (2006) 343–355.