PERGAMON

Computers & Graphics 25 (2001) 269–285

COMPUTERS &GRAPHICS

www.elsevier.com/locate/cag

Technical Section

# 3D collision detection: a survey

P. Jiménez*, F. Thomas, C. Torras

*Institut de Robòtica i Informàtica Industrial (CSIC-UPC), Gran Capità 2-4 (Ed. Nexus), 08034-Barcelona, Spain*

**Abstract**

Many applications in Computer Graphics require fast and robust 3D collision detection algorithms. These algorithms can be grouped into four approaches: space–time volume intersection, swept volume interference, multiple interference detection and trajectory parameterization. While some approaches are linked to a particular object representation scheme (e.g., space–time volume intersection is particularly suited to a CSG representation), others do not. The multiple interference detection approach has been the most widely used under a variety of sampling strategies, reducing the collision detection problem to multiple calls to static interference tests. In most cases, these tests boil down to detecting intersections between simple geometric entities, such as spheres, boxes aligned with the coordinate axes, or polygons and segments. The computational cost of a collision detection algorithm depends not only on the complexity of the basic interference test used, but also on the number of times this test is applied. Therefore, it is crucial to apply this test only at those instants and places where a collision can truly occur. Several strategies have been developed to this end: (1) to find a lower time bound for the first collision, (2) to reduce the pairs of primitives within objects susceptible of interfering, and (3) to cut down the number of object pairs to be considered for interference. These strategies rely on distance computation algorithms, hierarchical object representations, orientation-based pruning criteria, and space partitioning schemes. This paper tries to provide a comprehensive survey of all these techniques from a unified viewpoint, so that well-known algorithms are presented as particular instances of general approaches. 

*Keywords:* Geometric algorithms; Languages and systems; Collision detection; Interference tests

## 1. Introduction

Many collision detection algorithms have been proposed in recent years within the fields of Computational Geometry, Robotics, and especially Computer Graphics. Some appear tailored to particular applications, others stem from theoretical concerns, and their diverse origins and aims often hide the common ground on which they lie. This article tries to unravel this common ground.

A recent survey [1] on the subject classifies collision detection algorithms according to the geometric object model used. The present paper is rather oriented towards a systematic characterization of the solving strategies, as explained below. The reader interested in a quick comparison of the most well-known algorithms can have a look at the table in http://brl.ee.washington.edu/BRL/shc/collide.htm.

Computer Graphics encompasses a broad set of applications related to Computer-Aided Design, Virtual Reality and Physical Simulation that require fast collision detection. The algorithms developed in this context are above all intended to be of practical use. Contrarily, the field of Computational Geometry seeks to synthesize algorithms with the best possible worst-case complexity, which in many cases entails the design of intricate data structures. Thus, while computational geometers and graphicists have a substantial overlap of interest in geometry, their algorithms obey markedly different purposes.

Computational geometry algorithms often assume "general position" but real-world models tend to have

* Corresponding author.

*E-mail addresses:* jimenez@iri.upc.es (P. Jiménez), thomas@iri.upc.es (F. Thomas), torras@iri.upc.es (C. Torras).

0097-8493/01/$ - see front matter 
PII: S0097-8493(00)00130-8

a lot of degeneracies such as coplanar or parallel faces. Thus, any assumption of "general position" is inappropriate in practical settings. Not only degeneracies, but also "bad data", lead to challenging problems. For example, as noted in [2], polyhedral models generated by some widely used CAD systems tend to have various degrees of nearly coplanar vertices, i.e., polygonal faces bounded by four or more vertices where the vertices only approximately lie on the same plane. And even worse, many models tend to have self-intersections, i.e., they contain faces which intersect at places other than their boundaries. As a consequence, practical collision detection algorithms are shaped not only by the application itself, but also by the challenging inputs arising in practice.

The application largely delimits the kind of algorithms to be applied. For example, while the path taken by the moving object in Virtual Reality applications is not known a priori, precisely specified trajectories have to be checked for collision in rigid-body physical simulation systems that involve the exact reproduction of mechanical processes. Thus, a trajectory-based approach, suitable in the latter case, would be useless in the former one. Moreover, when the application allows it and for efficiency's sake, a collision detection algorithm might deliberately introduce error. For example, objects might be crudely approximated by cubes, spheres, polyspheres, etc., and complex trajectories decomposed into simpler ones or simply discretized. Practical collision detection algorithms have long sought to exploit this tradeoff between solution quality and computational time. We devote an important part of this survey to these approximations.

The paper is structured as follows. Section 2 shows that the available collision detection approaches lie within two main categories: geometric and algebraic, and explains how those based on the former category apply two techniques: projection and sampling, or combinations of them. This overview makes clear that tests for static interference between simple geometric entities lie at the base of most detection approaches. Section 3 presents these tests. The efficiency of a basic interference test does not guarantee that a collision detection algorithm based on it is in turn efficient, because the number of times the test is applied is another key factor. Thus, Section 4 reviews the different strategies to restrict the application of the interference test to those instants (time bounds) and object parts (space bounds) at which a collision can truly occur. Finally, conclusions are sketched in Section 5.

## 2. Approaches to collision detection

Collision detection admits several problem formulations, depending on the type of output sought and on the constraints imposed on the inputs. The simplest decisional problem, that looking for a yes/no answer, is usually stated as follows: Given a set of objects and a description of their motions over a certain time span, determine whether any pair will come into contact. More intricate versions require finding the time and features involved in the collision. Placing constraints on the inputs is a usual way of simplifying problems. Thus, often objects are assumed to be polyhedra, usually convex ones, and motions are constrained to be translational or linear in a given parameter space.

In the following subsections, the four main approaches that have been proposed to deal with the different instances of the collision detection problem are described.

### 2.1. Spatio-temporal intersection

The most general representation of the collision detection problem is based on the extrusion operation [3]. The *extruded volume* of an object is the spatio-temporal set of points representing the spatial occupancy of the object along its trajectory. A collision between two objects occurs if, and only if, their extruded volumes intersect (see Figs. 1 and 2).[1]

The extrusion operation is distributive with respect to the union, intersection and set difference operations. This motivated the development of the extrusion approach in the context of constructive solid geometry (CSG) representations. The mentioned distributive property guarantees that an object and its extruded volume can be represented through the same boolean combination of volumetric primitives and extrusions of these primitives, respectively.

The formal beauty of this approach is partially occluded by the high cost of its practical implementation, whose bottleneck is the generation of the 4D extruded volumes themselves. Thus, for example, the extrusion of a linear subspace subject to a constant angular velocity is bounded by a helicoidal hypersurface. For this reason, the implementation deals only with linear subspaces subject to piecewise translational motions [3].

If the computation of the extruded volumes in 4D were simple, no other approaches would have been introduced, since they are all aimed at avoiding this explicit computation. These approaches are either geometric or algebraic. Among the geometric ones, two main alternatives have been proposed, namely projecting the extruded volume onto a lower-dimensional subspace — which leads to the swept volume approach — and sampling along the trajectory, which boils down to the repeated application of an intersection detection algorithm. The

[1] Almost all figures that illustrate the different issues in the text represent the corresponding 2D situation for clarity.

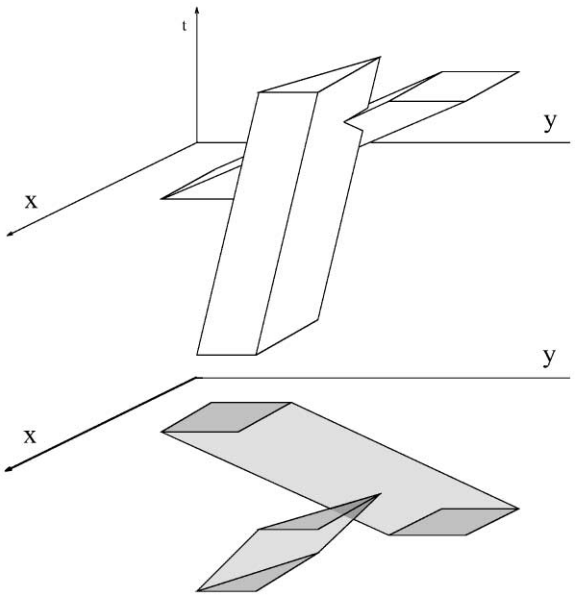


Fig. 1. Extruded and swept volumes. Time is explicitly taken into account. Since the extruded volumes interfere, a collision is detected. Below, the corresponding swept volumes in 2D are shown.

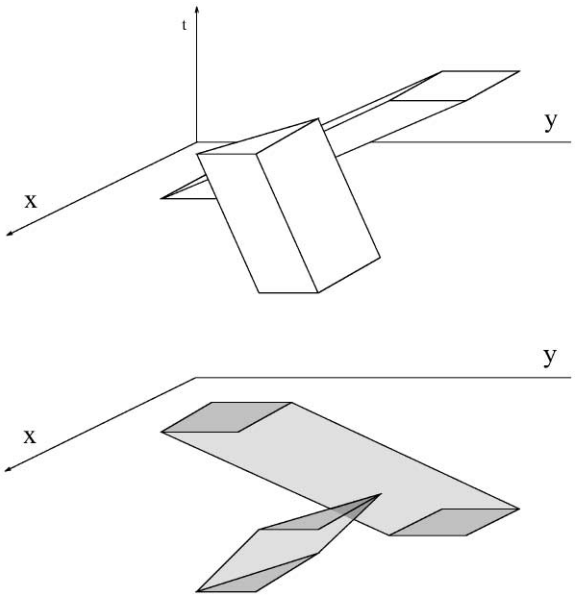


Fig. 2. Extruded and swept volumes (cont.). Here no collision takes place, and therefore the extruded volumes do not interfere. Note that the corresponding swept volumes in 2D are the same as those in the previous figure.

algebraic approach consists of parameterizing the trajectory. Next, we describe these approaches.

### 2.2. Swept volume interference

The volume containing all the points occupied by a moving object during a time period is called the *swept volume*. If the swept volumes for all the objects in a scene do not intersect, then no collision between them will occur during the specified time period. However, this is a sufficient, but not a necessary condition: It may happen that the swept volumes intersect but no collision takes place. This fact is shown in Figs. 1 and 2. In both situations, the same swept volumes are generated, but only in the first situation collision actually occurs.

In order for the condition to be also necessary, the sweep has to be performed according to the *relative* motion of one object with respect to another one, for each pair of objects. In this case, while one of the objects is considered fixed, the volume swept out by the other one during its relative motion is computed. This can be computationally very costly, although now it can be ensured that, if the swept volume intersects with the fixed one, collision actually happens.

The generation of the swept volume is also computationally expensive. This is the reason why many works adhering to this approach deal with convex approximations of the swept volume and, only when the global swept volumes intersect, they proceed to split the trajectory into pieces and to compute a convex approximation of the swept volume for each piece. For convex objects, Foisy and Hayward [4] have proved that the approximations obtained in the successive splittings of the trajectory converge to the real swept volume.

Simplifying alternatives are restricting the shapes and trajectories to very simple ones [5], and creating implicitly the swept volume from the volumes swept out by the primitives of the boundary representation *B*-rep [6]. Recall that an object can be described by the surface bounding it, so that the *B*-rep description of a polyhedra, for example, consists of a list of its boundary primitives (vertices, edges, and faces) together with their topological adjacencies.

### 2.3. Multiple interference detection

The simplest way to tackle collision detection is to sample object trajectories and repeatedly apply a static interference test. The way sampling is performed is crucial for the success of the approach. A too coarse sampling may miss a collision, while a too fine one may be computationally expensive. The reasonable way out is to apply *adaptive sampling*.

Ideally, the next time sample should be the earliest time at which a collision can really occur. The different sampling strategies differ in the way this earliest time is estimated. The most crude estimation is the one relating a lower bound on the distance between objects to an upper bound on their relative velocities [7,8].

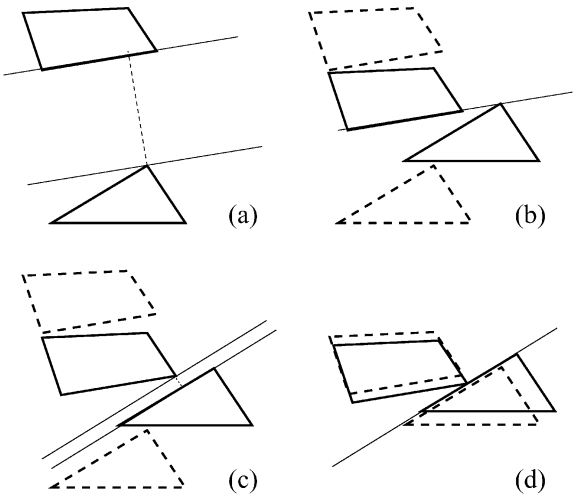


Fig. 3. Adaptive time sampling. The starting position is depicted in (a), where the closest points and the line joining them are computed. The projections of the objects on this line meet at instant (b), which is taken to be the next time sample. At this instant, the new closest points are computed (c), and the next time sample, where the polygons do actually collide, is determined in the same way (d).

More sophisticated strategies take not only distance into account, but also directional information. One such strategy [9] requires computing the closest points from two convex polytopes (extended to general convex objects in [10]) at the current time sample, as well as the line joining them. The first future instant at which the projections of the objects on the line meet is taken as the next time sample (see Fig. 3). Therefore, this technique can be viewed as a hybrid of sampling and projecting onto lower-dimensional subspaces (a 1D subspace in this case), according to the terminology introduced at the end of Section 2.1. Since the closest points between two objects lie always in their boundaries, it is usual practice to resort to *B*-reps when following a multiple interference detection approach. However, as we shall see in Section 4, to confine the application of the interference test to those object parts susceptible of colliding first, spatial partitioning techniques such as octrees and voxels have also been used in conjunction with this approach.

### 2.4. Trajectory parameterization

The collision instant can be analytically determined if the object trajectories are expressed as functions of a parameter (time). For example, consider the simple case in which a point undergoes a linear motion and we want to detect if it intersects a fixed triangle in space. Then, the parametric vector equation

$$\mathbf{p} + (\mathbf{p}' - \mathbf{p})t = \mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)u + (\mathbf{p}_2 - \mathbf{p}_0)v,$$

where $\mathbf{p}$ and $\mathbf{p}'$ are the initial and final positions of the point and the $\mathbf{p}_i$'s define the triangle, is set up and solved for the variables $u$, $v$, and $t$. $u$ and $v$ are parametric variables for the plane defined by the triangle, whereas $t$ is a time variable which is 0 at the beginning of the simulation step, and 1 at the end. If $0 \leqslant t \leqslant 1$ and $u \geqslant 0$ and $v \geqslant 0$ and $u + v \leqslant 1$, then the point intersects the triangle during the time step [11]. This vector equation represents three scalar equations in three unknowns which can be reduced to a single polynomial in $t$.

Conditions of intersection for general polyhedra following complex trajectories can be set up in the same way, the only difference being the degree of the polynomials in the variable $t$ to be solved. When rotations are present, the resulting expressions contain trigonometric functions, but they can also be reduced to polynomials in a single variable by means of a proper change of variables.

Depending on the trajectories, the degrees of the resulting polynomials may be arbitrarily high. Then, as polynomials of order 5 and above cannot be solved analytically, the determination of the collision instant can be computationally very expensive for arbitrary trajectories.

In [12], the problem is tackled in a radically different way: a trajectory connecting two arbitrary configurations for a moving polyhedron in a polyhedral environment is designed so that the obtained polynomials are of degree 3; i.e., the lowest possible degree when the moving polyhedron translates and rotates simultaneously. A polyhedra interference test is expressed as a combination of parameterized basic contact functions, these functions reflecting the spatial relationships between the primitives of the *B*-rep of the polyhedra. The zeros of these functions delimit several time intervals, whose combination according to the interference test provides the desired set of intervals over which objects would be intersecting, if they were adhering to the predefined trajectories. While [12] uses a parameterization based on quaternions, [13,14] follow the same approach using homogeneous coordinates.

In [15], the problem of detecting collisions between deformable models is regarded as a constrained minimization problem, which is solved using interval Newton methods.

In the context of Computer Graphics, a parameterized collision condition can be easily derived for triangulated surface representations [11], which can be extended to non-rigid time-dependent parametric surfaces [16].

## 3. Static interference detection

All but the last approach described in the preceding section eventually require to apply a static interference test between either 3D volumes or 4D ones. Here efficient

interference detection strategies are described, where the considered objects are convex or non-convex polyhedra. Convexity plays a very important role in the performance of interference detection algorithms, and it is therefore used as criterion for classifying these strategies.

### 3.1. Convex polyhedra

As pointed out in [17], intersection detection for two convex polyhedra can be done in linear time in the worst case. The proof is by reduction to linear programming, which is solvable in linear time for any fixed number of variables. If two point sets have disjoint convex hulls, then there is a plane which separates the two sets. The three parameters that define the plane are considered as variables. Then, a linear inequality is attached to each vertex of one polyhedron, which specifies that the point is on one side of the plane, and the same is done for the other polyhedron (specifying now the location on the other side of the plane).

Moreover, convex polyhedra can be properly preprocessed [18] to make the complexity of intersection detection drop to $O(\log n \log m)$. Preprocessing takes $O(n+m)$ time to build a *hierarchical representation* of two polyhedra with $n$ and $m$ vertices. The lowest level $P_1$ in the hierarchical representation is the original polyhedron, the highest one, say $P_r$, is a tetrahedron (where $r = O(\log n)$). At each level of the hierarchy, vertices of the original polyhedron are removed, such that they form an independent set (i.e., are not adjacent) in the polyhedron corresponding to the previous hierarchical level, and the corresponding edge and face adjacency relationships are updated. These hierarchical representations need to be computed only once, and they can be used for any interference query involving the same polyhedra. The algorithm described in [18] actually computes the *separation* (i.e., the minimum distance) between the polyhedra, interference is detected implicitly when this separation turns out to be null.

Actually, most algorithms used to detect interferences between convex polyhedra rely on the computation of the minimum distance and will be described in Section 4.4.1.

### 3.2. Polyhedra with convex faces

Disregarding the case in which one polyhedron is fully inside another one,[2] they intersect if their boundaries do. The detection of intersections between polyhedral surfaces reduces to detecting that an edge of one surface is piercing a face of the other surface. Since all edges are to be tested against all faces, the complexity of procedures following this scheme is necessarily $O(nm)$. However, when faces are convex polygons, interference detection becomes quite simple and easy to implement, as explained below.

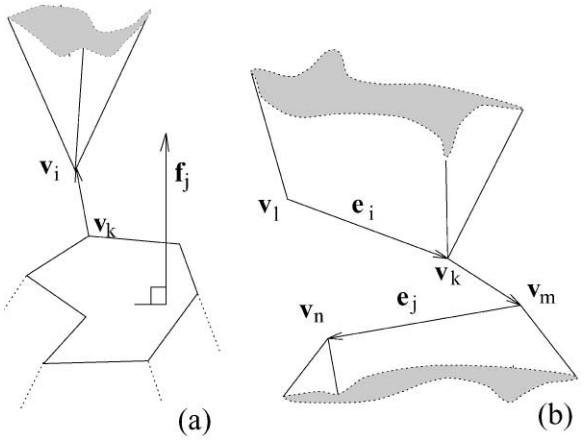


Fig. 4. Geometric elements involved in the definition of the predicates associated with Type-A (a) and Type-B (b) basic contacts.

This reduction of the interference problem to detecting edges piercing convex faces, formulated using the idea of *predicates* associated with *basic contacts*, was introduced in [19]. There are two basic contacts between two polyhedra. One takes place when a face of one polyhedron is in contact with a vertex of the other polyhedron (Type-A contact), and the other when an edge of one polyhedron is in contact with an edge of the other polyhedron (Type-B contact).

It is possible to associate a predicate with each basic contact, which will be true or false depending on the relative location between the geometric elements involved. Let us assume that face $F_i$ is represented by its normal vector $\mathbf{f}_i$; edge $E_j$, by a vector $\mathbf{e}_j$ along it; and vertex $V_k$ by its position vector $\mathbf{v}_k$. Although this representation is ambiguous, any choice of vector orientation leads to the same results in what follows.

According to Fig. 4(a), predicate $\mathbf{A}_{V_i,F_j}$, associated with a basic contact of Type-A, is defined as true when

$$\langle \mathbf{f}_j, \mathbf{v}_i - \mathbf{v}_k \rangle > 0 \tag{1}$$

for any vertex $V_k$ in face $F_j$, and false otherwise.

According to Fig. 4(b), predicate $\mathbf{B}_{E_i,E_j}$, associated with a basic contact of Type-B, is defined as true when

$$\langle \mathbf{e}_i \times \mathbf{e}_j, \mathbf{v}_m - \mathbf{v}_k \rangle > 0, \tag{2}$$

$V_m$ and $V_k$ being one of the two endpoints of $E_i$ and $E_j$, respectively, and false otherwise.

[2] Later it is shown that this case is no exception, since it can be dealt with using the same procedures.

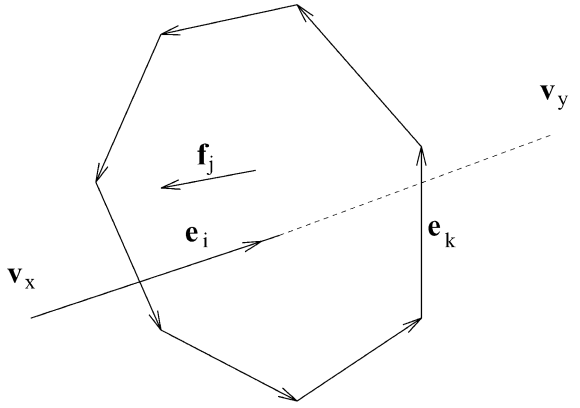


Fig. 5. Basic edge–face intersection test (convex faces).

It can be checked [19] that if one of the following boolean expressions:

$$\mathbf{O}^{out}_{E_i,F_j} = \neg \mathbf{A}_{V_x,F_j} \wedge \mathbf{A}_{V_y,F_j} \wedge \bigwedge_{E_k \in edges(F_j)} \mathbf{B}_{E_i,E_k},$$

$$\mathbf{O}^{in}_{E_i,F_j} = \mathbf{A}_{V_x,F_j} \wedge \neg \mathbf{A}_{V_y,F_j} \wedge \bigwedge_{E_k \in edges(F_j)} \neg \mathbf{B}_{E_i,E_k} \quad (3)$$

is true, then edge $E_i$ intersects convex face $F_j$, provided that its edges ($E_k$) are traversed counter-clockwise (refer to Fig. 5).

The case where one of the polyhedra is completely contained inside the other one can be handled with the same tools, by drawing an arbitrary ray from any point on the first polyhedron: if this ray intersects an odd number of faces of the second polyhedron (which can be checked with Eq. (3)), then inclusion exists.

### 3.3. General polyhedra

General non-convex polyhedra are qualitatively more difficult to handle. Therefore, most authors resort to decomposing them or their boundaries into convex parts (convex polyhedra or polygons, respectively), and to apply interference detection algorithms to those parts. Few works try to cope directly with non-convex polyhedra, without decomposing them. Moreover, note that, in general, it is not possible to express a non-convex curved object as the union of convex objects; for example, consider a block with a cylindrical hole drilled into it. This is the reason why the more accurate is the polyhedral approximation of a curved object, the more complex is, in general, its decomposition into convex parts.

#### 3.3.1. Decomposition into convex parts

It is possible to apply the above algorithms to non-convex polyhedra just by decomposing them into convex entities. Typically, decomposition is performed in a preprocessing step, and therefore has to be computed only once. The performance of this step is a tradeoff between the complexity of its execution and the complexity of the resulting decomposition. For example, the extreme case of solving the *minimum decomposition* problem is known to be NP-hard in general [20]. On the other hand, algorithms such as that in [21] can always partition a polytope of $n$ vertices into at most $O(n^2)$ convex entities.

Some interference detection algorithms work exclusively with convex polyhedra, others need only the faces of the polyhedron to be convex. In the first case, preprocessing will consist in a solid decomposition of the non-convex polyhedra, the output consisting of a set of smaller convex polyhedra (see [22,23]), whereas in the second case only a surface decomposition algorithm will be needed [24,25]. In any case, a number of additional fictitious entities are created, that have to be considered in the intersection tests.

#### 3.3.2. Direct approach

It has long been known [6] that, even for non-convex faces, a simple two-step test suffices to detect whether an edge intersects a face. First, check if the edge endpoints are on opposite sides of the face plane. If so, check whether the point of intersection between the edge and the face plane is located inside the face, by simply casting a ray from this point and determining how many times the ray intersects the polygon. Then, if this number is odd, intersection does exist (odd-parity rule). Note that the latter check corresponds directly to solving a *point-in-polygon* problem, for which several alternatives, different from that of shooting a ray, have been proposed [26, p. 239].

As mentioned in the preceding section, the application of this test to all edge–face pairs leads to an $O(nm)$ complexity. Thus, a quadratic number of intersection points may need to be computed to ascertain that there is no intersection between two polyhedra. A way to avoid these intersection computations is to reduce the test to computing the signs of some determinants [27], as in many other problems arising in Computational Geometry [28].

Consider a face from one polyhedron, defined by the ordered sequence of vertices around it, represented by their position vectors $\mathbf{p}_1, \ldots, \mathbf{p}_l$, expressed in homogeneous coordinates (that is, $\mathbf{p}_i = (p_{x_i}, p_{y_i}, p_{z_i}, 1)$), and an edge from the other, defined by its endpoints $\mathbf{h}$ and $\mathbf{t}$. Then, consider a plane containing the edge and any other vertex, say $\mathbf{v}$, of the same polyhedron, so that all edges in the face whose endpoints are not on opposite sides of this plane are discarded. In other words, we define, according to Fig. 6, $s := \text{sign}|\mathbf{h}\ \mathbf{t}\ \mathbf{v}\ \mathbf{p}_i|$. Then, if $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ are on opposite sides, $s$ should have a different sign from that of $|\mathbf{h}\ \mathbf{t}\ \mathbf{v}\ \mathbf{p}_{i+1}|$.

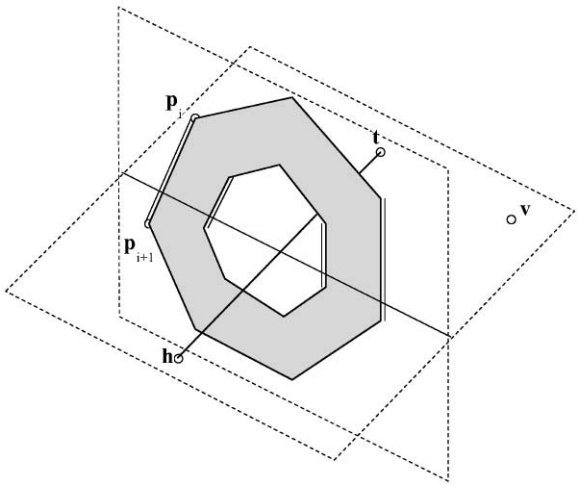


Fig. 6. Basic edge–face intersection test (general faces).

It can be checked [27] that, if the number of edges straddling the plane and satisfying $s\,sign|\mathbf{h}\ \mathbf{t}\ \mathbf{p}_i\ \mathbf{p}_{i+1}| > 0$ is odd, then the face is intersected by the edge. Actually, this is a reformulation of the odd parity rule that avoids the computation of any additional geometric entities such as those resulting from plane-edge or line-edge intersections.

The two special cases in which the arbitrary plane intersects at one vertex of the face or it is coplanar with one of the edges lead to determinants that are null. Actually, equivalent situations also arise when the ray shooting strategy is used. In order to take them into account, a simple modification of the odd parity rule has to be introduced as in [6].

It is also worth mentioning that, if the arbitrary point $\mathbf{v}$ is a vertex of one of the two faces in which the edge lies, different from its endpoints, the above approach is a generalization of Canny's predicates, since these predicates can also be expressed in terms of signs of determinants involving vertex locations [27].

Thus, in order to decide whether two non-convex polyhedra intersect, only the signs of some determinants involving the vertex location coordinates are required. Since the signs of all the involved determinants are not independent, it is reasonable to look for a set of signs from which all other signs can be obtained. This is discussed in [29] through a formulation of the problem in terms of oriented matroids.

## 4. Strategies for time and space bounding

Even if a basic interference test is made very efficient, as described in the preceding section, the collision detection algorithm can still be computationally expensive if the basic test has to be applied many times. Thus, the key aspect of any collision detection scheme is to restrict as much as possible *when* and *where* this test is applied, by taking advantage of the objects' geometry and the objects' dynamics, if this information is available. Knowing how the objects are moving and how far away they are from one another, it is possible to bound the time interval where the collision is likely to occur. Therefore, it is important to determine quickly the distance between objects. On the other hand, if the complexity of the objects is high, it is desirable to restrict the search for collisions to those object parts that may actually collide. Finally, if there are many moving objects in the scene, means to avoid having to check every pair of objects for collision need to be provided. These are the issues of the next subsections.

### 4.1. Distance computation for collision time bounding

Spherical representations are appealing because the elementary distance calculation between two spheres is trivial. The problem rather consists in determining which spheres of the representation have to be tested. In [30], objects are described in terms of *spherical cones* (generated by translating a sphere along a line and changing its radius) and *spherical planes* (which are obtained by translating a sphere in two dimensions, and eventually changing also its radius). These primitives can also be viewed as a collection of spheres. Any distance can be expressed as a combination of the distances between two *spherical cones* and between a sphere and a *spherical plane*.

In [31], ellipsoids are used to approximate convex polyhedra. A *free margin* function of one ellipsoid with respect to the other is then computed. This function behaves in a manner similar to the euclidean distance, except in that it is negative (instead of zero) as the ellipsoids interfere, and it is not symmetric in the general case.

Computing the distance between implicit and parametric surfaces is a very involved problem in general. Basic approaches to compute the closest points of two free-form objects neglect the fact that these points satisfy the necessary condition that their normals are aligned and opposite in direction. In [32], this constraint is used to obtain a measure of penetration distance, in the case that both objects intersect.

Most distance computation algorithms have been developed for convex polyhedra. Some exploit specific features of polyhedra and therefore cannot be used with other types of geometric models. Others, like the method explained in [33], can be used with spherical [34] or other non-polytopal surface descriptions [10]. These algorithms follow two main streams, namely the *geometric* and the *optimization* ones, which are the objects of the next subsections.

*4.1.1. The geometric stream*

The idea is to determine the closest points of two polyhedra, and then compute the euclidean distance between them. Three methods for determining the closest points have been proposed, two of them proceed by expanding an incremental representation in the direction of the minimum distance, while the third navigates along the boundaries of the polyhedra to find the closest points. These methods are described in the following paragraphs.

Using Dobkin and Kirkpatrick's hierarchical polyhedral representation (described in Section 3.1), distance computation can be performed in optimal $O(\log n \log m)$ time [18]. Every step in the closest points search procedure corresponds to a level in the hierarchical representation. In the first step, the closest points of two tetrahedra (the lowest level in the hierarchy) are trivially determined. Now, consider the direction of the segment that joins the closest points found at a given step. The two planes perpendicular to this direction that touch each polyhedron (at the level expanded so far) bound the zone where the next closest pair has to be searched for. The intersection of this zone with the polyhedra expanded at the next level may consist of either two simplices, one simplex or the empty set. If the closest points are not the same as in the previous step, then at least one of them belongs to one of these simplices. Therefore, every search step is restricted to at most two simplices. The number of steps is bounded by $\log n \log m$. Fig. 7 may help to understand this procedure.

The Minkowski difference $M_{P,Q} = \{p - q \mid p \in P, q \in Q\}$ of two polytopes $P$ and $Q$ has been used in distance computation algorithms [35], exploiting the fact that the distance between $P$ and $Q$ is equal to that from $M_{PQ}$ to the origin (Fig. 8). Since the complexity of computing the entire $M_{P,Q}$ is quadratic, a directional construction of this set, similar to that used by Dobkin and Kirkpatrick, has been proposed [33]. Starting from an arbitrary tetrahedron contained in $M_{P,Q}$, vertices closest to the origin in the direction of minimum distance are added one at a time, while non-relevant vertices are deleted, so that the search for the point closest to the origin is always performed on a simplex, as shown in Fig. 9. The "vertex-selection" part of the algorithm can be done in linear time: a single direction is tested over the set of vertices of one of the original polyhedra and the opposite direction over the vertices of the other one.

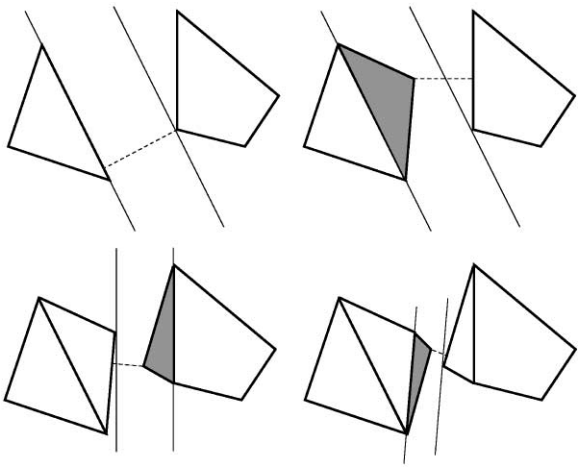

Fig. 7. The hierarchical representation allows to build up and search only those parts of the polygons where the closest points can be found.

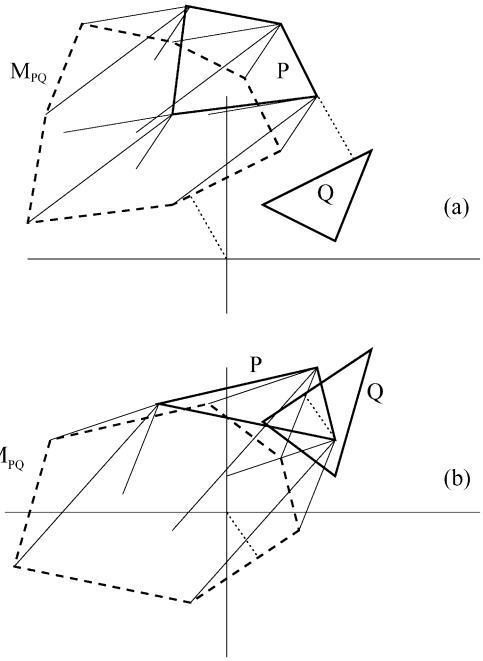


Fig. 8. The Minkowski difference $M_{P,Q}$ (------) can be constructed by taking the convex hull of the points resulting from subtracting the vertices of $Q$ from the vertices of $P$ (——). (a) The distance between $P$ and $Q$ ( $\cdots\ \cdots$ ) is the same as that from the origin to $M_{PQ}$. (b) If $P$ and $Q$ are interfering, the origin will be within $M_{PQ}$.

An alternative to the incremental construction of data structures, for speeding up distance computation, is to navigate along the boundaries of the involved polyhedra in the direction of decreasing distance. The key notion here is that of *Voronoi region* (refer to Fig. 10). Every feature (vertex, edge or face) of a polyhedron has associated one such region, consisting of all the points that are closer to it than to any other feature. The Voronoi regions for rectangular boxes were introduced in [36], and extended to convex polyhedra in [17], where an incremental algorithm for distance computation was

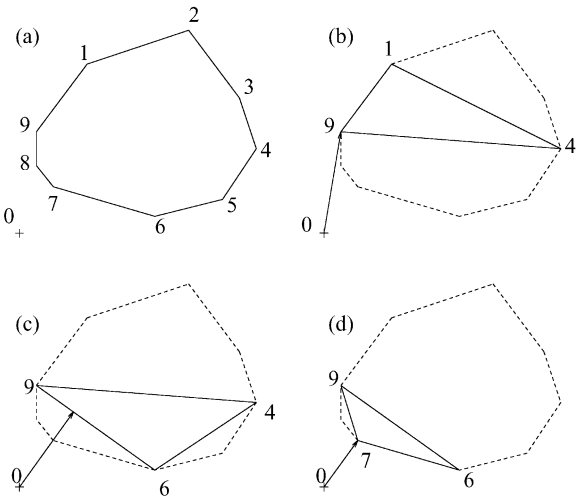


Fig. 9. (a) The closest point from the Minkowski polygon to the origin 0 has to be determined. (b) The first simplex is chosen arbitrarily, and the segment $\overline{09}$ realizing the minimum distance is computed. (c) The vertex 6 whose projection on this segment is closest to the origin is selected, vertex 1 is deleted, and the new direction realizing the minimum distance is computed. (d) The next vertex selected, 7, turns out to be the closest point from the Minkowski polygon to the origin.

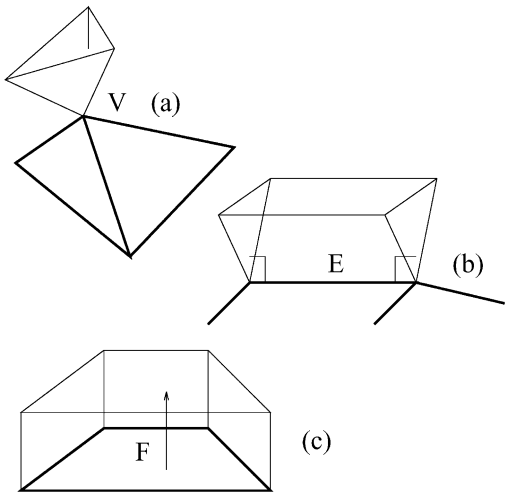


Fig. 10. Voronoi regions of a vertex (a), an edge (b), and a face (c).

proposed. The algorithm works as follows. First, two arbitrary features are selected and the closest points between them are obtained. If each of the two points belongs to the Voronoi region of the other feature, then they are actually the sought closest points between the polyhedra and the procedure stops. If not, each point has to be closer to another neighboring feature, which is selected, and these steps are repeated until the condition of inclusion in the respective Voronoi regions is met. This algorithm is linear in the total number of features. Note that, if the polyhedra are intersecting, the algorithm would go into a cyclic loop. To overcome this difficulty, some authors have extended the space partition to the interior of the polyhedron, by defining *pseudo*-Voronoi regions whose boundaries are faces determined by the centroid of the polyhedron and its edges [37–40]. These pseudo-Voronoi regions are used only to determine if the polyhedra interpenetrate or not. V-Clip [41] does also handle the case where the polyhedra interpenetrate and is very robust in both cases. It avoids the explicit computation of the closest points between features, by using simple clipping operations together with scalar derivative tests. The code is simple and its implementation does not require to specify any numerical tolerance.

In [17], another important point is addressed: consider that the distance between two polyhedra has to be computed as they move along a finely discretized path. The closest features do not change often, and a change almost always involves neighboring features, due to the convexity of the polyhedra and the small discretization step. Therefore, not an arbitrary pair of features, but the closest features at the previous step are taken as initial features for the next step. Simple preprocessing of the polyhedra, so that every feature has a constant number of neighboring features, allows the distance computation algorithm, once initialized, to run in expected constant time.

In [42–44], this ability to track the distance between two convex polyhedra was also analyzed for the algorithm described in [33], showing that a minor modification also gives it a expected constant execution time.

#### *4.1.2. The optimization stream*

Distance is viewed here as a quadratic function to be minimized, under linear constraints due to the convexity of the polyhedra. Formally, in [45], the function $f(p,q) = \|p-q\|^2/2$ is minimized subject to the linear constraints $\langle p, n_i^P\rangle \leqslant d_i^P$, $i = 1,\ldots,k^P$ and $\langle q, n_j^Q\rangle \leqslant d_j^Q$, $j = 1,\ldots,k^Q$, where these constraints mean that $p \in P$ and $q \in Q$, with the polyhedra $P$ and $Q$ being described as intersections of halfspaces. Rosen's gradient projection algorithm is used. At each step, the active constraints are determined (those where equality holds, with a certain tolerance) and Kuhn–Tucker conditions are used to test whether the global minimum has been attained. If this is not the case, the coefficients of the Kuhn–Tucker conditions are used to find the new search direction. There are two alternatives for obtaining the starting points: to apply a simplex minimization subalgorithm along the direction defined by the centroids of the polyhedra, or to obtain the points where the segment joining the centroids intersects the boundaries of the polyhedra.

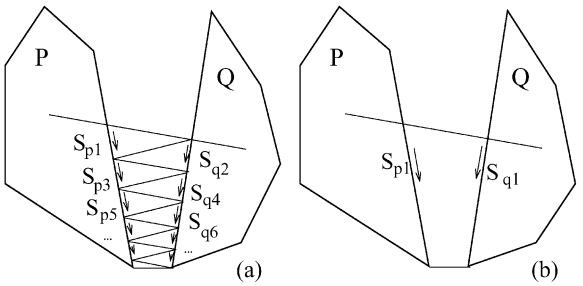


Fig. 11. The zig–zag phenomenon (a) is avoided if the projection of $S_{p1}$ on the active constraint of $Q$ is taken as the search direction $S_{q1}$ (b).

In applying Rosen's gradient projection method as Bobrow did, a convergence problem may occur, as stated in [46]. This problem is called the *zig–zag phenomenon* and it appears when the Kuhn–Tucker conditions are satisfied alternatively at each polyhedron. This happens because a zero vector is given as search direction on the polyhedron where the Kuhn–Tucker conditions are satisfied. The solution proposed by these authors is to consider as search direction for this polyhedron the projection of the search direction for the other polyhedron on the active constraints of the first one, instead of the zero vector, as shown in Fig. 11.

Certain quadratic optimization problems can be solved in linear time, as shown in [47]. In particular, as already mentioned in Section 3.1, Lin and Canny [17] proved that the computation of distances between convex polyhedra is one such problem. In [48], the complexity of computing other measures of proximity between polyhedra are discussed.

The described algorithms based on optimization techniques have not been proved to be superior to those based on geometric considerations which, in practice, have become the prevalent ones in most implementations, mainly the one described in [17] because of its conceptual simplicity.

No work has been devoted specifically to distance computation between non-convex polyhedra. In the context of collision detection, non-convex objects are usually approximated by simpler convex shapes, and a conservative lower bound on the distance is thus obtained. Some authors that deal with convex polyhedra mention the possibility of extending their algorithms to non-convex ones by decomposing them into convex entities, as explained in Section 3. Unfortunately, if the number of generated convex entities is important, a large number of pairwise distances have to be computed, and although the individual objects are simpler, the net result is an important increment in the global complexity. As a consequence, it is important to restrict the pairwise distances to be computed to those entities included in regions most likely to collide. The way these regions can be efficiently computed is described in the next section.

### 4.2. Bounding collision areas in objects

Three strategies can be followed to focus the search for collisions on relevant portions of the objects. One is to exploit a hierarchy of bounding volumes, another is to determine forefront features in the direction of motion, and the third is to exploit temporal coherence to keep track of closest points. Hierarchical bounding can be applied to both volume and boundary representations, while the latter two strategies are specifically suited to boundary representations.

#### 4.2.1. Hierarchical volume bounding

The idea behind the approaches using a hierarchy of bounding volumes is to approximate the objects (with bounding volumes) or to decompose the space they occupy (using decompositions), to reduce the number of pairs of objects or primitives that need to be checked for contact. Two main advantages of these approaches must be highlighted: (a) in many cases an interference or a non-interference situation can be easily detected at the first levels in the hierarchy, and (b) the refinement of the representation is only necessary in the parts where collision may occur. Space and object partitioning representations for collision detection are surveyed next.

Octrees [49] or octree-like structures [50], BSP-trees [51], brep-indices [52], tetrahedral meshes [53], and regular grids [54] are all examples of *spatial partitioning representations*. By dividing the space occupied by the objects, one needs to check for contact between only those pairs of objects (or parts of objects) that are in the same or nearby cells of the decomposition. Using such decompositions in a hierarchical manner can further speed up the collision detection process. Octrees and BSP-trees have been the most widely used. As their names indicate, octrees recursively partition cubes into octants, and BSP-trees recursively cut the space by hyperplanes. The octree representation allows to avoid checking for collision in those parts of space where octants are labeled "empty", that is, those entirely free of objects. If a "full" (totally occupied by an object) or "mixed" (partially occupied) octant is inside a "full" one of another object, interference occurs between them. Only if a "full" or "mixed" octant is inside a "mixed" one, the representation has to be further refined. The natural octree primitive is a cube [55,56], but there exist also models based on the same idea where spheres are used, as octant-including volumes [57] or within a different space subdivision technique, where the subdivision branching is 13 instead of 8 [58]. BSP-trees [59] can be considered a crossing between octrees and boundary representations. In them, partitioning is not restricted to be axis-aligned, as in octrees, and therefore transformations

(orientation changes, for example) can be simply computed by applying the transformation to each hyperplane, without rebuilding the whole representation.

*Object partitioning representations* are used in the collision detection context to simultaneously attain the following three goals: (a) approximate tightly the input primitives; (b) admit a rapid intersection test to determine if two bounding volumes overlap; and (c) be updated quickly when the primitives (and consequently the bounding volumes) are rotated and translated in the scene. Unfortunately, as recognized in [2], these objectives are usually in conflict, so a balance among them must be reached.

Commonly used object partitioning representations use hierarchies of spheres [60–64] or spherical shells [65,66] for bounding at different resolution levels. Inner and outer bounds are often used.

Volume bounding strategies can be used in conjunction with boundary representations. Hierarchies of volumes enclosing boundary features permit focussing on those susceptible of interfering. Thus, octrees have been used to build bounding box hierarchies around features of the polyhedron belonging to its convex hull and around concavities of non-convex polyhedra [39]. Once intersection has been detected between the convex hulls of two polyhedra, a sweep and prune algorithm is applied to traverse the hierarchies up to the leaf level, where overlapping boxes indicate which faces may intersect, and exact contact points can be quickly determined.

In cluttered environments, *oriented bounding boxes* (OBB) perform better than axis-aligned boxes or spheres, as they fit the objects tighter and, therefore, less intersections between bounding volumes are reported. An OBB tree is used in [67] to represent polyhedra with triangulated boundaries. Overlaps between OBBs are rapidly determined by performing 15 simple axis projection tests (about 200 arithmetic operations), as proved by the authors through their *separating axis theorem*. Routines for building OBB trees, as well as for performing fast overlap tests between them can be found in the RAPID interference detection package [67]. OBBs have also been used in [54] where, if interference between boxes is not discarded, OBBs are further subdivided into voxels and, if needed, the interference test is finally applied to boundary features of objects.

On the other hand, a hierarchy based on *axis-aligned bounding boxes* (AABB) has the advantage that the intersection test between each pair of AABB trees is not orientation-dependent, as is the case of OBB trees. In other words, the boxes in AABB trees need to be projected on the coordinate axes only once, whereas for each pair of boxes of OBB trees undergoing an intersection test, one box has to be projected onto the axes of the other one. Furthermore, AABB trees need less memory and are faster to build, and are even faster to update [68], which makes them specially suitable for deformable models. SOLID is a collision detection library that uses AABB trees for determining possible collisions in a scene composed of polygonal objects that may include complex deformable models [69].

Hierarchies of AABBs are also used in the context collision detection between deformable models in [70], where the problem of self-intersections is also considered. Self-intersections may be frequent in highly deformable models due to bending and wrinkling. This problem is treated in [71] where a surface hierarchy that captures the adjacency relationships is proposed. In [72], this hierarchical representation combined with a simple hierarchy of bounding boxes is used for handling both self-intersections and collisions between different objects.

The choice of "discrete orientation polytopes" ($k$-dops) as bounding volumes was made in [2] to attain a compromise between the relatively poor tightness of bounding spheres and AABBs, and the relatively high cost of overlap tests and updates associated with OBBs and convex hulls. $k$-dops are bounding volumes that are convex polytopes whose facets are determined by halfspaces with outward normals coming from a small fixed set of $k$ orientations. In the implementation reported in [2], their use compares favorably with RAPID, whose hierarchy is based on oriented volume boxes.

Though not hierarchical, the most common object partitioning representation is the *Constructive Solid Geometry* (CSG) *tree* and, in [73], a bounding technique tailored to this representation was proposed. The idea is to represent all objects to be checked for interference in one such tree and then iteratively compute simple enclosing volumes (the so-called *S-bounds*) for each node in the tree. The resulting bound at the root node delimits the part of the objects susceptible of interfering. The technique starts by placing S-bounds of the primitives at the leaves of the CSG tree. Then, S-bounds are alternatively propagated upwards and downwards according to the set operations attached to every node in the tree. Two examples of S-bounds, namely spheres and rectangular parallelepipeds aligned with the coordinate axes, are used and discussed in [73]. This technique has the advantages of hierarchical representations, as discussed at the beginning of this section, i.e., the cut-off of subtrees included in empty bounds, leading to possibly important computational savings, and the focussing of intersection searching on zones where intersection can actually occur. Although originally developed for 3D interference detection, the technique has been extended to extrusions [3].

#### 4.2.2. Bounding dependent on the direction of motion

If any kind of relative motion between two solids is allowed, every part of their boundaries may intersect. But if a polyhedron can only move in a specific way with respect to the other one, only certain parts of them can actually collide.

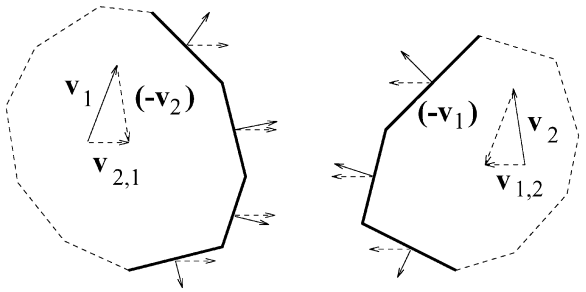


Fig. 12. Only the faces (shown as heavy lines) whose normals have positive projections on the relative motion vectors ($v_{2,1}$ and $v_{1,2}$) need to be considered.

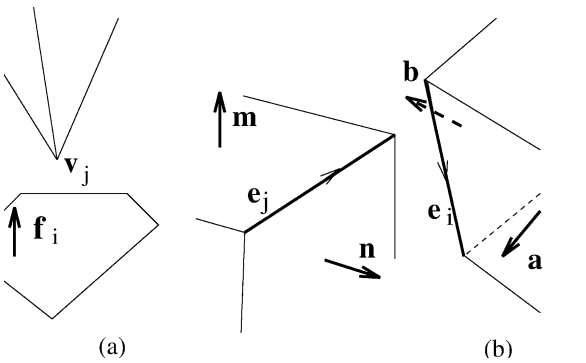


Fig. 13. (a) An applicable vertex ($V_j$)–face ($F_i$) pairing. (b) Edges $E_i$ and $E_j$ are also applicable.

*Back-face culling* techniques, which have been widely used in Computer Graphics to speed up the rendering of polyhedra, can also be used in the collision detection context to avoid unnecessary checking of boundary elements for collision, as shown in [74]. The basic idea consists of comparing the normal vectors of the faces of the polyhedra with the relative velocity vectors. A face is culled if its normal has a negative projection on the motion vector, as can be seen in Fig. 12. On the average, half of the faces of the two polyhedra are eliminated in this way.

Another possibility for feature bounding arises in the context of convex polyhedra subject to translational motions. *Applicability constraints* [75] permit detecting those vertex–face and edge–edge pairs that can actually come into contact (Fig. 13). The vertex–face applicability condition expresses the fact that a vertex can touch a face only if every adjacent edge projects positively on the face's normal (taking the vertex as origin of every edge interpreted as a vector). Analogously, the edge–edge applicability condition states that two edges can touch only if there exists a separating plane between their respective wedges.

The applicability constraints may be used as a preprocessing step in a collision detection scheme based on edge–face intersection tests. For each applicable vertex–face pair, only one of the edges adjacent to the vertex has to be tested for interference with the face. Any other edge–face test with this face can be cut off. In a similar way, edge–edge applicable pairs bound the number of edge–face tests to be performed. In [76], an efficient algorithm for bounding edge–face tests using applicability constraints is described. Experimental results show that the number of tests required is linear in the total number of edges, the constant of linearity being close to 1. The algorithm is based on a face orientation graph representation, where face adjacency relations are explicitly depicted. Both this representation and the bounding algorithm based on it have recently been extended to deal with non-convex polyhedra [77].

#### *4.2.3. Exploiting temporal coherence to track closest points*

The incremental minimum distance realization technique [17] has already been mentioned in Section 4.1. At a given instant, the boundary elements that realize the minimum distance must be close to those realizing it at the previous instant (this is known as geometric and temporal coherence), which are therefore taken as initial points for the search. In this case, it is not a specific orientation, but a neighborhood criterion which is used for saving computational effort. Based on this technique, the collision detection library I_COLLIDE has been developed and is publicly available on the web [38]. A library that unifies I_COLLIDE and RAPID in the framework of the VRML specification is described in [40].

The fact that the computation of the distance is not actually needed for reporting collision, and that it is not necessary to keep track of the pair of closest points unless a collision actually occurs, is used in [78] to develop Q_COLLIDE, a collision detection library also available on the web. The *separating vector* algorithm efficiently determines whether there exists a separating plane between two convex polyhedra (see Fig. 14). If so, they do not collide. Otherwise, the situation at the previous instant is examined: an improved version of the algorithm of Gilbert et al. [33] is applied in order to determine the pair of closest points (where the pair of supporting vertices given by the separating plane is a good initial guess for the closest points). As in I_COLLIDE, temporal coherence is exploited, so that the separating plane can be determined in expected constant time.

The interference test between two hierarchical volume representations having different defining reference frames can be very costly as to defeat the purpose of having a hierarchy. Thus, it is important to efficiently update the model as the objects rotate and translate. To this end,

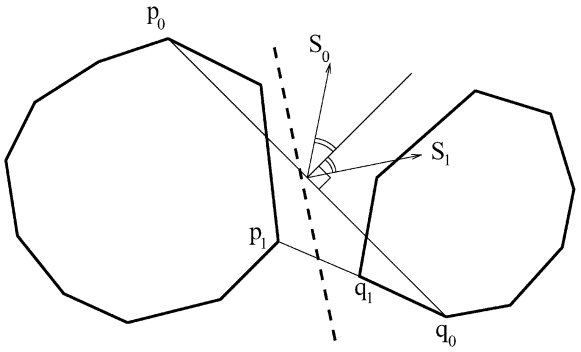


Fig. 14. A. separating vector $S_i$ is a vector that satisfies the condition $S_i \cdot (q_i - p_i) \geqslant 0$, for $p_i$ and $q_i$ the supporting vertices of two convex polytopes $P$ and $Q$ in the directions $S_i$ and $-S_i$, respectively. This is a sufficient condition for non-interference between $P$ and $Q$. The separating vector may be regarded as the normal to the separating plane between $P$ and $Q$. The next such vector to consider can be obtained by applying $S_{i+1} = S_i - 2(r_i \cdot S_i)r_i$, where $r_i = (q_i - p_i)/\|(q_i - p_i)\|$. In the case depicted in the figure, the separating plane, shown in bold dashed line, is obtained after one iteration. It has normal $S_1$ and contains point $(q_1 - p_1)/2$.

temporal coherence has also been exploited for updating hierarchical volume representations based on $k$-dops.

### 4.3. Priorizing collision pairs

We have seen how to focus the search for collisions on relevant portions of the objects. For highly cluttered workspaces, however, collision checking has to be performed for a potentially large number of pairs of objects. Thus, for the sake of efficiency, several criteria to priorize candidates for collision checking have also been proposed.

The first criterion one may consider is distance, but it is not enough if relative velocities are not taken into account, as pointed out in [79]. These authors introduce the concept of *awareness*, which approximates the earliest time at which two objects may collide, by considering the distance between them, their instantaneous relative velocity, together with velocity and acceleration bounds. Ordered according to increasing values of awareness, the pairs are partitioned into buckets, whose cardinalities are increasing powers of 2. Initially, an awareness value is computed for every pair to establish the buckets, but only one pair per bucket is updated at each subsequent time instant. Thus, pairs with lower awareness values get updated more frequently. Moreover, as their awareness values change, pairs may percolate from bucket to bucket.

In [80], a heap is used to store object pairs and earliest collision times, so that the pair on the top is the nearest to collide. This earliest collision time is computed from the distance between the objects, current velocities and accelerations, and acceleration bounds assuming ballistic trajectories for the objects. At each instant, integration of the dynamic state up to the time of collision is carried out for the pair on the top. Collision detection is performed for this pair, and if no collision actually occurs, the time of impact is recomputed and the heap updated. Only the objects whose bounding boxes for their swept volumes during the time-step intersect with other boxes are selected and included in the heap. The intersections between these $n$ boxes can be done in $O(n(1 + \log R))$, $R$ being the ratio of largest to smallest box size, as shown in [81].

A similar idea is followed in [37,38,82,83] where *temporal coherence* is exploited not only to speed up pairwise intersection detection (as mentioned in Section 4.2.3) but also to perform less of these pairwise tests. If time steps are small enough, there will be little change in the positions of the bounding boxes, and, consequently, in the sequence of intervals that these bounding boxes project onto the coordinate axes. Since bounding box interference is here determined by the simultaneous overlap of their projected intervals on the three axes, interval sorting techniques play here a crucial role. One such technique exploiting temporal coherence permits assessing interval overlap in expected linear time.

*Sweep* techniques have been extensively used to prune the number of object pairs to be checked for interference [84]. The idea is to sweep a plane through the scene and test for interference only those pairs of objects simultaneously intersecting the plane. A new usage of this technique is to use a 2D sweep to bound collision pairs in 3D. In [57], 4D hyper-trapezoids are used to bound the object along its motion. If one intersection between two hyper-trapezoids occurs, the corresponding objects are tested for collision. These intersections are computed from intersections between their faces. The problem is reduced, by successive projections, to a 2D segment intersection detection problem. The 2D sweep algorithm is described in [85] and runs in $O((m + k)\log m)$ time for $m$ segments that intersect $k$ times. Although for $n$ objects the worst-case value of $k$ is $O(n^2)$, empirical evidence shows that the average value of $k$ is much lower (0.07%).

## 5. Conclusions

Collision detection algorithms are of interest especially in the domains of Computer Graphics and Robotics. As described in Section 2, these algorithms usually rely on static interference tests, most of which have been developed in the domain of Computational Geometry. In this paper, we have summarized the basic ideas originated in the three domains mentioned, in relation to the topic of static and dynamic interference detection.

The dominating trend in Computational Geometry is that of obtaining algorithms with the best possible worst-case complexity. In the case of convex polyhedra, optimal algorithms have been developed for intersection and distance computation, which are linear in the total number of vertices. However, most algorithms developed within this domain have not been implemented and, therefore, their efficiency in a practical setting is hard to assess. The detailed complexity of the only known subquadratic algorithm for interference detection between non-convex polyhedra [86] has very large constant factors, which makes it unfeasible for practical purposes.

The situation in Robotics is in some sense opposite to the one just described. The key aspects in this domain are implementation and efficiency in practical settings. In many cases, the worst-case complexity of the developed algorithms either remains unknown or is far from optimal. Comparing algorithms in terms of CPU time is becoming a general practice. Several algorithms for interference detection between convex polyhedra have been shown to have a computational cost approximately linear in the total number of vertices. Usually, non-convex polyhedra are dealt with by decomposing them into convex pieces.

In Computer Graphics, the emphasis is placed on the possibility of detecting collisions in real-time, especially in computer animation applications, even if speed is gained at the cost of losing precision. This is the reason why hierarchical representations are often used in this domain, since they provide higher precision as one moves down the hierarchy, thus allowing to adapt output precision to the computing time available.

As for efficiency, a far more challenging application field has appeared recently: haptic interfaces require update rates of about 1 kHz, as compared with the 20–30 updates per second needed in "real-time" graphical applications. Algorithms developed so far [87] solve the problem of a probe point colliding with virtual objects, but this is clearly not sufficient as the feeling of one 3D object moving in contact with another object is to be displayed in a realistic manner.

As we have seen, most collision detection schemes only deal with polyhedral approximations. Nevertheless, there are some applications where this kind of approximation is not possible. The task of verifying whether a given tool, in numerically controlled machining, penetrates beyond a specified threshold the surface of a part to be machined is a good example. This is a challenging problem for collision detection not only due to the nature of the tool motion but also because, in this case, a polyhedral approximation is inadequate. When manufacturing plastic parts using moulds, the different elements of the mould have to separated to free the manufactured part. This motion has to be planned and coordinated to avoid collisions between the mould elements and the part itself. This is another application where collision detection between smooth surfaces would be of interest. Although some research has been performed on robust and interactive computation of closest features, much remains to be done in the direction of determining contact points and penetration distances between models with smooth surfaces (see [88] for some results). As a further step, accurate and efficient detection of contacts between deformable models is also a research area that will certainly deserve attention in the near future.

## Acknowledgements

This research has been partially supported by Comissionat per a Universitats i Recerca, grant 1997-SGR-00464, and by the Spanish Science and Technology Commission (CICYT) under Contract TAP99-1086-C03-01 ("Constraint-based computation in robotics and resource allocation").

## References

[1] Lin MC, Gottschalk S. Collision detection between geometric models: a survey. IMA Conference on Mathematics of Surfaces, San Diego, CA, vol. 1, May 1998. p. 602–8.

[2] Held M, Klosowski JT, Mitchell J. Evaluation of collision detection methods for virtual reality fly-throughs. Proceedings of the Seventh Canadian Conference on Computer Geometry, vol. 3, 1995. p. 205–10.

[3] Cameron SA. Collision detection by four-dimensional intersection testing. IEEE Transactions on Robotics Automation 1990;6(3):291–302.

[4] Foisy A, Hayward V. A safe swept volume method for collision detection. The Sixth International Symposium of Robotics Research, Pittsburgh, PE, October 1993. p. 61–8.

[5] Herman M. Fast, three-dimensional, collision-free motion planning. Proceedings of the IEEE International Conference on Robotics and Automation, vol. 2, April 1986. p. 1056–63.

[6] Boyse JW. Interference detection among solids and surfaces. Communication of the Association of the Computing Machinery 1979;22(1):3–9.

[7] Cameron SA. A study of the clash detection problem in robotics. Proceedings of the IEEE International Conference on Robotics and Automation, Saint Louis, MO, vol. 1, March 1985. p. 488–93.

[8] Culley RK, Kempf KG. A collision detection algorithm based on velocity and distance bounds. Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, vol. 2, April 1986. p. 1064–9.

[9] Gilbert EG, Hong SM. A new algorithm for detecting the collision of moving objects. Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale, AR, vol. 1, May 1989. p. 8–14.

[10] Gilbert EG, Foo C-P. Computing the distance between general convex objects in three-dimensional space. IEEE Transactions on Robotics and Automation 1990;6(1):53–61.

[11] Moore M, Wilhelms J. Collision detection and response for computer animation. ACM Computer Graphics 1988;22(4):289–98.
[12] Canny JF. Collision detection for moving polyhedra. IEEE Transactions on Pattern Analysis and Machinery Intelligence 1986;8(2):200–9.
[13] Jiménez P, Torras C. Collision detection: a geometric approach. In: Modelling and planning for sensor based intelligent robot systems. Singapore: World Scientific Pub. Co., November 1995. p. 68–85.
[14] Schömer E, Thiel C. Efficient collision detection for moving polyhedra. Proceedings of the Eleventh Annual Symposium on Computational Geometry, 1995. p. 51–60.
[15] Snyder JM, Woodbury AR, Fleischer K, Currin B, Barr AH. Interval methods for multi-point collisions between time-dependent curved surfaces. Proceedings of ACM Siggraph, 1993. p. 321–34.
[16] Von Herzen B, Barr AH, Zatz HR. Geometric collisions for time-dependent parametric surfaces. ACM Computer Graphics 1990;24(4):39–48.
[17] Lin MC, Canny JF. A fast algorithm for incremental distance calculation. In Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA, vol. 2, 1991. p. 1008–14.
[18] Dobkin D, Kirkpatrick D. Determining the Separation of Preprocessed Polyhedra — A Unified Approach. Lecture Notes in Computer Science, vol. 443, ICALP-90, 1990. p. 400–13.
[19] Canny J. The complexity of robot motion planning. Cambridge, MA: MIT Press, 1987.
[20] Bajaj C, Dey T. Convex decomposition of polyhedra and robustness. SIAM Journal on Computing 1992;21(2):339–64.
[21] Chazelle B. Convex partitions of polyhedra: a lower bound and a worst-case optimal algorithm. SIAM J. Comput. 1984;13:488–507.
[22] Chazelle B, Palios L. Decomposition algorithms in geometry. In: Bajaj C, editor. Algebraic Geometry and its Applications, vol. 5. Berlin: Springer, 1994. p. 419–47.
[23] Bern M. Triangulations. In: Goodman JE, O'Rourke J, editors. Handbook of discrete and computational geometry. Boca Raton, FL: CRC Press, 1997. p. 413–28.
[24] Chazelle B, Palios L. Decomposing the boundary of a nonconvex polytope. In Proceedings of the Third Scandinavian Workshop on Algorithm Theory, 1992. p. 364–75.
[25] Chazelle B, Dobkin D, Shouraboura N, Tal A. Strategies for polyhedral surface decomposition: an experimental study. Computational Geometry: Theory and Applications 1997;7(4–5):327–42, 484.
[26] O'Rourke J, editor. Computational Geometry in C, 2nd ed. Cambridge: Cambridge University Press, 1998.
[27] Thomas F, Torras C. Interference detection between nonconvex polyhedra revisited with a practical aim. Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, vol. 1, May 1994. p. 587–94.
[28] Avnaim F. Evaluating signs of determinants using single-precision arithmetic. Technical Report 2306, INRIA, 1994.
[29] Thomas F. An approach to the movers problem that combines oriented matroid theory and algebraic geometry. Proceedings of the IEEE International Conference on Robotics and Automation, Nagoya, J, vol. 3, May 1995. p. 2285–93.
[30] Tornero J, Hamlin J, Kelley RB. Spherical-object representation and fast distance computation for robotic applications. Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA, vol. 2, April 1991. p. 1602–8.
[31] Rimon E, Boyd SP. Obstacle collision detection using best ellipsoid fit. Journal of Intelligent and Robotic Systems 1997;18:105–26.
[32] Thomas F, Turnbull C, Ros L, Cameron S. Computing signed distances between free-form objects. Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, April 2000.
[33] Gilbert EG, Johnson DW, Keerthi S. A fast procedure for computing the distance between complex objects in three dimensional space. IEEE Journal of Robotics and Automation 1988;4(2):193–203.
[34] Hamlin GJ, Kelley RB, Tornero J. Efficient distance calculation using the spherically-extended polytope (s-tope) model. Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, vol. 3, May 1992. p. 2502–7.
[35] Cameron SA, Culley RK. Determining the minimum translational distance between two convex polyhedra. Proceedings of the IEEE International Conference on Robotics and Automation, SanFrancisco, CA, vol. 1, April 1986. p. 591–6.
[36] Meyer W. Distance between boxes: applications to collision detection and clipping. In Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, vol. 1, April 1986. p. 597–602.
[37] Lin MC, Manocha D, Canny JF. Fast contact determination in dynamic environments. Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, vol. 1, May 1994. p. 602–8.
[38] Cohen JD, Lin MC, Manocha D, Ponamgi MK. I-COLLIDE: an interactive and exact collision detection system for large-scale environments. Proceedings of ACM International 3D Graphics Conference, vol. 1, 1995. p. 189–96. http://www.cs.unc.edu/~geom/L-COLLIDE.html.
[39] Ponamgi MK, Manocha D, Lin MC. Incremental algorithms for collision detection between polygonal models. IEEE Transactions on Visualization and Computer Graphics 1997;3(1):51–64.
[40] Hudson TC, Lin MC, Cohen JD, Gottschalk S, Manocha D. V-collide: accelerated collision detection for vrml. Proceedings of VRML, 1997. http://www.cs.unc.edu/~geom/V_COLLIDE .html.
[41] Mirtich B. V-clip: fast and robust polyhedral collision detection. ACM Transactions on Graphics 1998; 17(3):177–208. http://www.merl.com/projects/vclip/.
[42] Cameron SA. A comparison of two fast algorithms for computing the distance between convex polyhedra. IEEE Transactions on Robotics Automation 1997; 13:915–20.
[43] Cameron SA. Enhancing gjk: computing minimum and penetration distances between convex polyhedra. In Proceedings of the IEEE International Conference on Robotics and Automation, Albuquerque, NM, April 1997. p. 3112–7.

[44] Van der bergen G. A fast and robust gjk implementation for collision detection of convex objects, 1999, submitted. Available at http://www.win.tue.nl/cs/tt/gino/solid.
[45] Bobrow JE. A direct optimization approach for obtaining the distance between convex polyhedra. International Journal of Robotics Research 1983;8(3):65–76.
[46] Zeghloul S, Rambeaud P, Lallemand JP. A fast distance calculation between convex objects by optimization approach. Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, vol. 3, May 1992. p. 2520–5.
[47] Megiddo N, Tamir A. Linear time algorithms for some separable quadratic programming problems. Operations Research Letters 1993;13(4):203–11.
[48] Sancheti NK, Keerthi SS. Computation of certain measures of proximity between convex polytopes: a complexity viewpoint. Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, vol. 3, May 1992. p. 2508–13.
[49] Hamada K, Hori Y. Octree-based approach to real-time collision-free path planning for robot manipulator. ACM96-MIE, 1996. p. 705–10.
[50] Bandi S, Thalmann D. An adaptive spatial subdivision of the object space for fast collision detection of animating rigid bodies. Eurographics'95, Maastricht, August 1995. p. 259–70.
[51] Naylor BF, Amatodes JA, Thibault WC. Merging bsp trees yields polyhedral set operations. Computer Graphics, SIGGRAPH' 90 Proceedings, Dallas, TX, vol. 24, May 1990. p. 115–24.
[52] Bouma W, Vanecek G. Collision detection and analysis in a physical based simulation. Eurographics Workshop on Animation and Simulation, Vienna, September 1991. p. 191–203.
[53] Klosowski J, Held M, Mitchell J, Sowizral H, Zikan K. Efficient collision detection using bounding volume hierarchies of k-dops.. IEEE Transactions on Visualization and Computer Graphics 1998;4(1).
[54] García-Alonso A, Serrano N, Flaquer J. Solving the Collision Detection Problem. IEEE Computer Graphics and Applications 1994;14(3):36–43.
[55] Ahuja N, Chien RT, Yen R, Bridwell N. Interference detection and collision avoidance among three dimensional objects. I Annual National Conference on AI, August, Stanford University, 1980. p. 44–8.
[56] Hayward V. Fast collision detection scheme by recursive decomposition of a manipulator workspace. In Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, vol. 2, 1986. p. 1044–9.
[57] Hubbard PM. Interactive collision detection. Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality, vol. 1, October 1993. p. 24–31.
[58] Liu Y-H, Arimoto S, Noborio H. A new solid model hsm and its application to interference detection between moving objects. Journal of Robotic Systems 1991;8(1):39–54.
[59] Thibault WC, Naylor BF. Set operations on polyhedra using binary space partitioning trees. ACM Computer Graphics 1987;21(4).
[60] Del Pobil AP, Serna MA, Llovet J. A new representation for collision avoidance and detection. Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, vol. 1, May 1992. p. 246–51.
[61] Martínez B, Del Pobil AP, Pérez M. Very fast collision detection for practical motion planning. Part I: the spatial representation. Proceedings of the IEEE International Conference on Robotics and Automation, Leuven, Belgium, vol. 1, May 1998. p. 624–9.
[62] Quinlan S. Efficient distance computation between non-convex objects. Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, vol. 4, 1994. p. 3324–9.
[63] Palmer IJ, Grimsdale RL. Collision detection for animation using sphere-trees. Computer Graphics Forum 1995;14(2):105–16.
[64] Hubbard PM. Real-time collision detection and time-critical computing. Proceedings of the First ACM Workshop on Simulation and Interaction in Virtual Environments, vol. 1, 1995. p. 92–6.
[65] Bonner S, Kelley RB. A representation scheme for rapid 3-D collision detection. Proceedings of the IEEE International Symposium on Intelligent Control, Arlington, VA, August 1988. p. 320–5.
[66] Krishnan S, Gopi M, Lin M, Manocha D, Pattekar A. Rapid and accurate contact determination between spline models using shelltrees. Eurographics'98, Leeds, UK, March 1998.
[67] Gottschalk S, Lin MC, Manocha D. Obb-tree: a hierarchical structure for rapid interference detection. Proceedings of ACM Siggraph'96, 1996. http://www.cs.unc.edu/~geom/OBB/OBBT.html.
[68] Van der bergen G. Efficient collision detection of complex deformable models using aabb trees. Journal of Graphic Tools 1997;2(4):1–13.
[69] Van der bergen G. SOLID. Software Library for Interference Detection, 1999. Available at http://www.win.tue.nl/cs/tt/gino/solid.
[70] Hughes M, DiMattia C, Lin MC, Manocha D. Efficient and accurate interference detection for polynomial deformation. Proceedings of Computer Animation '96 Conference, 1996.
[71] Volino P, Thalmann NM. Collision and self-collision detection: efficient and robust solutions for highly deformable surfaces. Eurographics Workshop on Computer Animation and Simulation'95, Maastricht, The Netherlands, 1995.
[72] Volino P, Thalmann NM. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. Eurographics'94, Computer Graphics Forum, Oslo, Norway, vol. 13, 1994. p. 155–66.
[73] Cameron SA. Efficient intersection tests for objects defined constructively. International Journal of Robotics Research 1989;8:3–25.
[74] Vanecek G. Back-face culling applied to collision detection of polyhedra. Journal of Visualization and Computer Animation 1994;5(1):55–63.
[75] Donald BR. A search algorithm for motion planning with six degrees of freedom. Artificial Intelligence 1987;31(3):295–353.
[76] Jiménez P, Torras C. Speeding up interference detection between polyhedra. Proceedings of the IEEE International

Conference on Robotics and Automation, Minneapolis, MN, vol. 2, April 1996. p. 1485–92.

[77] Jiménez P, Torras C. Benefits of applicability constraints in decomposition-free interference detection between non-convex polyhedral models. Proceedings of the IEEE International Conference on Robotics and Automation, Detroit, MI, May 1999.

[78] Chung K. An efficient collision detection algorithm for polytopes in virtual environments. Master's thesis, The University of Hong Kong, 1996. http://www.cs.hku.hk/~tlchung/collision_library.html.

[79] Foisy A, Hayward V, Aubry S. The use of awareness in collision prediction. Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati, OH, vol. 1, 1990. p. 338–43.

[80] Mirtich B, Canny J. Impulse-based dynamic simulation. Proceedings of the Workshop on Algorithmic Foundations of Robotics, 1994.

[81] Overmars M. Point location in fat subdivisions. Information Processing Letters 1992;44:261–5.

[82] Lin MC. Efficient collision detection for animation and robotics. Ph.D. thesis, University of California, Berkeley, 1993.

[83] Ponamgi MK, Manocha D, Lin MC. Incremental algorithms for collision detection between solid models. Proceedings of ACM/Siggraph Symposium on Solid Modelling, vol. 1, 1995. p. 293–304.

[84] Turk G. Interactive collision detection for molecular graphics. Master's thesis, University of North Carolina, 1989.

[85] Bentley JL, Ottmann TA. Algorithms for reporting and counting geometric intersections. IEEE Transactions on Computing 1979;28(9):643–7.

[86] Pellegrini M. Stabbing and ray shooting in 3-space. Proceedings of the Sixth ACM Symposium on Computational Geometry, 1990. p. 177–86.

[87] Gregory A, Lin MC, Gottschalk S, Taylor R. Fast and accurate collision detection for haptic interaction using a three degree-of-freedom force–feedback device. Computational Geometry: Theory and Applications, to appear.

[88] Turnbull C, Cameron S. Computing distances between Nurbs-defined convex objects. Proceedings of the IEEE International Conference on Robotics and Automation, Leuven, Belgium, May 1998. p. 3686–90.