GPU-Accelerated Minimum Distance and Clearance Queries

Adarsh Krishnamurthy, Sara McMains, Kirk Haller

Abstract—We present practical algorithms for accelerating distance queries on models made of trimmed NURBS surfaces using programmable Graphics Processing Units (GPUs). We provide a generalized framework for using GPUs as co-processors in accelerating CAD operations. By supplementing surface data with a surface bounding-box hierarchy on the GPU, we answer distance queries such as finding the closest point on a curved NURBS surface given any point in space and evaluating the clearance between two solid models constructed using multiple NURBS surfaces. We simultaneously output the parameter values corresponding to the solution of these queries along with the model space values. Though our algorithms make use of the programmable fragment processor, the accuracy is based on the model space precision, unlike earlier graphics algorithms that were based only on image space precision. In addition, we provide theoretical bounds for both the computed minimum distance values as well as the location of the closest point. Our algorithms are at least an order of magnitude faster and about two orders of magnitude more accurate than the commercial solid modeling kernel ACIS.

Index Terms—Minimum Distance, Closest Point, Clearance Analysis, NURBS, GPU, Hybrid CPU/GPU Algorithms

1 Introduction

Distance queries such as finding the minimum distance to a surface play an important role in many computer aided design and analysis applications, including tolerancing, clearance analysis, and accessibility analysis. Minimum distance queries are especially useful while designing complex assemblies to allow for sufficient clearance between different mechanical components. Such queries are easily answered if the objects or models are made of planar faces and have boxy shapes. However, modern designs make use of curved freeform surfaces; the standard representation of choice being Non-Uniform Rational B-Spline (NURBS) surfaces. Minimum distance queries on such freeform surfaces are currently being solved by commercial solid modeling software by first evaluating and tessellating the surface and then finding the minimum distance to the tessellation vertices [1]. This approach, in addition to being extremely slow and computationally intensive, is dependent on the tessellation resolution for the accuracy of the solution; the surface has to be very finely tessellated to get the required accuracy.

A technique to accelerate such slow geometric queries is to use programmable GPUs. We have developed a unified framework that uses GPUs as co-processors in accelerating geometric computations; we make use of the fragment processor in a GPU to perform parallel parts of the computations and use the CPU to perform the inherently serial parts. This framework can be extended to solve a wide range of geometric queries; we give a few practical examples of using this framework to answer distance queries. Previous GPU-based algorithms that render to the screen to perform these computations have restricted accuracy corresponding to the dimensions of the pixel or window. Our framework allows for the GPU algorithms to operate in the model space; therefore, the results of these geometric queries are accurate to any arbitrary user-defined tolerance.

Solid modeling kernels support certain distance queries such as the minimum distance from a point to a surface and the minimum distance between two surfaces. Applications of such distance queries include: finding the closest surface point on a surface to provide haptic feedback; dimensioning and tolerancing of CAD models; and constructing distance fields. In this paper, we present an algorithm that uses our hybrid CPU/GPU framework consisting of surface bounding-boxes to accelerate these queries. We focus on performing distance queries on objects made of trimmed NURBS surfaces in this paper. However, our algorithms are applicable for any surface that can be supplemented with a surface bounding-box structure. We provide theoretical bounds on the accuracy of both the computed minimum distance as well as the location of the closest point on the surface, which allow for arbitrary user-defined tolerance values. This is especially important in CAD systems since these distances might be used by the designer to define subsequent features; the model might fail to regenerate if there is an error in the computed distance.

In this paper, we provide a hybrid CPU/GPU framework to accelerate minimum distance computations, which is expanded from our previous conference presentation [2]. Our main contributions include:

- A practical GPU algorithm to find the minimum distance to a surface given any point in space. We use our hybrid framework to compute the distances efficiently in parallel using the GPU.
- A fast algorithm that computes the minimum distance between two surfaces or between two solid models rep-

- *A. Krishnamurthy, and S. McMains are with the Department of Mechanical Engineering, University of California, Berkeley, CA, USA. e-mail: {adarsh—mcmains}@me.berkeley.edu*
- *K. Haller is with SolidWorks Corporation, Concord, MA, USA. e-mail: khaller@solidworks.com*

Manuscript received Jan 22, 2010, revised Jun 12, 2010

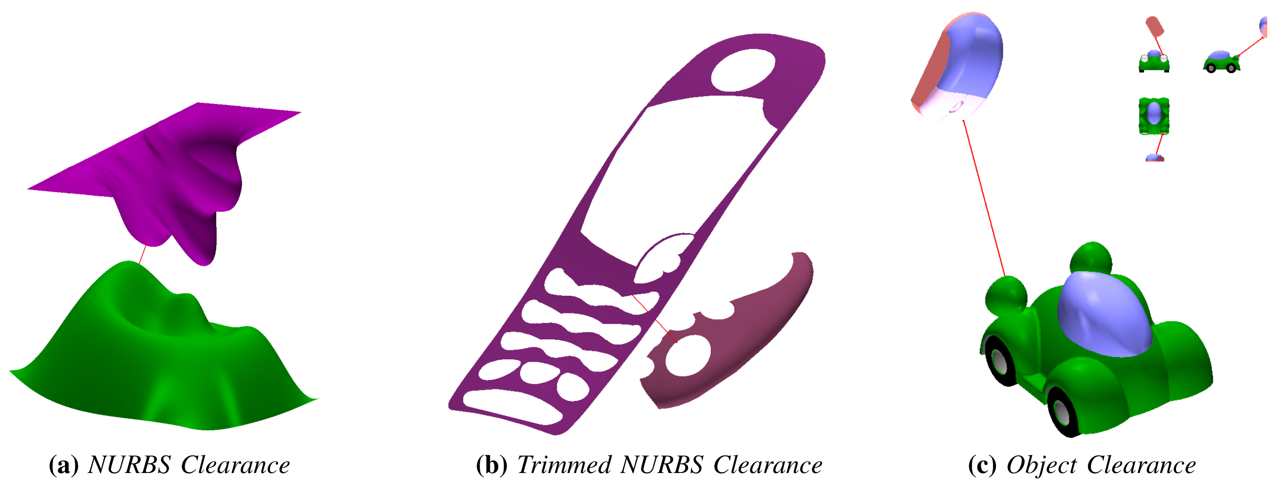


Fig. 1. Minimum distance/closest point computations between NURBS surfaces and complex CAD models accelerated using the GPU.

resented by B-reps, using bounding-box hierarchies on the GPU. Our algorithm is orders of magnitude faster and more accurate than the commercial solid modeling kernel ACIS in calculating these distances.

- An extension to the minimum distance computation algorithm to compute the minimum distance between two trimmed NURBS surfaces.
- A unified framework that uses the GPU as a co-processor to improve the performance of algorithms used for solving geometric queries. This framework can be extended to accelerate several related queries that are based on properties of the underlying shapes such as normals or curvature.
- Theoretical guarantees for all of our geometric computations. They allow for user-defined tolerance values that are essential for integrating our algorithms in a CAD system.

1.1 Hybrid Framework

We present a hybrid framework that can use both the CPU and GPU to perform geometric computations. The main idea is to split the computations into serial and parallel stages as shown in Fig. 2. To perform the parallel operations on the GPU, we make use of the map-reduce parallelism pattern that consists of assigning the computations to separate non-communicating parallel threads [3]. The inter-communication between the CPU and GPU is shown in Fig. 3. Once the computations are performed, the computed result can be used by the modeling system in the three different ways shown. Read-back is important for integrating the GPU algorithms with traditional modeling systems. In addition, since GPUs are designed for pipelining the data only in one direction from the CPU to the GPU for display, the method of read-back significantly affects the performance of hybrid algorithms. The most efficient method of read-back is reducing the results to a smaller set of values by using operations such as finding the maximum, minimum, sum, or by using non-uniform stream reductions ([4], [5]). The second method is to directly display the output on the screen using the GPU. This is ideal for certain operations that require only visual outputs; for example displaying the evaluated NURBS surface directly. The last and the most expensive method is to read back all the results from the GPU to the CPU; this might be required for certain computations where the result of a computation is required for further processing on the CPU.

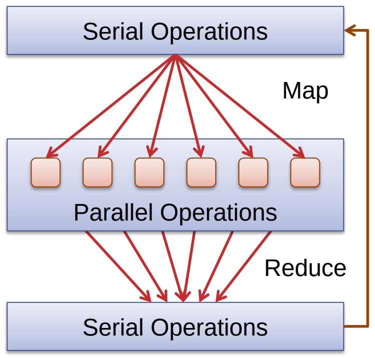


Fig. 2. Operation flow for performing geometric computations. The parallel operations are mapped and performed on the GPU while serial operations are performed on the CPU. The intermediate parallel output is reduced and read back to the CPU.

Our operations on the GPU fall into three main types. The first type includes parallel geometric computations that can be performed efficiently on the GPU. The outputs of such operations are usually numeric values that are then stored in the GPU as textures. If an operation produces more than one output value for each parallel operation, we can store those using separate channels of the same texture or using different textures. The second GPU operation type is parallel search operations that give a binary output of 0 or 1 based on the type of search; these include operations such as bounding-box intersection tests, finding if a value lies within a given range, etc. The third GPU operation type is reduction, which is performed using multiple passes on the GPU. GPU reductions can in turn be classified into two types. The first type, called standard reductions, include reducing the given input to a single value

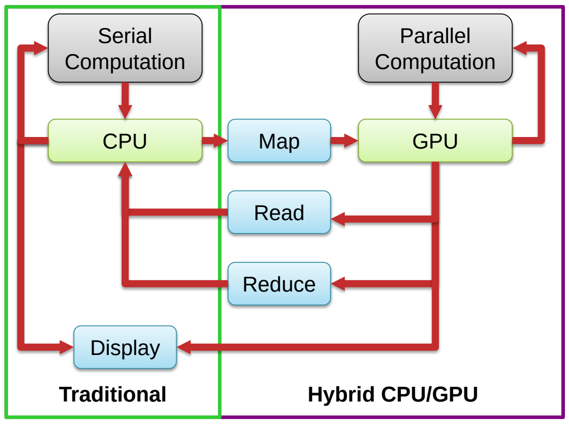


Fig. 3. Schematic showing our hybrid framework that extends traditional geometric computations to use the GPU as a co-processor to perform some parts of the computations in parallel.

such as computing the sum, min, max, etc. Standard reduction operations are usually performed in $O(\log n)$ passes and hence are very efficient. The second type of reductions, called non-uniform stream reductions, reduces the input to a smaller set of values. Non-uniform stream reduction operations are particularly important when the result of a reduction operation is not a single value but multiple values that satisfy a particular criterion. Since the positions of the output elements do not have any fixed correspondence with the positions of the input, the stream-reduction process is considered non-uniform. We make use of an $O(n)$ GPU stream-reduction algorithm that we presented in previous work [6] to perform non-uniform stream reductions.

To perform geometric computations on NURBS surfaces or assemblies, we make use of a surface bounding-box structure to map the computations to the GPU. We make use of Axis-Aligned Bounding-Boxes (AABBs) constructed from an evaluated mesh of points on the NURBS surface to accelerate the computations [6]. The main advantage of AABBs over Oriented Bounding-Boxes (OBBs) is that several geometric computations such as finding intersections and distances are simpler in the case of AABBs. This is especially important 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 has to wait until the most computationally intensive branch of the kernel in a particular pass is completed before proceeding to the next pass. In addition, since OBB kernels make use of more temporary registers, the number of computations that can be active simultaneously on the GPU (called fragments in flight) is reduced; it is difficult to hide the memory access latency in this case. Thus, we found that the advantage provided by tight OBBs is offset by the increase in complexity of the algorithms that use them. We achieve better results by using AABBs even if we must decompose the model to a finer resolution with AABBs than OBBs in order to maintain the same tolerance bounds.

2 Background and Previous Work

2.1 Related Work

Minimum distance computations are used by many algorithms that generate geometrical constructs such as Voronoi diagrams and medial axis transforms. They are also used in path planning and robot motion planning [7] and for projecting points onto a patch of a CAD model [8]. Minimum distance computations on curved NURBS surface are very time-consuming; hence, the commercial solid modeling system ACIS makes use of the tessellation of the surface to find the closest vertex or pair of vertices while performing tolerance analysis [1]. Johnson et al. [9] gave a unified framework for minimum distance computations, which was later extended to find the closest point for haptics applications by Nelson et al. [10]. We use a similar method that uses AABBs to find the regions of the model that are likely to contain the closest points. However, the methods they describe were better suited for a serial CPU implementation, since they make use of the convex hull of the freeform surface to iteratively refine the search. In our algorithm, the distance computations and search operations are done in parallel, which is better suited for a GPU implementation. In addition, we also provide theoretical guarantees for the solutions we compute.

Edelsbrunner [11] proved that the minimum distance between two convex polygons can be computed in $O(\log n)$. However, the algorithm used in the proof is theoretical and has large time-constants in practice. Quinlan [12] extended the minimum distance computations to non-convex objects by first performing a convex decomposition and then using bounding spheres for the convex pieces to create a hierarchy. However, this method is not practical for dynamic geometries since the convex decomposition might be expensive. Chen et al. [13] compute the minimum distance between a point and a NURBS curve by subdividing the curve into portions that might contain the closest point. Many minimum distance algorithms use Bounding Volume Hierarchies (BVHs) to accelerate the computations. CPU algorithms usually make use of BVHs that are more complex than AABBs. Gottschalk et al. [14] make use of OBBs to perform distance computations. Larsen et al. [15] perform proximity queries using a construct called a sphere swept volume, which consists of a sphere swept over a point, line or a plane, as primitives of a BVH.

Collision detection and distance field computation are two problems that are closely related to minimum distance computations that have been effectively accelerated using the GPU. Occlusion queries on graphics hardware were used by Govindaraju et al. [16] to detect collisions of polygonal meshes in large environments. Greß et al. [17] solve the collision detection problem by generating a bounding-box hierarchy for deformable parameterized surfaces and then detect collisions by checking overlap between the bounding-boxes using the GPU. Sud et al. [18] use the GPU to generate 3D distance fields by first slicing the model into 2D slices and by using culling and spatial coherence to reduce the number of distance computations in each slice. Lauterbach et al. [19] use the GPU to construct BVHs that can then be used to accelerate collision detection.

There has been only limited use of GPUs to perform geometric operations because they are restricted to image-space resolution if the computations are to be performed by rendering on the screen. Agarwal et al. [20] make use of the GPU to perform geometric computations on a stream of points by using point-line duality. They compute geometric properties such as diameter and width of a set of points. However, these algorithms are not stable for points that are very close and are limited to image-space resolution. Hoff et al. [21] use the GPU to perform fast proximity queries on 2D shapes using a pixel grid to perform distance computations, but their technique does not extend to 3D shapes. To overcome the image-space resolution for spline intersections, researchers at SINTEF adapted the serial subdivision algorithm to use the GPU. They accelerate the computations by using the GPU to test for intersections and iteratively subdivide the spline patches until a prescribed accuracy is attained [22], [23].

Prior algorithms for proximity queries on spline models used higher-order bounding volumes such as OBBs or swept spheres. Krishnan et al. calculate contact between spline models using a combination of bounding volumes that include spherical shells and OBBs [24]. Even though these higher-order bounding volumes have low memory requirements, the individual overlap computations are more complex. We make use of an AABB hierarchy in which the bounding boxes are not as tight as higher-order bounding volumes, but reduce the complexity of the computations for each bounding-box pair.

Our algorithm to compute the minimum distance between objects is an hybrid CPU/GPU algorithm that uses the CPU for certain computations that are inherently serial. Lauterbach et al. have recently developed a GPU algorithm where the hierarchy traversal and primitive queries are also performed on the GPU [25]. Even though such an exclusive GPU algorithm overcomes the CPU/GPU bottleneck, it requires newer hardware to perform the atomic operations on the GPU. However, these operations are not supported by all GPU hardware vendors; as a result, such algorithms will be difficult to be adopted by the CAD industry. Furthermore, parts of our algorithm that are performed on the CPU can be easily ported to the GPU when atomic operations are more widely supported by all GPU vendors.

2.2 NURBS Evaluation and Modeling

Our minimum distance computation and silhouette extraction algorithms build on our previous papers on GPU NURBS evaluation and modeling. We present a short outline of our algorithms that were explained in detail in [2], [6], [26], [27]. In our NURBS evaluation paper [27] we developed a method to directly evaluate a mesh of points on a NURBS surface using the GPU. Our algorithm used a fragment program to evaluate a NURBS surface of arbitrary degree in several passes. After evaluation we have the sampled NURBS surface as 4-component vectors—(x, y, z, w) coordinates—in space stored as a texture on the GPU. While rendering, we interpret these values stored in the texture as vertex coordinates using a Vertex Buffer Object (VBO) and display the VBO as a mesh directly on the screen.

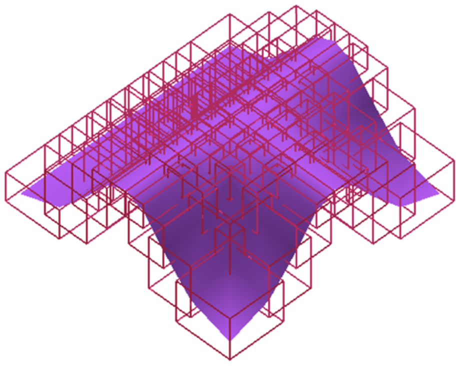


Fig. 4. Surface bounding-boxes constructed from points evaluated on a NURBS surface.

In our NURBS modeling work [2], [6], we construct surface AABBs that enclose a surface patch having four adjacent surface points as corners (Fig. 4). As a first step in the construction, we find the minimum and maximum coordinates of the four adjacent surface points to fit an AABB. However, the AABBs constructed by this method do not guarantee that the surface patch lies completely inside the constructed bounding-box. In order to guarantee complete coverage of the surface patch, we find the maximum possible deviation K of a curved surface from the linearized approximation, and then expand the bounding-boxes in all three dimensions by K (Fig. 5). The analytical expression for the factor that can be used to expand the bounding-boxes based on the surface curvature is given by Filip et al. [28]. They show that if a parametric C^2 surface is evaluated at $(n+1) \times (m+1)$ grid of points, the deviation of the surface from the piecewise linear approximation cannot exceed the constant K defined by Equations (1) – (4). We use this constant K in computing the bounds for our closest point algorithms.

$$M_1 = \max_{\forall(u,v)} \left[\max \left(\left| \frac{\partial^2 x}{\partial u^2} \right|, \left| \frac{\partial^2 y}{\partial u^2} \right|, \left| \frac{\partial^2 z}{\partial u^2} \right| \right) \right] \tag{1}$$

$$M_2 = \max_{\forall(u,v)} \left[\max \left(\left| \frac{\partial^2 x}{\partial u \partial v} \right|, \left| \frac{\partial^2 y}{\partial u \partial v} \right|, \left| \frac{\partial^2 z}{\partial u \partial v} \right| \right) \right] \tag{2}$$

$$M_3 = \max_{\forall(u,v)} \left[\max \left(\left| \frac{\partial^2 x}{\partial v^2} \right|, \left| \frac{\partial^2 y}{\partial v^2} \right|, \left| \frac{\partial^2 z}{\partial v^2} \right| \right) \right] \tag{3}$$

$$K = \frac{1}{8} \left(\frac{1}{n^2} M_1 + \frac{2}{nm} M_2 + \frac{1}{m^2} M_3 \right) \tag{4}$$

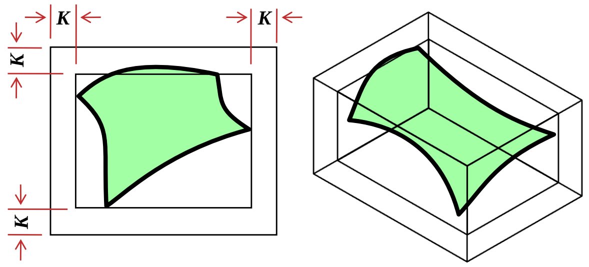


Fig. 5. We expand the AABBs by K in all three dimensions to guarantee that the surface patch is completely enclosed.

An alternative approach for constructing an AABB hierarchy for the NURBS surface is to recursively construct the bounding-boxes by evaluating surface points using knot insertion. This method has the advantage that it can be adaptively refined based on the curvature of the surface. However, implementing this method on the GPU is tedious since the number of bounding-boxes in the finest level of the hierarchy is not known a priori. In addition, recursive algorithms do not achieve optimal performance when implemented on the GPU. Hence, we construct AABBs as explained above.

3 Distance Queries on NURBS Surfaces

We first present distance queries that are performed on individual NURBS surfaces and later in Sec. 5 extend them to complex objects made up of multiple curved surfaces.

3.1 Minimum Distance to a NURBS Surface

The first distance query we accelerate using the GPU is computing the minimum distance and the closest point on a NURBS surface given any point in space. As a first step, we evaluate the NURBS surface as a grid of points using our NURBS evaluator and construct surface AABBs enclosing four neighboring points. Using these bounding-boxes and the input point, we calculate the range of distances to each bounding box as explained in Sec. 3.2.

Fig. 6 shows how our GPU closest point algorithm fits into our hybrid framework. We first use the GPU to compute the minimum and maximum distance to each AABB efficiently in parallel. These distances are stored using the red and green channels in a min/max texture on the GPU. We then perform a parallel reduction in $\log n$ passes on the GPU to find the bounding-box with the minimum lower value for the distance range. We read back the range of this particular bounding-box. In the next pass, we use the upper bound of this particular bounding-box as a distance cutoff to search for potentially close bounding-boxes. We use the GPU to perform a parallel search on the same min/max texture we computed in the first step to find all the bounding-boxes whose ranges overlap with the upper bound. This prunes the list of bounding-boxes to search for the closest point; we read back this smaller list by performing non-uniform stream reduction on the results of the search.

Once we read back the potentially close bounding-boxes, we approximate the surface patch inside each of the bounding boxes with two triangles formed from the evaluated surface points. We then find the distance to each of these triangles and finally choose the one with the minimum distance. We also find the point lying on the triangle that has the minimum distance as the closest point on the surface. We prove theoretical error bounds for the evaluated minimum distance and the calculated closest point in Section 4.

3.2 Minimum and Maximum Distance to an AABB

The first step of our minimum distance algorithm requires the computation of the minimum and maximum distance between a point and an AABB. Since we want to perform these

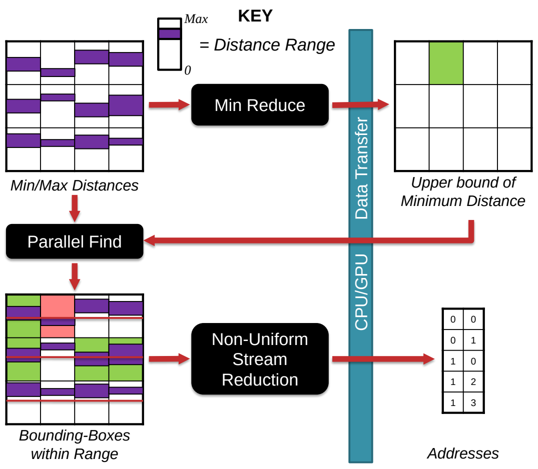


Fig. 6. Schematic of our closest point algorithm showing the inter-communication between the CPU and GPU. The vertical bars represent the range of minimum and maximum distances from the point to the bounding box.

computations in parallel for each AABB, the computations have to be efficient and optimized for the GPU. The maximum distance can be computed in a straightforward manner by finding the vertex of the bounding-box that is farthest from the given point. However, to compute the minimum distance, we not only need to find the minimum distance to the vertices of the AABB but also to the faces. The number of computations becomes prohibitively many if we have to check all the possibilities.

We make use of the fact that the bounding-boxes are axis-aligned to efficiently compute the minimum and maximum distance. This makes the calculations simpler and unified for computing both the minimum and maximum distance simultaneously (Fig. 7). For computing the maximum distance from a point O to an AABB, we compute the maximum distance along each axis separately and finally take the L_2 norm of the individual maximum distances to find the maximum distance (Equations (5) – (8)). However, if we extend the same method to compute the minimum distance, we have to make sure that the individual distance components are non-zero; if we directly subtract the half bounding-box widths, we will end up with negative distances. To overcome this, we take the minimum distance along a particular direction as zero if it is negative (Equations (9) – (12)).

$$x_{max} = D_{cx} + B_x \tag{5}$$

$$y_{max} = D_{cy} + B_y \tag{6}$$

$$z_{max} = D_{cz} + B_z \tag{7}$$

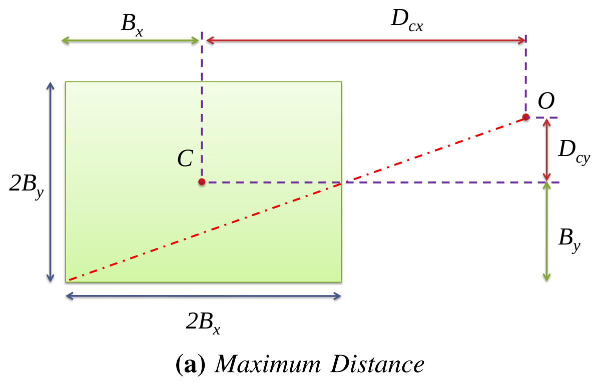
$$D_{max} = \sqrt{(x_{max}^2 + y_{max}^2 + z_{max}^2)} \tag{8}$$

$$x_{min} = \max(D_{cx} - B_x, 0) \tag{9}$$

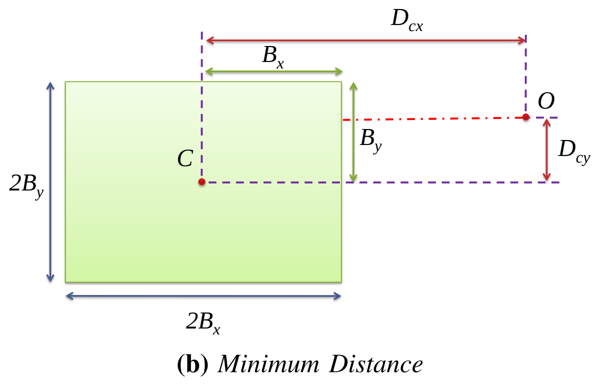
$$y_{min} = \max(D_{cy} - B_y, 0) \tag{10}$$

$$z_{min} = \max(D_{cz} - B_z, 0) \tag{11}$$

$$D_{min} = \sqrt{(x_{min}^2 + y_{min}^2 + z_{min}^2)} \tag{12}$$



(a) *Maximum Distance*



(b) *Minimum Distance*

Fig. 7. Efficiently computing the maximum and minimum distance between a point and an AABB. The example shown here is for the 2D case, but the method can be extended to 3D. See Equations (5) – (12).

This formulation is efficient for GPU implementation, since it has only one branch for each $\max$ while computing the minimum distances. We implement these equations using a single fragment program and output the minimum and maximum distance to a texture using the red and green channels. Thus, the minimum and maximum distances are computed simultaneously for all AABBs in parallel. We then use these min/max distances as the input texture for finding the minimum distance to a NURBS surface (Fig. 6) as explained in Sec. 3.1.

4 Theoretical Bounds

In this section, we give theoretical bounds for both the computed minimum distance and the location of the closest point on the curved surface given any point in space.

Theorem 1. (Minimum Distance Bound) *The computed minimum distance does not deviate from the theoretical minimum distance to the actual surface by more than the surface deviation value K.*

Proof: Let O be the point from which we want to find the minimum distance to a curved surface patch S showed in green in Fig. 8. Let A_1, A_2, A_3 be three points (of the four points used to construct the bounding-box) evaluated on the surface. The surface can be approximated linearly by triangle $A_1A_2A_3$; the maximum deviation of the linear approximation from the curved surface is K (Eqn. (4)). Let Q be the actual point closest to O on the curved surface and P' be the computed closest point on the triangle. Let P be the closest point to P' on the surface. Since Q is the closest point on the surface from O, $OQ < OP$. From triangle OPP', by applying the triangle inequality to the sides, we get $OP < OP' + PP'$. Since the maximum deviation of the surface from the triangle is K, distance $PP' < K$. Combining these inequalities, we get $OQ < OP' + K$ or $OQ - OP' < K$. This shows that the distance OQ, the theoretical minimum distance, cannot be larger than the computed distance OP' by more than K.

Now, consider the point on the triangle that is closest to Q, call it Q'. In this case $OP' < OQ'$ since P' is the closest point on the triangle from O. Again from triangle OQQ', we get $OQ' < OQ+QQ'$ and $QQ' < K$ since Q' is the closest point on the triangle from Q. Combining these three inequalities,

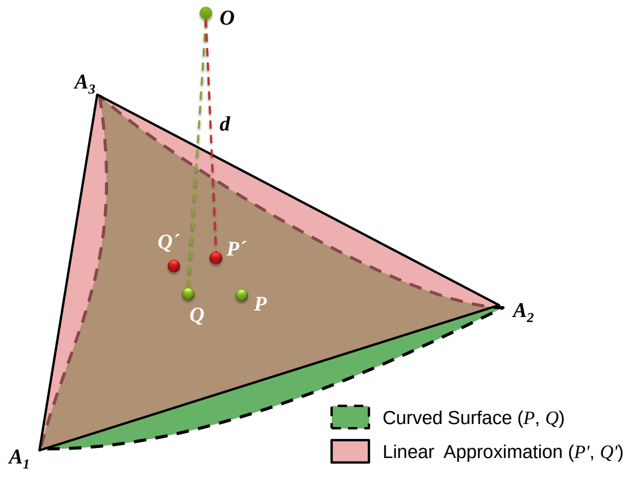


Fig. 8. Illustration to prove the bound for the minimum computed distance. The actual surface is shown in green while the linearized approximation is shown in orange.

we get $OP' < OQ + K$ or $OP' - OQ < K$. This shows that the theoretical minimum distance cannot be smaller than the computed distance by K. Combining the minimum and maximum bound on the distance, we get $|OP' - OQ| < K$. □

Thus, from Theorem 1, we know that the theoretical minimum distance is bounded to lie within the range $(d-K, d+K)$, where d is the computed minimum distance. We now show how we use this bound to prove that the location of the closest point we compute is also bounded.

Theorem 2. (Closest Point Location Bound) *The maximum possible distance between the computed closest point and the theoretical one is $\sqrt{4Kd + K^2}$ where d is the computed minimum distance to the surface.*

Proof: From Theorem 1, the theoretical minimum distance cannot deviate from d by more than K, i.e. $OQ \in [d-K, d+K]$. We have two possible cases: the closest point P' computed on the plane lies inside the triangle used to approximate the surface or it lies on one of the edges of the triangle (see Fig. 9(a) and Fig. 9(b), which show a 2D cross-section). In the first case (Fig. 9(a)), the minimum distance bound restricts the theoretical closest point Q to lie in an

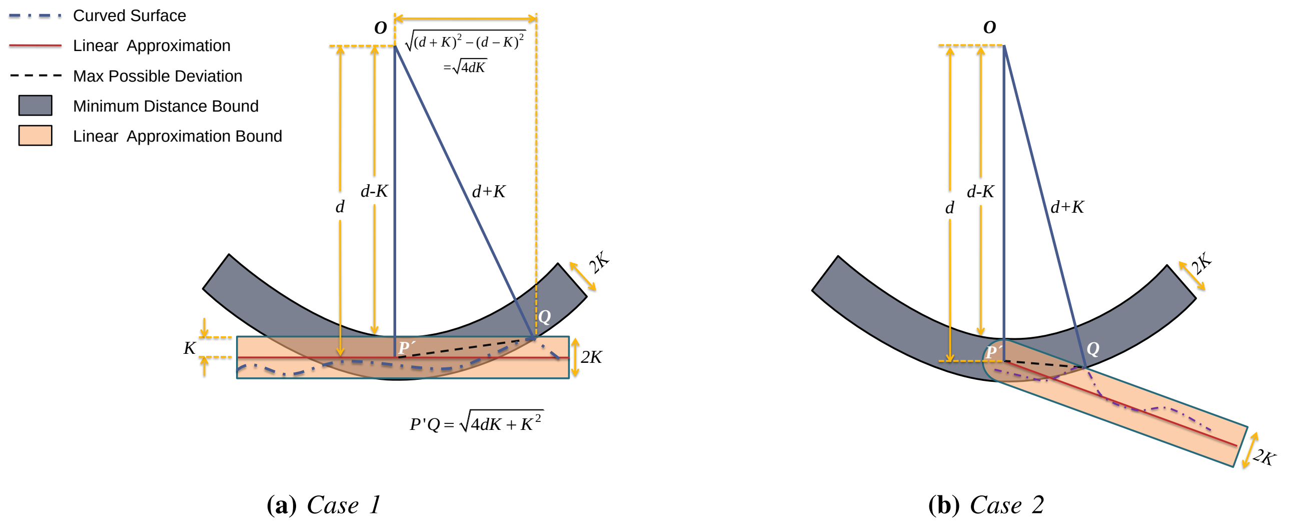


Fig. 9. Illustration to evaluate the bound for the computed closest point location when the closest point on the plane lies either (a) inside or (b) on the edge of the triangle approximating the surface.

annular region between spheres with center O and radii $d+K$ and $d-K$ (marked in blue). From our tessellation bound K, we know that the actual surface lies within a region of width $2K$ centered around the approximating triangle (marked in red). Thus the point Q lies in the intersection of these overlapping regions. The maximum possible distance $P'Q$ in this intersecting region is $\sqrt{4Kd+K^2}$. In the second case (Fig. 9(b)), the approximating triangle is oriented at an obtuse angle with respect to OP'. In this case, the maximum distance in the overlapping region occurs only when OP' is perpendicular to the triangle; for all other angles of rotation of OP', it is always less than $\sqrt{4Kd+K^2}$ (please refer to the Appendix for a detailed explanation). Hence, the maximum possible distance between the computed closest point and the theoretical one is always $\sqrt{4Kd+K^2}$. □

Thus, both the computed minimum distance and the location of the closest point are bounded. We show in the Results section that these theoretical bounds translate to realistic values that are useful in practice. Next, we extend our minimum distance computations to compute the minimum distance between two NURBS surfaces or two complex CAD objects represented as B-reps.

5 Clearance Analysis

5.1 Minimum Distance Between Two NURBS Surfaces

We use a method similar to finding the minimum distance from a point to a surface to find the minimum distance between two surfaces. However, it is impractical to use this method directly because the number of distance comparisons increase as $O(n^2)$, where n is the number of AABBs of each surface. Therefore, we make use of a method that uses bounding-box hierarchies to successively refine the number of potentially-close bounding-box pairs. We show that this approach, which is similar to a breadth-first search, can also be fit into our hybrid framework. We perform the search for potentially-close bounding-box pairs in parallel at each level using the GPU.

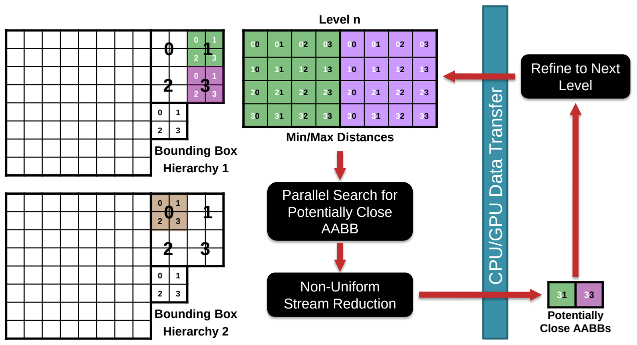


Fig. 10. We perform minimum distance computation between two NURBS surfaces with the help of AABB hierarchies for both the surfaces. We compute a list of potentially close bounding-boxes at each level using the GPU and then refine on the CPU until we reach a set of potentially close bounding-boxes at the lowest level.

We first construct surface AABBs as shown in Fig. 4; denote these as original AABBs. We then generate a bounding-box hierarchy by recursively combining four AABBs in a level to get a bigger AABB of the next higher level. Thus, we construct an AABB hierarchy starting with the original AABBs and finally reaching a single, level-0 bounding box. This operation can be effectively performed in $O(\log n)$ passes using the GPU. We store the bounding-boxes in a manner that optimizes GPU storage space (Fig 10) similar to mip-map layouts. When the model is transformed (translated or rotated), we fit new AABBs that contain the transformed original AABBs and rebuild the hierarchy. However, we still store and use the original AABBs for fitting after every transformation, since if we keep only the newly fitted AABBs, the bounding-boxes will keep growing in size.

We compute the minimum distance between the surfaces by recursively going down the hierarchy and finding potentially-close bounding-boxes at the finest level of the hierarchy. We start at level 1 of the hierarchy where we compute the

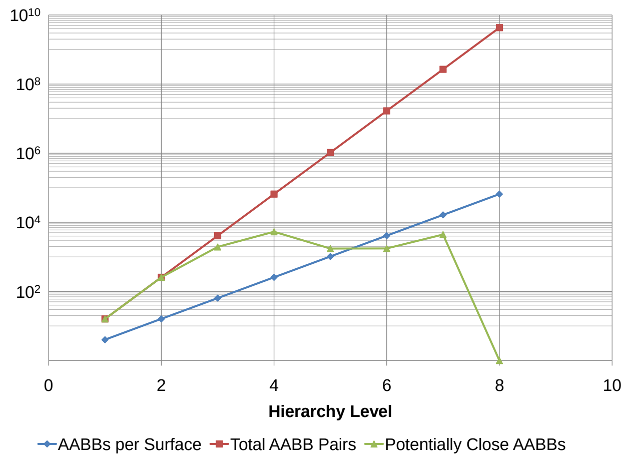


Fig. 12. Plot showing the actual number of AABB pairs compared during a typical minimum distance computation. The number of pairs being tested in parallel remains almost constant after level 3 of the hierarchy. Note logarithmic scale used for the y-axis.

minimum and maximum distance between four AABBs from surface 1 with each of the four AABBs of level 1 from surface 2, a total of 16 minimum and maximum distance pairs. The method used for finding the minimum and maximum distance between two AABBs is explained in Section 5.2. Once we compute the set of minimum and maximum distances, we prune those AABB pairs that are outside the min/max distance range of the closest AABB pair, similar to our method described in Section 3.1. We get a list of potentially-close AABB pairs for this level of the hierarchy at the end of the search. We then use the GPU to map the next finer level of the hierarchy, in sets of 4×4 AABB pairs, and repeat finding the potentially-close AABB pairs in the next finer level on the GPU (Fig. 10). Finally at the end of the recursion, we get a list of potentially-closest AABB pairs in the finest or highest level of the hierarchy of both the surfaces. Using a hierarchy to prune AABBs outside the range keeps the number of potentially-close AABB pairs almost constant. Fig 12 shows that the number of pairs to be tested increases at first and after level 3 remains almost constant at a few thousand potentially-close pairs. These computations can be done efficiently by the GPU in parallel at each level, as seen in the Results section.

Finally, once we obtain all the potentially-closest AABB pairs at the highest level, we compute the closest distance between the surface patches enclosed by these AABBs on the CPU since the list of pairs is usually small. We approximate each surface patch with two triangles and then compute the minimum distance between the triangles. Similarly, we also compute the pair of closest points that have the minimum distance between them.

5.2 Minimum and Maximum Distance Between AABBs

We extend our computations described in Sec. 3.2 to compute the minimum and maximum distance between two AABBs (Fig. 11). Similar to the point case, we compute the minimum and maximum distance along each dimension and then calculate the overall minimum and maximum distances (Equations (13) – (20)). As before, if any component is negative while computing the minimum distance, we take that component as zero.

$$x_{max} = D_{cx} + B_{1x} + B_{2x} \tag{13}$$

$$y_{max} = D_{cy} + B_{1y} + B_{2y} \tag{14}$$

$$z_{max} = D_{cz} + B_{1z} + B_{2z} \tag{15}$$

$$D_{max} = \sqrt{(x_{max}^2 + y_{max}^2 + z_{max}^2)} \tag{16}$$

$$x_{min} = \max(D_{cx} - B_{1x} - B_{2x}, 0) \tag{17}$$

$$y_{min} = \max(D_{cy} - B_{1y} - B_{2y}, 0) \tag{18}$$

$$z_{min} = \max(D_{cz} - B_{1z} - B_{2z}, 0) \tag{19}$$

$$D_{min} = \sqrt{(x_{min}^2 + y_{min}^2 + z_{min}^2)} \tag{20}$$

These equations are implemented using a fragment program on the GPU; we output the values to the red and green channels of a texture. The distances are computed for all potentially-close AABB pairs at a particular level in parallel and are then used for finding the potentially-close AABB pairs in the next level as explained in Sec. 5.1.

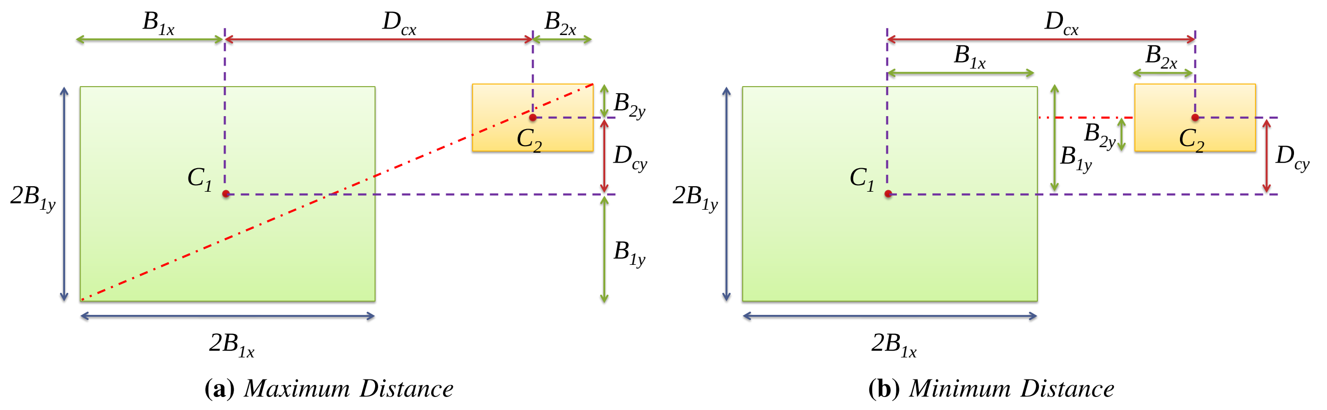


Fig. 11. Computing the maximum and minimum distance between two AABBs. The equations are similar to the point-AABB distance case. See Equations (13) – (20)

5.3 Minimum Distance Between Two Trimmed NURBS Surfaces

We extend our NURBS minimum distance computations to find the minimum distance between two trimmed NURBS surfaces. In order to address trimmed NURBS surfaces, we have to cull the bounding-boxes that lie in the trimmed regions of the surface. We generate bounding-boxes for a set of four points evaluated on the surface only if any of the four points lie outside the trimmed region. Since we store the bounding-box data using two four-channel floating-point textures, we can indicate whether the bounding-box is valid by using the fourth alpha channel in the texture. If all the four points lie inside the trimmed region of the NURBS surface, we cull the bounding-box by setting the alpha channel of the corresponding bounding-box texels to zero.

To perform this culling operation, we first generate a trim-texture [27] of the same size as the evaluated points. This gives a one-to-one correspondence for checking whether an evaluated point lies inside the trimmed region. While generating the bounding-boxes for the surface patches on the GPU, an extra test is performed to check if every set of four points lie inside the trimmed region in the parametric space. If so, the bounding-box is culled by setting the alpha channel to be zero; otherwise, it is set to one. Fig. 13 shows surface bounding-boxes for a trimmed NURBS surface that are not culled. Once we generate the bounding-boxes, we construct the bounding-box hierarchy similar to the method explained in Sec. 5.1. However, we combine only the bounding-boxes that are not culled to generate the hierarchy.

After we have generated the bounding-box hierarchy, we use the same algorithm given in Sec. 5.1 to find a list of potentially-close bounding-box pairs. While performing the search operation, we make sure that we do not include the bounding-boxes that are culled in the calculations. Finally, once we obtain all the potentially-closest AABB pairs at the highest level, we approximate each surface patch with two triangles and then compute the minimum distance between the triangles. However, we have to do an additional check to make sure that the computed closest point lies outside the trimmed region of the surface. If the point lies inside the trimmed region, we discard the point and continue the search.

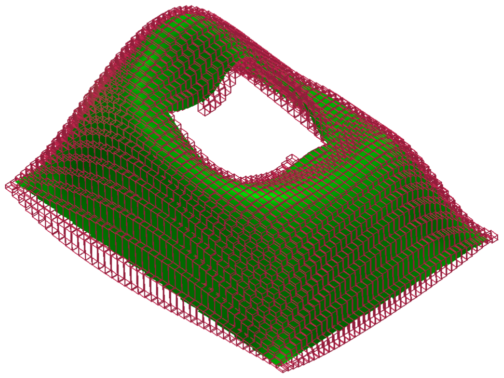


Fig. 13. Example of non-culled surface bounding-boxes for a trimmed NURBS surface. The bounding-boxes that lie in the trimmed region are culled.

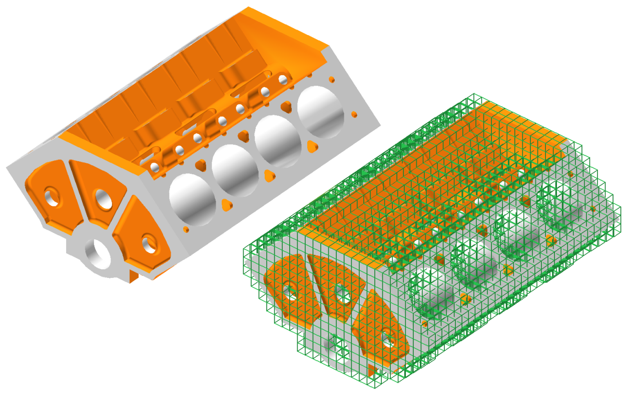


Fig. 14. A complex model and its voxel representation. We store the surfaces that intersect with a particular voxel to accelerate the minimum distance computation.

A main implication of the presence of trims is that we cannot always guarantee as tight a tolerance for the minimum distance as in untrimmed surfaces. There are two cases where the tolerances may be looser. The first case happens when the closest point lies on the edge of a trim-curve. In this case, since we do not explicitly find the intersection of the trim-curves and the surface patches that lie inside the closest bounding-boxes, the tolerances calculated for the untrimmed surface cannot be used. However, the tolerance values are still guaranteed to be less than the size of the bounding-box that contains the surface patch. The second case is the degenerate case when the closest point is on an untrimmed feature that is smaller than the parametric tolerance used for creating the trim-texture and the corresponding bounding-box is culled as a result. In this case, since we use the same trim-texture for display, the small feature will also display as trimmed away. When this happens, the user will get visual feedback that the model has not regenerated correctly and can adjust the parametric tolerance accordingly.

5.4 Minimum Distance Between Two Complex Objects

Finally, we extend our minimum distance computations between NURBS surfaces to complex objects made up of many NURBS surfaces. CAD systems have support for this query to give feedback about the clearance between the models in an assembly while the user is manipulating them. However, existing systems are not interactive due to long computation times for performing this query. We perform this query in two stages; in the first stage we find a list of potentially close surface pairs and in the second stage we find the minimum distance between the surfaces.

Voxel-based First Stage

In the voxel-based approach for the first stage, we construct a grid of voxels in the region occupied by the object (Fig. 14). We then consider these voxels as individual AABBs to perform the minimum distance computation. We create the voxel representation of the model as a preprocessing step. We first overlay a regular voxel grid that covers the object completely.

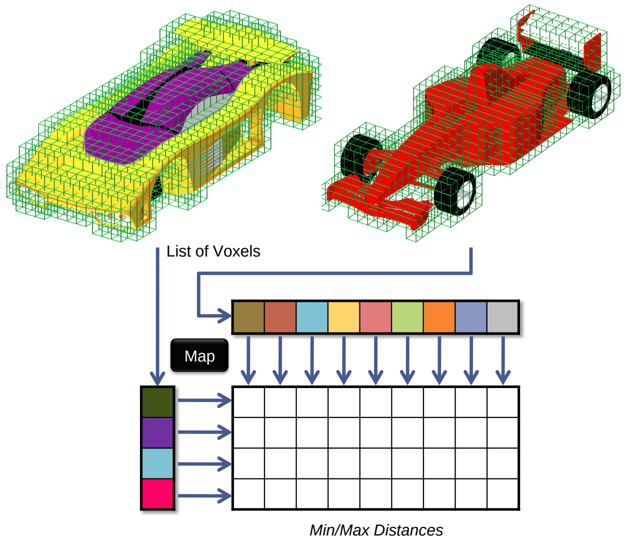


Fig. 15. We map the list of voxels of one object to the rows and the other object to the columns of a 2D texture to compute the minimum and maximum distances between the voxels.

We then use the coarse tessellation of the object that is used for display to populate the voxel grid. For each triangle in the tessellation, we find the voxels that the triangle intersects and then add a reference in the voxel to the surface to which the triangle belongs. Thus, each voxel has information about its minimum and maximum point extents that define the AABB and a list of surfaces that intersect it. Since this is done only once per object when the object is loaded for display, we perform this operation on the CPU. In addition, since this is a linear $O(n)$ operation, where n is the number of triangles in the tessellation, it is fast.

As a first step in finding the closest points, we find a set of potentially-close voxel pairs by performing a single pass of minimum distance computation. To perform this operation on the GPU, we map the voxels from the first object to the rows and the voxels from the second object to the columns of a 2D texture (Fig. 15). We compute the minimum and maximum distances for each voxel pair of the two objects and output these distances to the texture. This texture is then used to find the list of potentially-close voxel pairs that lie within the range of the closest voxel pair (as in Fig. 6). We perform non-uniform stream reduction to transfer address information of the potentially-close voxel pairs to the CPU. Since each voxel has information about the surfaces that pass through it, we can create a list of potentially-close surface pairs from these potentially-close voxel pairs. We also make sure that there are no duplicated entries in the surface pairs list, since the same surface can pass through many voxels in the potentially-close voxel pair list.

Surface-based Second Stage

In the second stage, we compute the minimum distance for each surface pair in the potentially-close surface list using our algorithm explained in Sec. 5.3 or Sec. 5.1, depending on whether the surface is trimmed or not, respectively. We can then output the minimum distance or clearance between the two objects as the minimum distance computed from all the surface pairs. We also output the points on each surface as the closest points on the two objects. Even though we use the coarse tessellation for constructing the voxel grid, we do not use it for the minimum distance computations. Our computations are performed using the NURBS surfaces directly and lie within the computed bounds. Hence, they are more accurate than only using the tessellation for the computations. In the Results section we show that our algorithm performs orders of magnitude faster than a commercial CPU-based kernel.

6 Results

We timed our GPU-accelerated distance queries on a 2.66GHz (quad-core) CPU running Windows Vista with 4GB of RAM and an NVIDIA Quadro FX5800 with 4GB graphics memory. We compare our timings to perform the geometric queries with those of the commercial solid modeling kernel ACIS (v20).

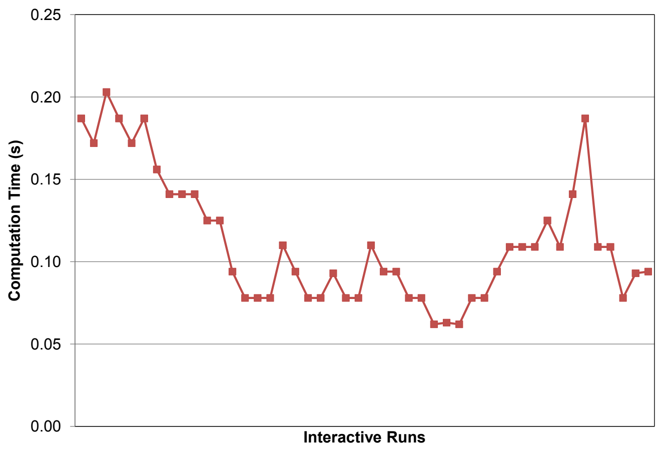
NURBS Minimum Distance Timings

We timed our minimum distance computations between two curved NURBS surfaces by interactively translating as well as rotating one surface made of 199×33 control points relative to the another surface made of 100×105 control points (Fig. 1(a)). Fig. 16(a) shows the interactive computation times recorded during the interaction; the computation times were less than 0.15 seconds for most positions, a near-interactive average frame rate of 9.07 fps. Fig. 16(b) shows the distance and position tolerances computed corresponding to the runs in Fig. 16(a). Since these tolerance values are dependent on the model size, we report them as a fraction of the model size in order to make them consistent with tolerance definitions used by ACIS [29]; a value of 0.01 corresponds to 1% of the model size. The model size is the length of the diagonal of the smallest AABB that will enclose the model.

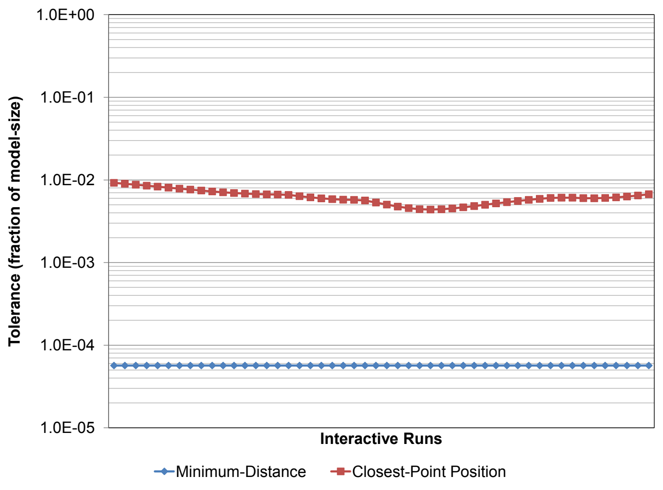
Position	ACIS Time (s)	GPU Time (s)	Speed-up
1	64.4	0.218	296x
2	65.4	0.109	600x
3	66.9	0.093	720x
4	66.2	0.171	387x

TABLE 1
Time for performing minimum distance computations between two NURBS surfaces.

We recorded the time taken by ACIS to compute the minimum distance at some arbitrarily chosen positions of the NURBS surfaces relative to one another by using the api command *api_check_face_clearance*. We set the tolerance value for ACIS to be 4×10^{-2}, well looser than our closest-point position tolerances reported in Fig 16(b). We chose this tolerance because we can guarantee it in our algorithm even if the closest point lies on a trim-curve edge; the tolerances can be guaranteed to be much tighter if the closest point does not lie on a trim-curve edge. Table 1 summarizes the results of our NURBS minimum distance computations for these positions, including the positions where our algorithm was slowest (note that there is little variation in the ACIS timings for different positions). The GPU accelerated algorithm is at least two



(a) *Computation Time*



(b) *Tolerance*

Fig. 16. Interactive times for evaluating the minimum distance between two NURBS surfaces and the corresponding distance and position tolerances scaled with respect to the maximum model size.

orders of magnitude faster than ACIS. This can be explained by the fact that ACIS first tessellates the object to get a dense mesh of points on the surface and then performs the minimum distance computation on these points. We on the other hand use our fast NURBS evaluator to evaluate the surface and construct surface bounding-boxes in real time. In addition, we not only achieve better performance but also a higher accuracy; our results have theoretical bounds that are practical for use in a CAD system.

Table 2 lists the break-down of the timings for performing different operations while computing the minimum distance between two NURBS surfaces shown in Fig. 1(a). It can be seen that evaluation and hierarchy traversal operations have similar run times.

Operation	Time (s)
NURBS evaluation	0.036
Normal evaluation	0.038
Bounding-box construction	0.019
Bounding-box hierarchy construction	0.008
Total evaluation time	0.101
AABB distance evaluation	0.027
Finding potentially close AABBs	0.045
Finding closest triangles on CPU	0.048
Total hierarchy traversal time	0.120
Total computation time	0.221

TABLE 2
Break-down of the timings for performing different operations of the minimum distance computation algorithm.

Trimmed NURBS Minimum Distance Timings

Table 3 summarizes the results for computing the minimum distance between two trimmed NURBS surfaces, where the bottom surface from Fig. 1(a) is replaced by a trimmed version of the same surface (Fig. 13). The surfaces were timed by positioning them at the same positions as in Table 1 and using the same tolerance of 4×10^{-2} for ACIS. As expected, the GPU timings for finding the minimum distance between trimmed NURBS surfaces are slightly higher than the timings for un-trimmed surfaces. This is because of the extra tests that are performed at each stage to exclude the trimmed regions from the computations. However, the timings are still at least two orders of magnitude faster than ACIS.

Position	ACIS Time (s)	GPU Time (s)	Speed-up
1	55.9	0.234	239x
2	57.7	0.125	461x
3	59.1	0.109	542x
4	58.5	0.187	313x

TABLE 3
Time for performing minimum distance computations between two trimmed NURBS surfaces.

Object	Surfaces	Triangles
Car Body	80	7134
Toy Car	127	17170
Scooby	157	72094
Plane	215	68696
Space Ship	631	37914

TABLE 4
Complexity of the objects used for the clearance computations. The number of triangles shown is the default coarse level of tessellation used for display.

Object Clearance Timings

We performed object clearance computations using the CAD models listed in Table 4; the models are made of trimmed NURBS surfaces and are of approximately the same complexity as standard CAD models used in a mechanical assembly. We used a voxel grid of $40 \times 40 \times 40$ to perform the first-stage

Object1	**Object2**	**ACIS**		**GPU**		**Improvement**	
		Time (s)	**Tolerance**	**Time (s)**	**Tolerance**	**Time**	**Tolerance**
Toy Car	Car Body	9.73	10^{-2}	0.672	2.5×10^{-5}	14x	408x
Car Body	Car Body	17.34	10^{-2}	0.439	2.2×10^{-5}	39x	463x
Scooby	Scooby	70.62	10^{-1}	0.382	13.7×10^{-5}	185x	728x
Plane	Plane	156.03	10^{-1}	1.501	5.2×10^{-5}	104x	1938x
Plane	Space Ship	263.01	10^{-1}	0.794	2.9×10^{-5}	331x	3453x

TABLE 5
Time for performing minimum distance computations between different complex objects.

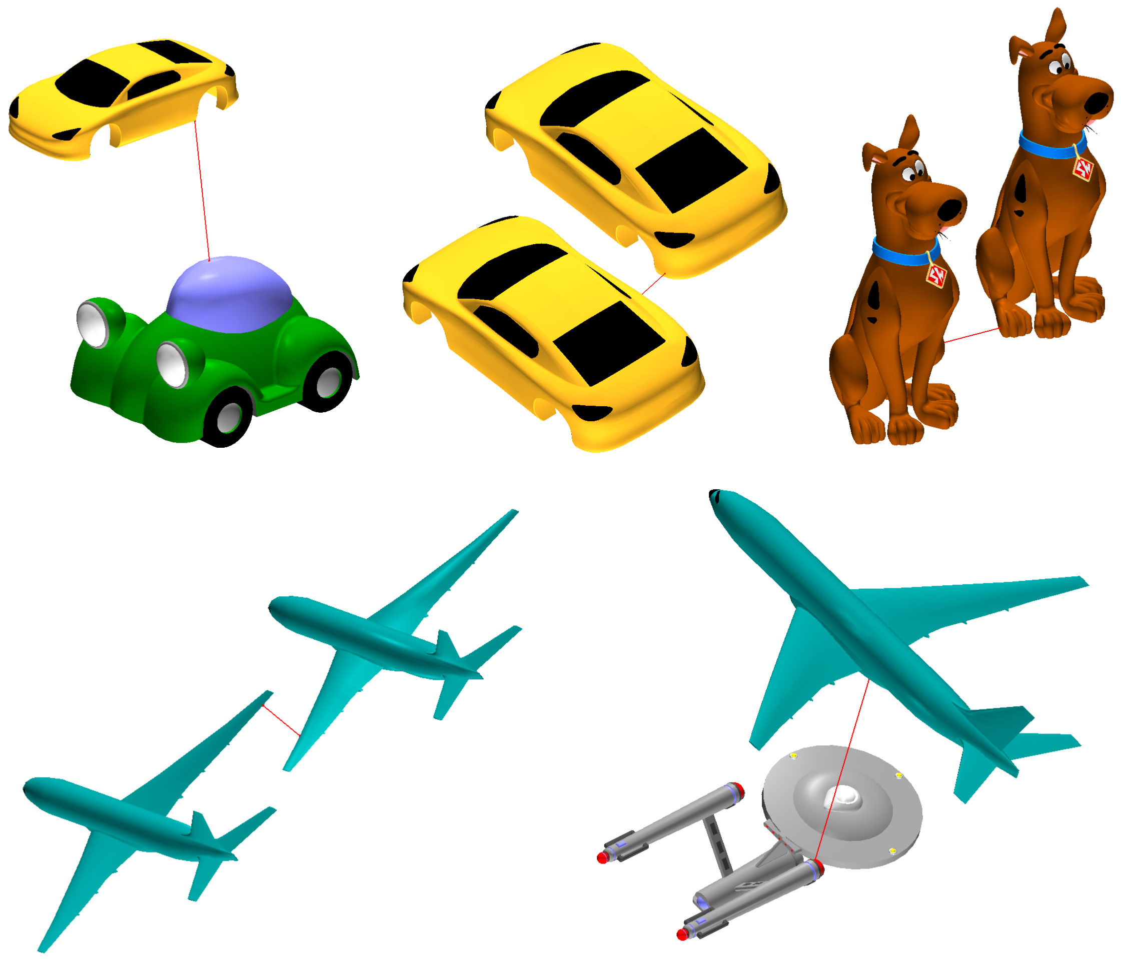


Fig. 17. The different pairs of objects that were timed for the minimum distance computations.

of the minimum distance computations. The objects were also tessellated to a coarse level that is sufficient for display; the number of triangles in this tessellation is given in Table 4.

Minimum distance queries were performed between the object pairs shown in Table 5; the objects were randomly positioned with respect to each other to perform the queries using both ACIS and our GPU accelerated algorithm. We use parametric tolerances that are at least as tight as those specified in the models are. This guarantees that we can accurately calculate the tolerances. We used the api function call *api_check_solid_clearance* to compute the minimum distances in ACIS. It can be seen that the GPU accelerated algorithm is again at least an order of magnitude faster than ACIS.

7 Conclusions

We have developed a hybrid framework that uses GPUs to accelerate distance computations. Our algorithms have theoretical bounds; they have resolutions that are based on object-space instead just image-space. They make use of actual surface data and not just the tessellation, which make them independent of tessellation errors. We can also compute minimum distances between trimmed NURBS surfaces, which make the implementation of our algorithms in an existing CAD system simpler. We also show tremendous performance improvements over existing commercial CPU-based systems.

Our framework can be easily extended to solve other CAD geometric operations such as silhouette extraction, intersection curve evaluation and collision detection. We find that having alternating serial and parallel stages and using the map-reduce motif for parallelism to be ideally suited for developing geometric algorithms that use the GPU. In addition, the parallel stages can be easily modified and executed on a multi-core CPU in the absence of a powerful GPU. Our framework provides for maximum flexibility and optimized performance in developing fast geometric algorithms.

Acknowledgments

We would like to thank NVIDIA and AMD for providing us with their hardware. The models used in the paper were downloaded from 3D Content Central [30]. This material is based upon work supported in part by SolidWorks Corporation, UC Discovery under Grant No. DIG07-10224, and the National Science Foundation under CAREER Award No. 0547675.

Appendix

Detailed Explanation for Theorem 2

We give a detailed explanation for our closest-point bound proved in Theorem 2. From Theorem 2, we know that there are two possible cases. In the first case, the closest point P' lies inside the triangle and the bound can be computed directly to be $\sqrt{4Kd + K^2}$. However, in the second case, to find the maximum possible value of $P'Q$, we have to consider all possible orientations of the triangle with respect to OP'. Let α denote the angle the triangle makes with OP'; α can vary from $90°$ to $180°$ (the two extremes and a general case are shown in Fig. 18). Angle α cannot be less than $90°$ because then P' will no longer be the closest point on the triangle. The angle subtended by $P'Q$ at the center of the sphere, denoted by θ, monotonically decreases from θ_{max} to θ_{min}, as α increases from $90°$ to $180°$. The values of θ_{max} and θ_{min} can be computed to be $\sin^{-1}\left(\frac{\sqrt{4dK}}{d+K}\right)$ and $\sin^{-1}\left(\frac{K}{d+K}\right)$ from Fig 18(a) and Fig 18(b) respectively.

Consider the general case when $90 < \alpha < 180$. $P'Q$ can be computed to be $\sqrt{(d+K)^2 + d^2 - 2d(d+K)\cos\theta}$ from the cosine rule on triangle $OP'Q$. $P'Q$ will be maximized when the term $2d(d+K)\cos\theta$ is minimized, since all the other terms in the expression are positive. $2d(d+K)\cos\theta$ is minimized when θ is the maximum possible value in the range $[\theta_{min}, \theta_{max}]$. Thus $P'Q$ is maximized when $\theta = \theta_{max}$; the extreme case is shown in Fig 18(a) with maximum value of $P'Q$ again being $\sqrt{4Kd + K^2}$ as shown in Fig 9(a).

References

[1] Spatial Corporation, *ACIS Geometric Modeler: User Guide*, 2009, version 20.0.

[2] A. Krishnamurthy, S. McMains, and K. Haller, "Accelerating geometric queries using the GPU," in *SIAM/ACM Joint Conference on Geometric and Physical Modeling*. ACM, 2009, pp. 199–210.

[3] T. G. Mattson, B. A. Sanders, and B. L. Massingill, *Patterns for Parallel Programming*. Addison-Wesley, 2004.

[4] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens, "Scan primitives for GPU computing," in *Symposium on Graphics Hardware*, ACM. Eurographics Association, 2007, pp. 97–106.

[5] G. E. Blelloch, Ed., *Vector Models for Data-Parallel Computing*. MIT Press, 1990.

[6] A. Krishnamurthy, R. Khardekar, S. McMains, K. Haller, and G. Elber, "Performing efficient NURBS modeling operations on the GPU," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 4, pp. 530–543, 2009.

[7] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[8] W. D. Henshaw, "An algorithm for projecting points onto a patched CAD model," *Engineering with Computers*, vol. 18, no. 3, pp. 265–273, 2002.

[9] D. Johnson and E. Cohen, "A framework for efficient minimum distance computations," *IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3678–3684, 1998.

[10] D. D. Nelson, D. E. Johnson, and E. Cohen, "Haptic rendering of surface-to-surface sculpted model interaction," in *SIGGRAPH '05: ACM SIGGRAPH Courses*. ACM, 2005, Course: Recent advances in haptic rendering and applications.

[11] H. Edelsbrunner, "Computing the extreme distances between two convex polygons," *Journal of Algorithms*, vol. 6, no. 2, pp. 213–224, 1985.

[12] S. Quinlan, "Efficient distance computation between non-convex objects," in *Proceedings of IEEE International Conference on Robotics and Automation*. IEEE, 1994, pp. 3324–3329.

[13] X.-D. Chen, J.-H. Yong, G. Wang, J.-C. Paul, and G. Xu, "Computing the minimum distance between a point and a NURBS curve," *Computer-Aided Design*, vol. 40, no. 10-11, pp. 1051 – 1054, 2008.

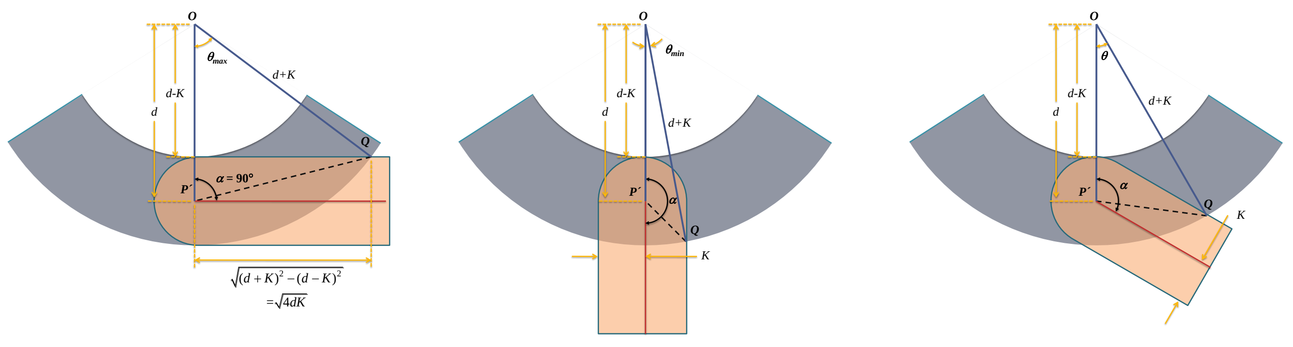
[14] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A hierarchical structure for rapid interference detection," in *ACM SIGGRAPH*. ACM, 1996, pp. 171–180.

[15] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, "Fast distance queries with rectangular swept sphere volumes," *Proceedings of ICRA '00: IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3719–3726, 2000.

[16] N. K. Govindaraju, S. Redon, M. C. Lin, and D. Manocha, "CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware," in *ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. Eurographics Association, 2003, pp. 25–32.

[17] A. Greß, M. Guthe, and R. Klein, "GPU-based collision detection for deformable parameterized surfaces," *Computer Graphics Forum*, vol. 25, no. 3, pp. 497–506, 2006.

[18] A. Sud, M. A. Otaduy, and D. Manocha, "DiFi: Fast 3D distance field computation using graphics hardware," *Computer Graphics Forum*, vol. 23, no. 10, pp. 557–566, 2004.



(a) *Upper Limit Case* **(b)** *Lower Limit Case* **(c)** *General Case*

Fig. 18. The three different cases that can arise when the closest point computed is on the edge of the triangle.

[19] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, “Fast BVH construction on GPUs,” in *Proceedings of Eurographics 2009*. Eurographics Association, 2009.

[20] P. Agarwal, S. Krishnan, N. Mustafa, and S. Venkatasubramanian, “Streaming geometric optimization using graphics hardware,” in *11th European Symposium on Algorithms*, 2003.

[21] K. E. Hoff, A. Zaferakis, M. Lin, and D. Manocha, “Fast and simple 2D geometric proximity queries using graphics hardware,” in *I3D ’01: Proceedings of the 2001 Symposium on Interactive 3D Graphics*. ACM, 2001, pp. 145–148.

[22] S. Briseid, T. Dokken, T. R. Hagen, and J. O. Nygaard, *Computational Science - Lecture Notes in Computer Science*. Springer, 2006, vol. 3994/2006, ch. Spline Surface Intersections Optimized for GPUs, pp. 204–211.

[23] T. Dokken, V. Skytt, T. R. Hagen, and J. O. Nygaard, *US Patent 20080259078 - Apparatus and Method for Determining Intersections*, 2005.

[24] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar, “Rapid and accurate contact determination between spline models using shelltrees,” *Computer Graphics Forum*, vol. 17, no. 3, pp. 315–326, 1998.

[25] C. Lauterbach, Q. Mo, and D. Manocha, “gProximity: Hierarchical GPU-based operations for collision and distance queries,” in *Proceedings of Eurographics 2010*, 2010, p. To Appear.

[26] A. Krishnamurthy, R. Khardekar, and S. McMains, “Direct evaluation of NURBS curves and surfaces on the GPU,” in *ACM Symposium on Solid and Physical Modeling*. ACM, 2007, pp. 329–334.

[27] ——, “Optimized GPU evaluation of arbitrary degree NURBS curves and surfaces,” *Computer Aided Design*, vol. 41, no. 12, pp. 971–980, 2009.

[28] D. Filip, R. Magedson, and R. Markot, “Surface algorithms using bounds on derivatives,” *Computer Aided Geometric Design*, vol. 3, no. 4, pp. 295–311, 1987.

[29] J. Corney and T. Lim, *3D Modeling with ACIS*. Saxe-Coburg, 2001.

[30] 3D Content Central, “http://www.3dcontentcentral.com,” 2009.