Continuous Collision and Interference Detection For 3D Geometric Models[*]

Horea T. Ilieş
Computational Design Laboratory
University of Connecticut[†]
Email: *ilies*@engr.uconn.edu

Abstract

This paper describes a new approach to perform continuous collision and interference detection between a pair of arbitrarily complex objects moving according to general 3-dimensional affine motions. Our approach, which does not require any envelope computations, recasts the problem of detecting collisions and computing the interfering subsets in terms of inherently parallel set membership classification tests of specific curves against the original (static) geometric representations. We show that our approach can compute the subsets of the moving objects that collide and interfere, as well as the times of collision, which has important applications in mechanical design and manufacturing.

Our approach can be implemented for any geometric representation that supports curve-solid intersections, such as implicit and parametric representations. We describe an implementation of the proposed technique for solids given as a boundary representation (B-rep), and illustrate its effectiveness for several rigid and deformable moving objects bounded by tesselated and freeform surfaces of various complexities. Furthermore, we show that our approach can be extended to also identify the local and global self-intersections of the envelopes of the moving objects without requiring to compute these envelopes explicitly. The paper concludes by summarizing the proposed approach as well as reviewing relevant computational improvements that can decrease the computational cost of the prototype implementation by orders of magnitude.

1 Motivation

Detecting collision and interference between moving objects is a ubiquitous problem in a number of critical application domains, such as mechanical design and manufacturing, computer graphics and animation, robotics, virtual reality, computer aided surgery and teleoperation. The existing algorithms for collision and interference detection are fairly tailored to specific application domains, and can be classified into *discrete* and *continuous* approaches. The first set of approaches sample the motions at discrete parameter intervals, perform static interference detection between objects, and require recursive backtracking computations to estimate the actual time of collision. These methods may miss contacts and their computational cost may be hard to predict [32, 10]. On the other hand, continuous approaches do not discretize the motion, but treat the collision detection as a continuous problem. These methods embed the detection of the time of collision directly into the algorithms so no additional recursion is necessary. However, the additional information does not come for free, which is why these methods can often be slower than the discrete approaches.

Many techniques that aim to speed up the collision detection methods have been proposed, including those that use tests on hierarchies of bounding volumes, specialized data structures, or hardware-accelerated computations for tesselated models [3, 9, 8, 22, 16, 10, 10, 30, 15]. It is intuitive that the computational cost associated with detecting collisions and interference depends on both the complexity of the interference test being performed, which in turn depends on the complexity of the geometry and motion, as well as on

[*]ASME Transactions, Journal of Computing and Information Science in Engineering, vol. 9, no. 2, 2009.
[†]Department of Mechanical Engineering, Storrs, CT 06269-3139

how many times the test is actually performed. Therefore, the most efficient algorithms do not only have efficient tests (for example, by maximizing the advantages offered by simplifying geometric assumptions, or by exploiting the temporal and spatial coherence of the problem), but they perform these tests on carefully chosen "simpler" subdomains [16, 15], while taking full advantage of preprocessing computations.

Many efficient collision detection algorithms have been developed for tesselated models widely used in graphics and animation, as discussed above. However, all CAD systems use parametric surfaces as their native geometric representation, and may require the determination of potential collisions and interferences with accuracies that are often higher than what is needed in graphics/animation applications. At the same time, using the existing algorithms developed for tesselated shapes would require an extra step to tessellate the native solid geometry, which involves an additional approximation of the original geometric information. Constructing such approximations can be tricky since it is easy to construct examples in which the actual surfaces collide with each other, while their approximations do not.

1.1 Previous Work for Non-Polyhedra

Detecting collision between polyhedral models is a well established area of research, and there are many fast and robust algorithms that are available [10, 15]. However, there are considerably fewer attempts to compute collisions and interference of non-polyhedral objects. Consequently, even though our approach discussed in Section 2 can handle objects bounded by both freeform and tesselated/polygonal surfaces, we focus here on the existing collision and interference detection research for objects that are *not* bounded by piecewise linear surfaces.

The first class of approaches for detecting collisions of time-varying non-polyhedral objects share one common feature, namely, they use recursive subdivisions to localize the computations of the colliding points [29, 11, 13, 20]. These algorithms typically detect the time of earliest collision, do not seem to detect subsequent (multiple) collisions, and do not perform interference computations. One other class of approaches that are conceptually general (i.e., remain valid for arbitrary objects and motions) rely on sweeps [9, 8]. However, the main difficulty with the approach based on sweeping shapes is that computing the set swept by complex 3-dimensional shapes and motions is not commercially available [7] and is always a computationally expensive task. A recent survey of the current algorithms for computing sweeps can be found in [1].

1.2 Goals and Outline

We propose a continuous collision and interference detection method that is applicable to arbitrarily complex objects moving according to general 3-dimensional affine motions. Our approach, which does not require any envelope computations, recasts the problem of detecting potential collisions and interference in terms of inherently parallel set membership classification tests against the original (static) geometric representations. We show that our approach does not only detect collision and interference (including the time of first or subsequent collisions), but also computes the interfering subsets of the moving objects, which has important applications in mechanical design and manufacturing. We make no restrictions on the class of objects being considered, so we consider 3-dimensional objects of arbitrary complexity, including those objects bounded by freeform surfaces that are moving according to general one parameter affine motions. Our main goals are (1) the rigorous formulation of a new and generic approach to *detect* and *quantify* collisions for the class of problems defined above; (2) to show that our approach does *not* require envelope computations, and that it has attractive computational properties; (3) to illustrate that our approach also produces the interfering subsets of the moving objects when such an intersection exists, as well as the parameters that correspond to the first time of collision (even when multiple collisions occur during the prescribed motion); and (4) to show that our approach can be extended to potentially detect and quantify the singularities of the envelopes of the moving shapes.

We formulate our approach in sections 2.2 and 2.3, and we discuss several extensions and computational issues, as well as our prototype implementation for the remainder of section 2. Section 3 discusses four examples illustrating the proposed capabilities of the proposed method. Finally, section 4 summarizes the contributions of this paper, and discusses a number of important computational issues that are key to an efficient implementation of this approach.

2 Formulation of the Set Membership Classification

2.1 Preliminaries

2.1.1 Motions and trajectories

Following the notation in [14], assume that motion M is a one-parameter family of transformations $M(t)$, and that parameter t belongs to the normalized unit interval. At every instant $t = a$, a set A moves to a new location in space determined by transformation $q = M(a)$ relative to a fixed coordinate system. We represent transformations M(a) as matrices in homogenous coordinates, and use the superscript notation to define the transformed (set of) points as $A^q = A^{M(a)} = M(a)A$.

The transformation $q \in M(t)$ for some instantaneous value $t = a$ determines the position and orientation of A relative to an *absolute* coordinate system. In such a fixed coordinate system, every point x of A will describe a trajectory

$$T_x = \{x^q \mid q \in M\} \tag{1}$$

in the d-dimensional Euclidean space $\mathbb{E}^d$ in which the motion takes place. Clearly, curve T_x contains all points of the space that will be occupied by point x at *some* parameter value during the motion. However, the same motion of A will "look" different as observed from different coordinate systems. An intuitive understanding can be gained with a simple example. Assume that a point moves with constant speed in the positive direction along the x axis starting from the origin. Clearly, an observer placed at the origin and attached to a fixed coordinate system sees the point moving in the $+x$ direction. On the other hand, an observer that is traveling together with the moving point sees the origin moving in the opposite direction (i.e., $-x$ direction). Now consider a point $x \in A$, where A is a moving object, and assume that z is a fixed point in space such that $z = x$, at $t = 0$. As object A moves according to motion M, the fixed point z will appear to be moving relative to A according to the *inverted motion* $\hat{M}$, which is the inverse of $M(t)$ for every value of t [14]. Because of our choice of z, the trajectory of z relative to the moving A is in fact the trace of x as seen from the moving coordinate system, which is denoted by $\hat{T}_x$:

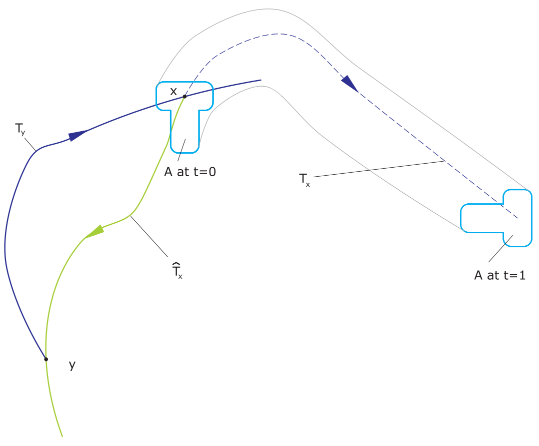


Figure 1: Trajectory of point $y \in \hat{T}_x$ will pass through x at some parameter value during the relative motion.

$$\hat{T}_x = \{x^p \mid p \in \hat{M}\} \tag{2}$$

Now consider two points in the space where the motion takes place: one point x that belongs to set A, and a second point $y \in \hat{T}_x$, as illustrated in Figure 1. Based on equation (2), there exists a parameter value $t = a$ of $\hat{M}(t)$ such that

$$y = \hat{M}(a)x = x^p \tag{3}$$

Since $\hat{M}(a) = M^{-1}(a) \quad \forall a \in [0, 1]$, by multiplying equation (3) to the left by $M(a)$ we obtain

$$M(a)y = x \tag{4}$$

In other words, the trajectory of point $y \in \hat{T}_x$ moving according to the prescribed motion $M(t)$ will pass through point x at some parameter value. In fact, the inverted trajectory $\hat{T}_x$ contains **all** points $y \in \mathbb{E}^d$ that will pass through a given point x when moved according to motion M, or

$$\hat{T}_x = \{y \in \mathbb{E}^d \mid \exists a \in [0, 1] : M(a)y = x\} \tag{5}$$

In the above discussion, motion M can be considered to be a special case of a more general *relative* motion between two coordinate systems that move relative to a fixed coordinate system. This relative motion can be expressed in any coordinate system defined in the same space (see for example [14]).

2.2 Detecting and Quantifying Collisions

Let A and B be two sets of points moving in a d-dimensional Euclidean space. To simplify the notation, denote the motion of A relative to B by M. Thus, $\hat{M}$ will represent the motion of B relative to A. Observe that, if the motion information is given as a sequence of discrete configurations rather than as a continuous function, then one can use a motion interpolation algorithm that will produce a motion that interpolates the given configurations. The choice of a particular interpolation algorithm will clearly influence the resulting continuous motion, and, hence, may influence the results of the collision detection tests. However, such a discussion is outside of the scope of this paper. Two relatively recent reviews of motion interpolation algorithms are presented in [35, 28].

The problem that we are addressing in this paper can be stated as follows. Given two sets of points A and B, and the relative motion M of A relative to B, detect whether collision occurs between A and B, as well as the parameter values of $M(t)$ that correspond to the time(s) of collision between A and B. If the two sets collide, then compute the subsets of A and B that interfere during M.

Consider a point x that belongs to set B. Recall from equation (5) that the inverted trajectory of a point $x \in \mathbb{E}^d$ contains *all* points of the space that will pass, at some parameter value, through x as these points move according to M.

Thus, if $\hat{T}_x \bigcap A = \emptyset, \quad \forall x \in B$ then A and B do not collide during the prescribed relative motion M. By contrast, if

$$\hat{T}_x \bigcap A \neq \emptyset, \quad x \in B \tag{6}$$

the two objects A and B will collide. Furthermore, the set of all points $x \in B$ for which the intersection in equation (6) is non-empty will represent the subset of B that will collide with A during M. Moreover, the corresponding points of intersection in equation (6) will represent the subset of A that interferes with B during the same motion M.

Observe that computing the set of points of interference according to equation (6) involves a set membership classification between the inverted trajectory curves and set A to determine the subsets of the curve that are 'in', 'on', or 'out' of A [34]. Set membership classifications (SMC) rely on point membership classifications (PMC) that classify a point of the space against a given solid X as "in", "on" or "out" of the solid X. PMC is one of the fundamental operations in solid modeling [26, 33], which guarantees that a given solid modeling representation is unambiguous [26], and is implemented in all commercial solid modeling systems. Thus, the subset of B that collides and interferes with A during M can be written as

$$\{x \in B \quad | \quad \hat{T}_x in A \cup \hat{T}_x on A \neq \emptyset\} \subset B \tag{7}$$

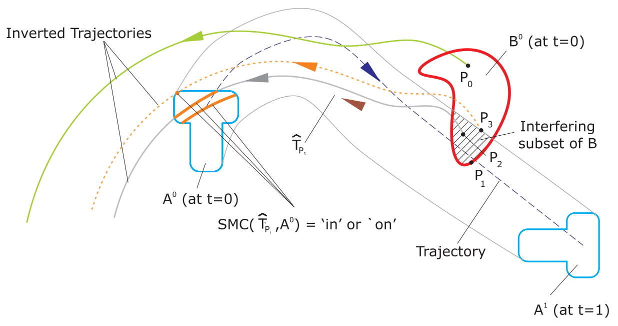


Figure 2: Collision and interference detection based on set membership classifications (SMC) of the inverted trajectory against the original static object representation.

where $\hat{T}_x in A$ and $\hat{T}_x on A$ are the points of the inverted trajectory $\hat{T}_x$ that are classified as 'in' or 'on' A by the SMC. Moreover, the subset of A that collides and interferes with B during M can be written as

$$\{\hat{T}_x in A \cup \hat{T}_x on A \quad | \quad x \in B\} \subset A \tag{8}$$

Note that the set described by equation (8) is a set comprised of boundary points of A, and a set of curve segments that are in the interior of A. Intuitively, the subset of A that interferes with B is made of a bundle of curve segments belonging to the inverted trajectories..

Because they are formulated in terms of set theoretic properties, equations (6), (7) and (8) remain valid for any sets of points, and allow us to compute to a user-specified precision not only whether the collision occurs, but also the interfering subsets of A and B. The actual implementation of the SMC test will depend on the representations used for the inverted trajectory and for the geometry of set A. For example, standard algorithms for solids given as a CSG or boundary representation are illustrated in [27].

2.3 Reducing the Density of the Point Sampling

Our approach supports the collision detection as well as the computations of the interfering subsets of A and B to any desired level of accuracy that is limited, in principle, only by the machine precision. The density of the point sampling increases with the required quality of the approximation, but an extensive sampling of points (both boundary and interior points) is often impractical, that is, unless the computed interference subsets need to be highly accurate.

Reducing the number of points for which inverted trajectories are computed can be achieved by taking advantage of a number of available techniques such as hierarchical bounding boxes or a variety of cellular decompositions. In addition, when A and B are solid shapes, one of the most obvious ways to reduce the number of points of B that are being tested is to only sample the boundary of B rather than points that are either "on" or "in" B as suggested by equation (6). Such a sampling dramatically reduces the number of points for which we compute the inverted trajectories, and therefore the number of intersections that need to be performed.

On the other hand, considering only points that are on B will no longer provide the interior points of B that will interfere with A. However, these boundary points of B can be used to construct an approximation of the interfering subset of B as discussed below. At the same time, instead of using the portions of the inverted trajectories that are either "in" or "on" (according to equation (8)) one can only store the points of $\hat{T}_x$ that are "on" A, and use these points to construct a similar approximation of the interfering subset of A. In this case, if B is a $d-$dimensional solid object moving in a $d-$dimensional Euclidean space, equation (6) becomes:

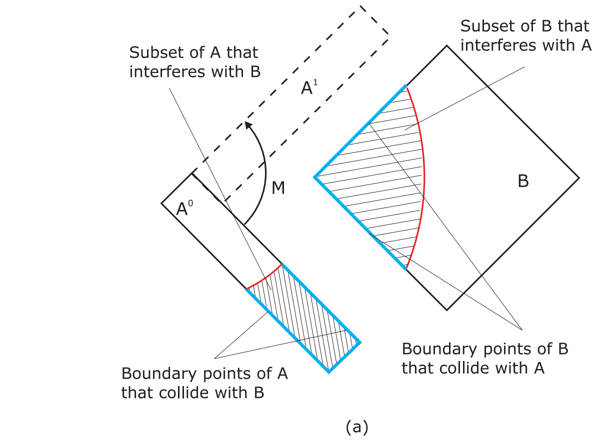
$$\hat{T}_x \bigcap \partial A \neq \emptyset, \quad x \in \partial B \tag{9}$$

and the colliding boundary points of B and A can be written as

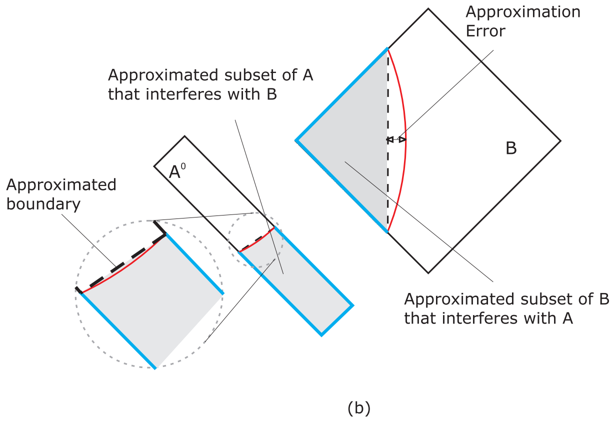
$$\{x \in \partial B \quad | \quad \hat{T}_x on A \neq \emptyset\} \subset \partial B \tag{10}$$

and

$$\{\hat{T}_x on A \quad | \quad x \in \partial B\} \subset \partial A \tag{11}$$



Subset of A that interferes with B
Subset of B that interferes with A
A¹
M
A⁰
B
Boundary points of A that collide with B
Boundary points of B that collide with A
(a)



Approximation Error
Approximated subset of A that interferes with B
B
A⁰
Approximated boundary
Approximated subset of B that interferes with A
(b)

Figure 3: Sampling only boundary points of B does not provide the interior points of B that will interfere with A. Using only these boundary points to estimate the interfering subsets can lead to either an underestimated (in case of B) or an overestimated (in case of A) set of points of interference as shown in (b).

Consider a rectangular set A that is rotating about one of its corners by 90^o in a counter clockwise direction as shown in Figure 3. During this rotation, set A interferes with the square set B, and the interference subsets of A and B are shown in Figure 3(a). Let's now assume that we want to reduce the sampling size and the computational complexity, and that we choose to do so by only using the colliding boundary points of A and B according to equations (10) and (11). There are many ways to construct approximations of these interfering sets from the set of computed boundary points, but a straight forward approach would be to triangulate these boundary points as suggested in Figure 3(b)) by using one of the standard algorithms for surface reconstruction from point clouds, such as constrained Delaunay triangulations [31], or a variety of other algorithms to transform point clouds into surfaces [4, 21, 38] (see also section 3 for several examples). Note that the error in the approximation will depend on the actual motion and geometric features, but these approximations can be improved by adding sampling points in the interior of B that are in the neighborhood of the newly constructed boundary (see Figure 3(b)), and by using some additional points of the trajectory segments that are "in" A.

The relationship between M and T on one hand and $\hat{M}$ and $\hat{T}$ on the other hand implies that the same approach to collision and interference detection can be performed in terms of motion M and trajectory T. The choice of the coordinate system used to represent the relative motion between A and B will depend on how complex the geometries of A and B are (for example measured in terms of differential properties of the boundary as well as the number and individual area of faces). Since sampling points on the boundary is much faster than performing curve-face intersection, one would sample points on the boundary of the more "complex" object and perform the intersections with the boundary of the "simpler" object.

2.4 Extensions

Computing the envelope singularities

During motion M, each moving set will sweep a set of points that is bounded by the envelopes to the family of shapes generated by the moving objects [2]. The fact that the inverted trajectory of a point x contains all points of space that will pass through x at some time during the prescribed motion M can be used to develop a point membership classification test that can classify any point of the space as being an "in", "on", or "out" point relative to the set swept by a moving object. The details of this PMC test are given in [6], where we argue that this PMC test can be used either as a stand alone boundary evaluator for sweep or in conjunction with other boundary evaluators for detecting the candidate faces that belong to the boundary of the sweep.

Briefly, in [7] we show that performing the PMC test for sweeping solids of arbitrary complexity moving according to affine motions relies on intersections between the inverted trajectories of points in space and the moving object in its original *static* configuration followed by a careful postprocessing of the intersection results. This procedure allows the classification of any point of the space as being in the *interior*, on the *boundary* or in the *complement* of the set swept by the moving solid, and the points interior to the sweep are classified further as either *regular*, *fold* or *fold boundary* points. Furthermore, we show in [5] that the existence of fold and fold boundary points is a necessary but not sufficient condition for the existence of singularities in the envelopes of the moving object. We also argue there that the inverted trajectory intersection test determines the subsets of the boundary points of the moving object that pass through a given fold region, which can be used to develop the sufficient conditions for the existence of envelope singularities. At the same time, the detected fold regions can be used to interactively assess the existence of envelope singularities without requiring the computation of the envelopes that bound the swept set.

Combinatorial Collision Detection

In practice it is often required to compute the collision and interference between multiple objects in motion. The simplest such case is when all objects *except* one move rigidly relative to each other, for example when assembling a new component into an assembly. Conceptually, such a case is equivalent with the case of two objects moving relative to each other, if one considers the union of the components in the assembly as one of the moving objects. If the relative motion between the new component and the assembly is known, one can use standard methods to reduce the set of sampled points as well as the set of faces that need to be intersected as discussed above. On the other hand, the most complex situation is when all objects move relative to each other, which is a problem that has been studied in the collision detection literature. The usual pairwise collision detection as well as specialized data structures and algorithms to reduce the number of objects that need to be considered can still be applied to our approach. At the same time, one practical approach may be in these cases to tessellate the objects, apply one of the fast approximate collision detection algorithms to eliminate the most obvious non-colliding objects which would be followed by an accurate computation of the collision and interference between the remaining objects based on the formalism developed in this paper.

3 Examples

Based on the above discussion, one we can construct an algorithm to detect and quantify the collisions between two moving sets in a $2-$ or $3-$dimensional Euclidean space. Our approach can be implemented for any geometric representation that support curve-solid or curve-surface intersections. Since our approach is formulated in terms of set theoretic properties, it remains valid regardless of the regularity or dimensionality of the moving sets. In our prototype implementation, we input the boundary representations of two moving objects. The output consists of the interfering subsets as point clouds, and of all the parameter values of all the intersection points, including the time of first collision. We first compute a point sampling of the boundary representation of one of the moving objects. For each such point, we construct the inverted trajectory as a piecewise polynomial curve, intersect it with the boundary of the second object, and we collect the intersection information when such an intersection occurs.

The colliding boundary regions can be computed to any desired level of accuracy depending on the sampling strategy employed according to a user-specified precision. Given a motion M, one can generate exact[1] or approximate trajectories as well as inverted trajectories for any point in the space. The curve-surface intersection between $\hat{T}$ and the boundary surface will depend on the representations used for both $\hat{T}$ and geometry. For example, if set A is given as a boundary representation, then the curve-surface intersection algorithm reduces to curve-face intersection between the inverted trajectory and the corresponding faces of the solid. This is a standard operation in CAD that is efficiently implemented in all commercial CAD systems. It is important to note that the curve-surface intersection is a standard geometric modeling operation, and therefore there are many other algorithms available for performing this intersection (see for example [19]).

In this section we illustrate how our approach can be applied to perform continuous collision and interference detection of solid shapes moving according to both rigid and non-rigid spatial motions. In our examples, the shapes are bounded by both planar and non-planar NURBS surface patches. We implemented the collision and interference detection algorithm outlined above in Parasolid, using Microsoft Visual Studio 2005.

In all three examples the motion information was assumed to be given as a sequence of several discrete configurations represented as 4×4 matrices in homogenous coordinates. During the preprocessing stage, we interpolated the prescribed configurations by using the algorithm outlined in [12] to obtain a continuous description of the relative motion as well as of its inverted counterpart (see section 2.1). In all four examples discussed here we used the inverted relative motion $\hat{M}$ to perform the collision and interference detection. As argued in section 2, one could formulate the collision and interference tests in terms of the relative motion M, depending on the relative complexity of the boundaries of the moving objects. The inverted trajectories were constructed as interpolating cubic B-spline curves by using points sampled along each inverted trajectory according to the inverted motion. However, it is known that, if the motion interpolation algorithm generates a rational B-spline motion, then the trajectories of points moving according to the interpolated motion are rational B-spline curves (see for example [18]). This, in turn, implies that one can, in principle, construct the trajectories directly from the motion representation rather than by interpolating sampled points, which would result in more efficient computations.

Figure 4(a) shows two cubes moving relative to each other as illustrated by the sampled intermediate configurations of the inverted relative motion. The motion is a rigid body motion that contains a translation along a prescribed curve, and two simultaneous rotations around x and z axes, respectively. Several representative inverted trajectories are shown in Figure 4(a.2). On each face of cube B we sampled 10k points whose inverted trajectories were intersected with all faces of cube A. For any such curve that intersected a face of A we stored the original point on B and the intersecting point(s) on the corresponding face of A. To illustrate the subsets of the boundaries of A and B that collide, we displayed two small spheres for each pair of colliding boundary points of A and B, as shown in Figure 4(a.3). The same colliding boundary points have been used to compute the Delaunay triangulations approximating the interior points of A and B that are also involved in the collision. Here we decomposed the original boundary points into convex subsets and used the Qhull algorithm [23] to compute the two Delaunay triangulations as shown in Figure 4(a.4). However, one could use other approaches to construct such approximations, for example by using

[1]Rational motions discussed, for example, in [18, 17, 36, 37] have the property that the trajectory of any moving point is a NURBS curve, which can be obtained, in principle, directly from the motion formulation.

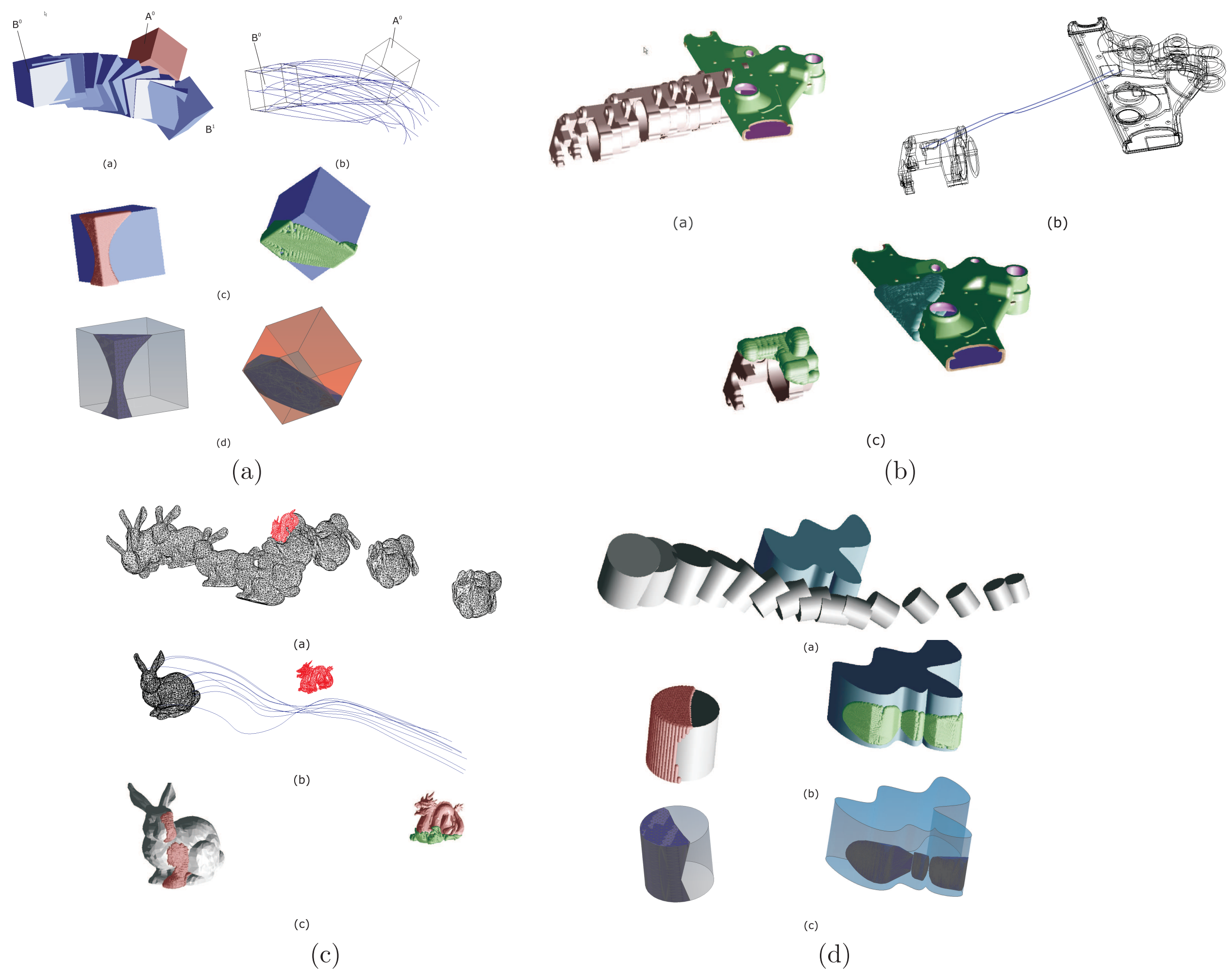


Figure 4: Continuous collision and interference detection between rigid and deformable shapes.

constrained Delaunay triangulations [31], or a variety of other algorithms to transform point clouds into surfaces [4, 21, 38].

A second example shows two objects from the NIST design repository [24] having a more complex geometry, and containing 850 and 240 faces, respectively. The faces of each object in Figure 4(b) are represented as NURBS surface patches. For simplicity, we used a grid bases sampling of points on each face of object A, although one could employ an adaptive sampling method based on some metric related to the properties of the face. For example, the number of points sampled in each face could depend on some measure of size (such as the surface area), and on the complexity of the face, which would be computed in the preprocessing stage. In this example, we sampled points in the parameter space of each face of A along a rectangular grid, which was followed by a Point Membership Classification of the sampled point against that face. Those points that were not "in" or "on" the face were discarded. For each of the remaining points we computed the inverted trajectories (as described above), and intersected each curve with the boundary of A. The colliding points of the boundaries ∂A and ∂B of A and B are displayed as spheres.

Our third example shows two tesselated surface models obtained from the Stanford 3D scanning repository [25] for which we constructed a standard boundary representation. Our model of the Stanford bunny contains over 2900 triangular faces. Since the bunny surface is not closed, it does not bound a solid set of points. On the other hand, the dragon surface, which contains about 2700 faces, is closed and regular and bounds a solid set of points. The relative motion between the two models was given as a discrete sequence of configurations composed of a translation along a prescribed space curve and two simultaneous rotations around the x and z axes. Figure 4(c.1) shows several intermediate configurations of the bunny as it moves relative to the dragon according to the inverted motion, while a few typical inverted trajectories are shown in Figure 4(c.2). In this example, we used the same point sampling strategy as in the previous two examples. The results of our tests are shown in Figure 4(c.3), where each colliding point is, once more, visually identified by a sphere.

Our last example shows a cylindrical object that is deforming as it moves according to a 3-dimensional affine motion. The motion in this example was interpolated from discrete configurations that were obtained by combining a translation along a prescribed curve, two rotations around x and z axes with angles varying between $0 - 80^o$ as well as a scaling transformation. The two Delaunay triangulations approximating the interfering subsets of the deformable and rigid object shown in Figure 4(d) were computed from the colliding boundary points by using the Qhull algorithm [23].

4 Conclusions

Collision and interference detection is an important problem that arises in a variety of applications domains. Since each such domain imposes different requirements on the procedures used to detect and quantify collisions, the existing algorithms are fairly tailored to specific applications. From a practical point of view, all approaches end up reaching some tradeoff between computational efficiency and accuracy to meet these requirements.

This paper introduces a new method to compute the collision and interference between geometric models to a user-specified accuracy that is essentially limited only by the machine precision. Our approach is based on set membership classification tests between the trajectories of points described by the relative motion of the objects, and their original geometric representation. We have shown that the method is applicable to any geometric representation that supports set membership classifications between a curve and a set of points or, alternatively, curve-surface intersections. Specifically, we implemented our approach within a commercial geometric kernel, and discussed several examples containing both rigid and non-rigid objects bounded by either NURBS or tesselated (triangulated) surfaces.

As is the case with any other algorithm for collision and interference detection, the computational cost associated with our approach depends on both the efficiency of the specific intersection test, as well as on how many times this test is actually performed. While the most efficient curve-surface intersections will necessarily take advantage of specific representations, one can dramatically reduce the number of times the intersection is actually carried out. To this end, many of the existing techniques designed to speed up the computations, such as hierarchies of bounding volumes, and specialized data structures published in the literature could be employed by our approach. At the same time, if the moving shapes are (possibly deformable) d-dimensional solids moving in a d-dimensional space, then one can significantly reduce the number of intersection tests by only sampling boundary points based on what subsets of the boundary will potentially collide, as discussed in section 2.2. For example if the objects are given in a boundary representation, one would only sample points in those faces that are involved in collision.

The main contribution of this paper is in the formulation of a new approach to perform continuous collision detection as well as to determine the interfering subsets for both rigid and non-rigid geometric models of arbitrary complexity moving according to general one parameter affine motions. We do not require the motion to be known in closed form, but we assume that the motion information is given as discrete configurations[2]. To summarize, our approach:

- can be applied to *any* geometric representation scheme that supports set membership classification between a curve (trajectory or inverted trajectory) and a set of points;
- can detect the collision and the amount of interference *for both moving sets*, as well as the time of first or subsequent collisions *without* requiring any envelope computations;

[2]Observe that many applications requiring interactive collision detection have only "local" information about the motion of each object in the environment, usually obtained by integrating the equations of motion. In principle, the trajectories of the moving points can be obtained through this integration.

- can take advantage of any reduced complexity in the boundaries of the moving sets to decrease the number of intersection tests being performed, and therefore to speed up the computations by expressing the relative motion between two sets of points in different coordinate systems. At the same time, our approach can fully benefit from the existing techniques aimed at reducing the number of tests performed such as those employing hierarchical bounding boxes;
- can be used to potentially detect and quantify the singularities of the envelopes of the moving shapes, for example to identify the undercutting/overcutting in NC machining applications, as discussed in section 2.4. Moreover, it is worth noting that the same computations can be used to generate points that are on the boundary or in the interior of the set $\texttt{sweep}(A, M)$, and that our computations do not require envelope computations;
- is inherently parallel due to the fact that our tests are defined pointwise, which implies that parallel algorithms can potentially achieve impressive speedups.

In order to preserve the generality of our approach, we avoided making restrictive assumptions on the geometries and motions that have been considered in this paper. In turn, this implies that a number of specific issues have not been discussed here, and have to be addressed separately. The most important challenge of this approach is clearly the efficiency of the computations. While the challenge that we face is common to all exiting algorithms to detect collision and interference, we believe that the generality of our approach makes it attractive to a variety of applications requiring highly accurate collision and interference detection algorithms.

Acknowledgments

This work was supported in part by a grant from the University of Connecticut Research Foundation and by the National Science Foundation grant CMMI-0555937, and CAREER award CMMI-0644769. All 3D examples were created by using Parasolid and Unigraphics, courtesy of Siemens PLM.

References

[1] K. Abdel-Malek, D. Blackmore, and K. Joy. Swept volumes: Foundations, perspectives, and applications. *International Journal of Shape Modeling*, 12(1):87–127, 2006.

[2] James W. Bruce and Peter J. Giblin. *Curves and Singularities*. Cambridge University Press, 1992.

[3] J. Canny. Collision detection for moving polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8*, 8:200–209, 1986.

[4] H. Edelsbrunner and E.P. Mücke. Three-dimensional alpha shapes. *ACM Transactions of Graphics*, 13(1):43–72, 1994.

[5] H. Erdim and H.T. Ilieş. Detecting and quantifying envelope singularities: the 2-dimensional case. Technical report, Computational Design Laboratory, University of Connecticut, 2006.

[6] H. Erdim and H.T. Ilieş. Point membership classification for solid sweeping. In *ASME IDETC 2007, Design Automation Conference*, Las Vegas, NV, USA, 2007.

[7] H. Erdim and H.T. Ilieş. Classifying points for sweeping solids. *Computer-Aided Design*, 40(9):987–998, 2008.

[8] M. A. Ganter and J. J. Jr. Uicker. Dynamic collision detection using swept solids. In *American Society of Mechanical Engineers Design Engineering Technical Conference*, 1986.

[9] M.A. Ganter and B.P. Isarankura. Dynamic collision detection using space partitioning. *Journal of Mechanical Design*, 115(1):150–155, March 1993.

[10] Sunil Hadap, Dave Eberle, Pascal Volino, Ming C. Lin, Stephane Redon, and Christer Ericson. Collision detection and proximity queries. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, page 15, New York, NY, USA, 2004. ACM Press.

[11] Brian Von Herzen, Alan H. Barr, and Harold R. Zatz. Geometric collisions for time-dependent parametric surfaces. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 39–48, New York, NY, USA, 1990. ACM Press.

[12] T. Horsch and B. Jüttler. Cartesian spline interpolation for industrial robots. *Computer Aided Design*, 30(3):217–224, 1998.

[13] M. Hughes, C. DiMattia, M. C. Lin, and D. Manocha. Efficient and accurate interference detection for polynomial deformation. In *CA '96: Proceedings of the Computer Animation*, page 155, Washington, DC, USA, 1996. IEEE Computer Society.

[14] H.T. Ilieş and V. Shapiro. The dual of sweep. *Computer Aided Design*, 31(3):185–201, March 1999.

[15] Doug L. James and Dinesh K. Pai. BD-Tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3), August 2004.

[16] P. Jimnez, F. Thomas, and C. Torras. 3D Collision Detection: A Survey. *Computers and Graphics*, 25(2):269–285, April 2001.

[17] B. Jüttler. Spatial rational motions and their applications in computer aided geometric design. In M. Dæhlen, T. Lyche, and L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, pages 271–280. Vanderbilt University Press, Nashville, TN, 1995.

[18] B. Jüttler and M.G. Wagner. Computer-aided design with spatial rational B-spline motions. *Journal of Mechanical Design*, 118:193 – 201, June 1996.

[19] D. Manocha and S. Krishnan. Algebraic pruning: a fast technique for curve and surface intersection. *Computer Aided Geometric Design*, 14(9):823–845, 1997.

[20] F. Page and F. Guibault. Collision detection algorithm for nurbs surfaces in interactive applications. In *IEEE CCECE 2003, Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 1417– 1420, May 2003.

[21] M. Peternell. Developable surface fitting to point clouds. *Computer Aided Geometric Design*, 21(8):785–803, 2004.

[22] M. K. Ponamgi, D. Manocha, and M. C. Lin. Incremental algorithms for collision detection between general solid models. Technical Report TR94-061, Department of Computer Science, University of North Carolina - Chapel Hill, November 18 1994.

[23] Qhull. The geometry center. http://www.qhull.org.

[24] The NIST Design Repository. http://www.designrepository.org/.

[25] The Stanford 3D Scanning Repository. http://graphics.stanford.edu/data/3Dscanrep/.

[26] A.A.G. Requicha. Representations for rigid solids: Theory, methods and systems. *Computing Surveys*, 12(4):437–463, 1980.

[27] Ari Requicha. Geometric modeling: a first course. available on-line.

[28] O. Röschel. Rational motion design - a survey. *Computer Aided Design*, 30(3):169–178, 1998.

[29] Elmar Schömer, Joachim Reichel, and Thomas Warken. Efficient collision detection for curved solid objects. In *SMA '02: Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 321–328, New York, NY, USA, 2002. ACM Press.

[30] J.K. Seong, G. Elber, and M.S. Kim. Trimming local and global self-intersections in offset curves/surfaces using distance maps. *Computer Aided Design*, 38(3):183–193, March 2006.

[31] Hang Si. Tetgen, a quality tetrahedral mesh generator and three-dimensional delaunay triangulator, v1.3 user's manual. Technical Report 9, Weierstrass Institute for Applied Analysis and Stochastics, 2004. http://tetgen.berlios.de/.

[32] John M. Snyder. Interval analysis for computer graphics. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 121–130, New York, NY, USA, 1992. ACM Press.

[33] R. B. Tilove. Set membership classification: A unified approach to geometric intersection problems. *IEEE Transactions on Computer*, C-29(10):874–883, October 1980.

[34] Robert B. Tilove. A study of geometric set membership classification. Master's thesis, University of Rohester, 1977.

[35] M. Žefran and V. Kumar. Interpolation schemes on $SE(3)$. *Computer Aided Design*, 1997. to appear.

[36] M. Wagner. Planar rational b-spline motions. *Computer Aided Design*, 27(2):129–137, 1995.

[37] M. Wagner and B. Ravani. Computer aided design of robot trajectories using rational motions. In L. Lenarčič and V. Parenti-Castelli, editors, *Recent Advances in Robot Kinematics*, pages 151–158. Kluwer Academic Publishers, 1996.

[38] J. Wang, M. M. Oliveira, and A. E. Kaufman. Reconstructing manifold and non-manifold surfaces from point clouds. In *IEEE Visualization*, Minnapolis, MN, October 2005.