Collision Detection Algorithms for Motion Planning

P. Jiménez, F. Thomas and C. Torras

Institut de Robòtica i Informàtica Industrial, Barcelona

1 Introduction

Collision detection is a basic tool whose performance is of capital importance in order to achieve efficiency in many robotics and computer graphics applications, such as motion planning, obstacle avoidance, virtual prototyping, computer animation, physical-based modeling, dynamic simulation, and, in general, all those tasks involving the simulated motion of solids which cannot penetrate one another. In these applications, collision detection appears as a module or procedure which exchanges information with other parts of the system concerning motion, kinematic and dynamic behaviour, etc. It is a widespread opinion to consider collision detection as the main bottleneck in these kinds of applications.

In fact, static interference detection, collision detection and the generation of configuration-space obstacles can be viewed as instances of the same problem, where objects are tested for interference at a particular position, along a trajectory and throughout the whole workspace, respectively. The structure of this chapter reflects this fact.

Thus, the main guidelines in static interference detection are presented in Section 2. It is shown how hierarchical representations allow to focus on relevant regions where interference is most likely to occur, speeding up the whole interference test procedure. Some interference tests reduce to detecting intersections between simple enclosing shapes, such as spheres or boxes aligned with the coordinate axes. However, in some situations, this approximate approach does not suffice, and exact basic interference tests (for polyhedral environments) are required. The most widely used such test is that involving a segment (standing for an edge) and a polygon in 3D space (standing for a face of a polyhedron). In this context, it has recently been proved that interference detection between non-convex polyhedra can be reduced, like many other problems in Computational Geometry, to checking some signs of vertex determinants, without computing new geometric entities.

Interference tests lie at the base of most collision detection algorithms, which are the subject of Section 3. These algorithms can be grouped into four approaches: multiple interference detection, swept volume interference, space-time volume intersection, and trajectory parameterization. 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. The efficiency of a basic interference test does not guarantee that a collision detection algorithm based on it is in turn efficient. The other key factor is the number of times that this test is applied. Therefore, it is important to restrict the application of the interference test to those instants and object parts at which a collision can truly occur. Several strategies have been developed: 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, orientation-based pruning criteria and space partitioning schemes.

Section 4 describes how motion planners adopt different strategies with respect to the use of static interference and collision detection procedures, depending on their global or local nature. While global planners use static interference tests, or their generalizations, to generate a detailed description of either configuration-space obstacles or free-space connectivity, incremental and local path planners avoid this costly computation by fully relying on collision detection tests during the search process.

Finally, some conclusions are sketched in Section 5.

2 Interference detection

Objects to be checked for interference are usually modeled by composing simple shapes. Hierarchies of spheres (or other primitive volumes) and polyhedral approximations are the most commonly used. The former exploit the low cost of detecting interference between spheres, which reduces to comparing the distance between their centers and the sum of their radii. This type of model is particularly adequate in situations not demanding high accuracy, since achieving that would require going down many levels in the hierarchy. Objects with planar faces and subject to small tolerances are usually dealt with using polyhedral representations of their boundaries.

Hierarchical approximations permit focusing on the regions susceptible of interfering, as described in Section 2.1. Then, basic interference tests, which are the subject of Section 2.2, need only be applied within the focused regions.

2.1 Focusing on relevant regions

The two main approaches to confine the search for interferences to particular portions of the solids are representation dependent. On the one hand, there are algorithms that bound volume portions, and they are suited for volume representations, like Constructive Solid Geometry (CSG), octrees, or representations based on spheres. On the other hand, there are procedures that restrict the elements of the boundary of the objects that can intersect, and these algorithms are of course used together with boundary representations.

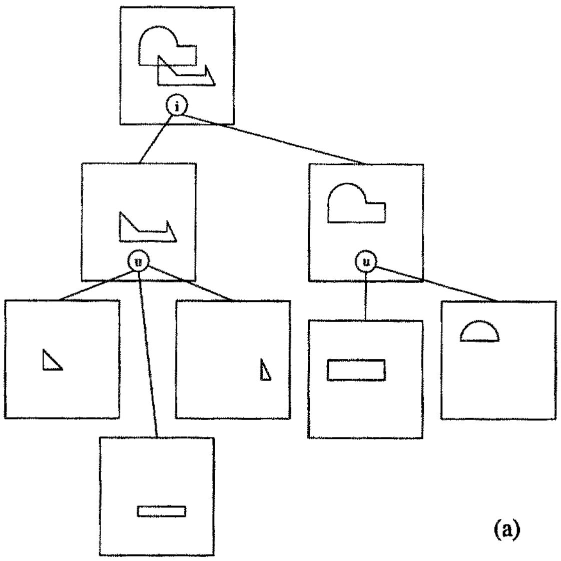
Hierarchical volume representations Two advantages of hierarchical representations must be highlighted:

- In many cases an interference or a non-interference situation can be easily detected at the first levels of the hierarchy. This leads to substantial savings under all interference detection schemes.
- The refinement of the representation is only necessary in the parts where collision may occur.

There are two types of bounding techniques for hierarchical volume representations, those that are based on an object partition hierarchy, and those where subregions of a space partition are considered.

Object partition hierarchies The so called "S-bounds" were developed and used in [8] for bounding spatially the part of the CSG tree that represents an intersection between two solids. S-bounds are simple enclosing volumes of the primitives at the leaves of the CSG tree: two examples are shown and discussed in [8] where rectangular parallelepipeds aligned with the coordinate axes and spheres are used as S-bounds. According to the set operations attached to every node in the tree, the S-bound corresponding to the root of the CSG intersection tree can be obtained after a number of pseudo-union and intersection operations of S-bounds. An algorithm that runs upwards and downwards on the tree performs all these operations (see Figs. 1 and 2). The main advantages of this procedure are the cut-off of subtrees included in empty bounds, leading to possibly important computational savings, and the focusing of intersection searching on zones where intersection can actually occur.
The "successive spherical approximation" described in [6] allows focusing on the region of possible interference by checking intersection of spherical sectors at different levels in the hierarchy (Fig. 3). Hierarchies based on spheres that bound objects at different levels of refinement are also used in [50] and in [52].

Space partition hierarchies The *octree* representation allows to avoid checking for collision in those parts of the workspace where octants are labelled *empty*, that is, where no part of any object exists. If a *full* (totally occupied by the solid) or *mixed* (partially occupied) octant is inside a *full* one of the other solid, interference occurs. Only if a *full* or *mixed*



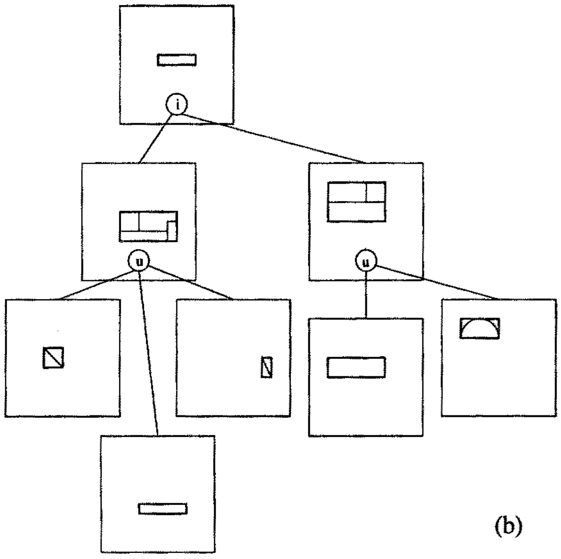
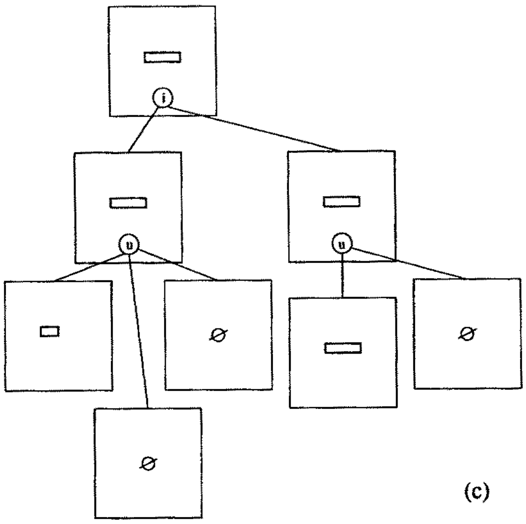


Fig. 1. *S-bounds.* (a) *The intersection (i) between two polygons described by their CSG representations has to be computed.* (b) *The rectangular boxes that bound the primitives are combined and the boxes corresponding to the higher levels are determined, according to the nature of the nodes (union or intersection).*



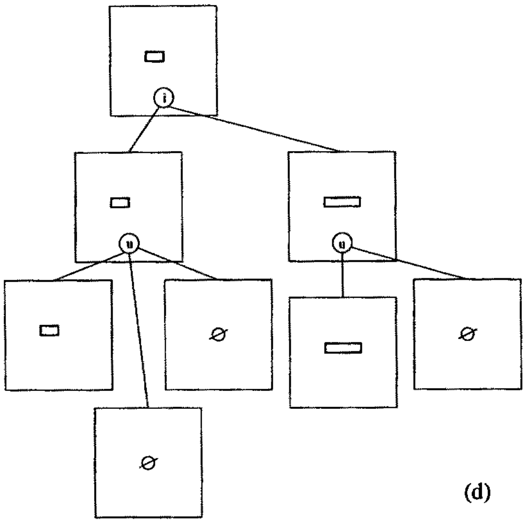


Fig. 2. *S-bounds (cont.).* (c) *The box obtained at the root node is intersected with the boxes of nodes at lower levels. The empty set is obtained for some nodes, which can be eliminated.* (d) *The representation is once again explored upwards, and a smaller box is obtained at the root. If the process is repeated once more every node will contain the small box or the empty set. This small box bounds the region where intersection has to be looked for.*

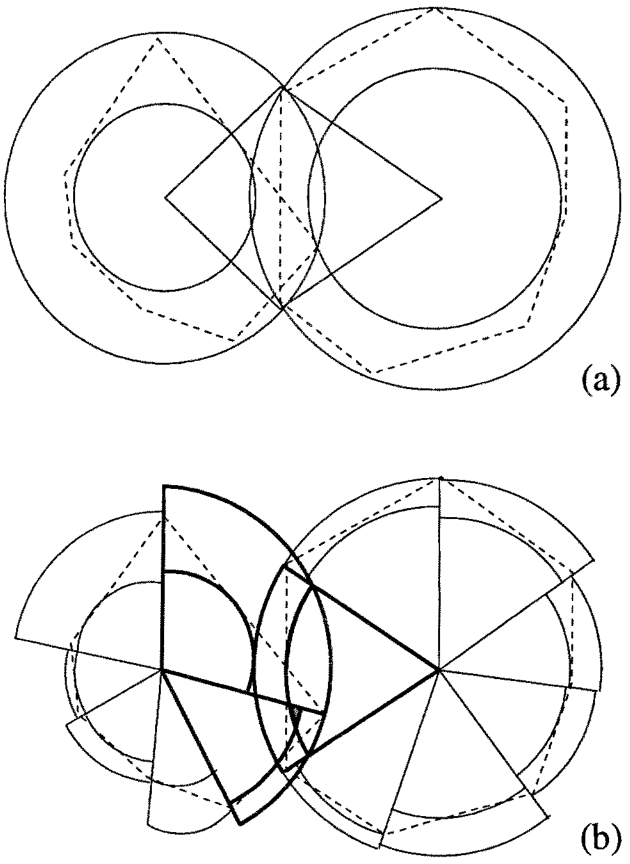


Fig. 3. (a) *Interference cannot be decided at the first level in the hierarchy, since neither the inner circles intersect nor the outer circles are disjoint. Nevertheless, the region of possible interference can be bounded, using the intersection points of the outer spheres.* (b) *At the next level two inner sectors intersect, thus interference exists.*

octant is inside a *mixed* one, the representation has to be further refined. The "natural" octree primitive is a cube [1,27], but there exist also models based on the same idea where spheres are used, as octant-including volumes [31] or within a different space subdivision technique, where the subdivision branching is 13 instead of 8 [39]. In the *binary space partition tree* [56], a binary tree is constructed that represents a recursive partitioning of space by hyperplanes. The authors describe such representation as a "crossing between octrees and boundary representations", but partitioning is not restricted to be axis-aligned, as in the octree representation, and therefore transformations (a change in orientation, for example) can be simply computed by applying the transformation to each hyperplane, without rebuilding the whole representation.

Boundary representations Hierarchical representations associated to bounding volumes that contain boundary features allow to restrict the effort of determining which parts of the objects boundaries may intersect to the most "promising" parts. Octrees have been used for subdividing axis aligned bounding boxes and constructing a bounding box hierarchy for the hull features (features of the polyhedron also appearing on its convex hull) and concavities of non-convex polyhedra [51]. Once penetration has been detected between the convex hulls of two polyhedra, a sweep and prune algorithm is applied to traverse the hierarchies down to the leaf level, where overlapping boxes indicate which faces may intersect, and exact contact points can be quickly determined.

In dense, cluttered environments, Oriented Bounding Boxes (OBB) perform better than axis aligned boxes or spheres, as they do fit more tightly to the objects and therefore less interferences between bounding volumes are reported. A hierarchical structure called OBB-Tree is used in [25] to represent polyhedra whose surfaces have been triangulated. 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.*

2.2 Basic interference tests

Convexity plays a very important role in the performance of interference detection algorithms, and it is therefore used as classification criterion in the description below.

Convex polyhedra As pointed out in [37], 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, as described in [17], 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 in the hierarchical representation is a tetrahedron. At each level of the hierarchy, vertices of the original polyhedron are added, such that they form an independent set (i.e. , are not adjacent) in the polyhedron corresponding to this hierarchical level, and the corresponding edge and face adjacency relationships are updated.

In fact, this algorithm computes the distance between two convex polyhedra. Likewise, all algorithms developed for distance computation can be adapted to detect interference. We refer the reader to Section 3.2.

One convex and one one non-convex An algorithm for computing the intersection between a convex and a non-convex polyhedron is described in [45]. A by-product of intersection *computation* is interference *detection*. Let $\overline{P}$ and $\overline{Q}$ be the surface of P (convex, n edges) and of Q (possibly non convex, m edges), respectively. The algorithm needs to solve the *support problem*, that is, to determine at least one point of each connected component of $\overline{P} \cap \overline{Q}$ (this set of points will be called S). The methods for interference detection between convex polyhedra and linear subspaces developed by Dobkin and Kirkpatrick [16] are used for determining the intersections of $\overline{P}$ with edges and faces of Q: a hierarchical representation is used for P, so that the intersection between a line l, supporting an edge of Q, and $\overline{P}$ is computed in time $O(\log n)$, and a point in $h \cap \overline{P}$, where h is a plane supporting a face of Q, can also be computed in time $O(\log n)$. Therefore, an algorithm can be constructed that solves the support problem in $O(m \log n)$. The next step consists in determining $C = \overline{P} \cap \overline{Q}$, by taking points of S, which are intersections between a face and an edge, and determining the intersections between the face and the two faces which are adjacent to the edge. Finally, the segments of edges of P and Q which are *inside* the intersection have to be determined. Figure 4 illustrates the main steps of the strategy. The overall complexity is $O((n+m+s)\log(n+m+s))$, where s is the number of edges in the intersection.

Non-convex polyhedra: Decomposition into convex parts It is possible to extend the usage of the above algorithms to non-convex polyhedra just by decomposing these polyhedra into convex entities. Typically, decomposition

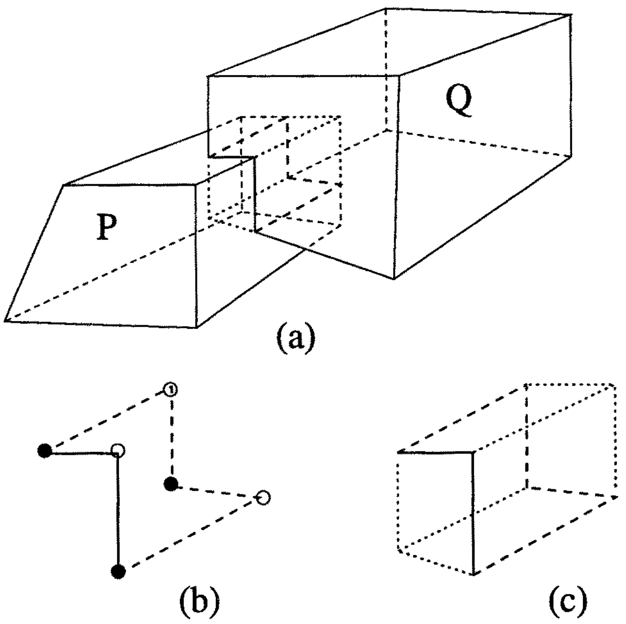


Fig. 4. (a) *Intersection computation (and -implicitly- interference detection) between two polyhedra, one of which is allowed to be non-convex (here, both have been depicted as convex for clarity).* (b) *Solving the support problem, the set of black (intersections of edges of Q and $\overline{P}$) and white (intersections of edges of P and $\overline{Q}$) points are obtained. Each pair of adjacent faces to these edges is intersected with the face of the other polyhedron that intersects this edge.* (c) *The segments of edges of one polyhedron inside the other one (and vice-versa) are finally computed (dotted lines).*

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 [3]. On the other hand, algorithms such as that in [13] can always partition a polytope of n vertices into at most $O(n^2)$ convex entities.

Consider two polyhedra. Discarding the case in which one is fully inside the other, they intersect if their surfaces 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.

Although interference detection becomes quite simple when faces are decomposed into convex polygons, and easy to implement, as explained below, the sequence of reductions used implies that the final complexity is $O(nm)$.

This reduction of the problem to detect edges piercing faces, formulated using the idea of *predicates* associated with the *basic contacts*, was introduced in [12]. The concept of basic contacts was introduced in [40], and its name derives from the fact that all other contacts can be expressed as a combination of them.

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). Although in [40] and in [18] two different contacts between vertices and faces were considered, depending on whether they belong to the mobile polyhedron or the obstacle, avoiding to make this distinction greatly simplifies the presentation.

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, as we will describe next.

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 leads to the same results in what follows.

According to Fig. 5(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. 5(b), predicate $\mathbf{B}_{E_i,E_j}$, associated with a basic contact of Type-B, is defined as true when

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

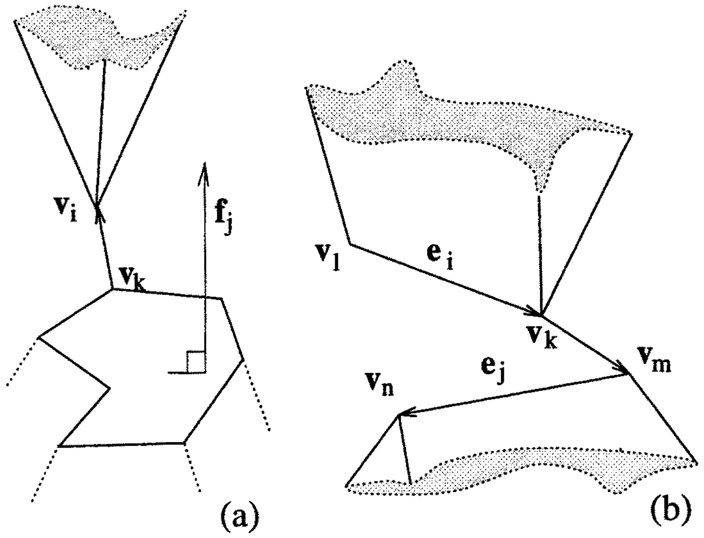


Fig. 5. *Geometric elements involved in the definition of the predicates associated with Type-A* (a) *and Type-B* (b) *basic contacts.*

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

It can be checked that if one of the following predicates

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

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

is true (see Fig. 6), then edge E_i intersects face F_j, provided that its edges (E_k) are traversed counter-clockwise.

Non-convex polyhedra: Direct approach If faces are not decomposed into convex polygons, two simple steps can be followed 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 if the intersection point 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, the intersection does exists (odd-parity rule). Note that the

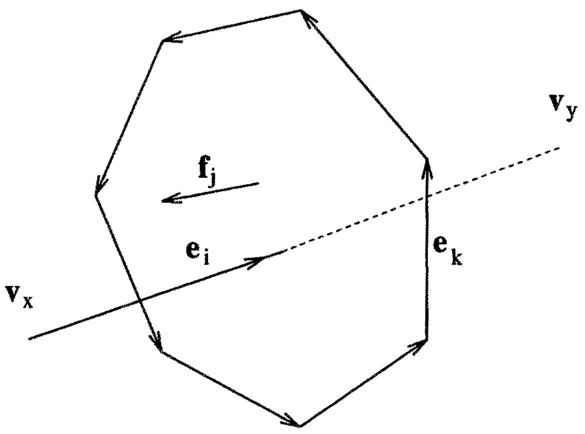


Fig. 6. *Basic edge-face intersection test (convex faces).*

latter step corresponds directly to solving a ***point-in-polygon*** problem, for which several alternatives, different from that of shooting a ray, have been proposed [28].

This was the approach adopted in [7] almost twenty years ago. Although the final complexity of the algorithm is clearly $O(nm)$, it is still the solution adopted in most implementations. Note that the only subquadratic algorithm developed so far [49] lacks practical interest because of the high time constants involved.

A simpler approach is to reduce the problem to computing the signs of some determinants [58], as it has been done for many other problems within the field of Computational Geometry [2],

Consider a face from one polyhedron, defined by the ordered sequence of the 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. 7, $s := \text{sign}\,|\mathbf{h}\ \mathbf{t}\ \mathbf{v}\ \mathbf{p}_i|$. Then, if $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ are on opposite sides, s should have a different sign from that of $|\mathbf{h}\ \mathbf{t}\ \mathbf{v}\ \mathbf{p}_{i+1}|$.

It can be checked 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.

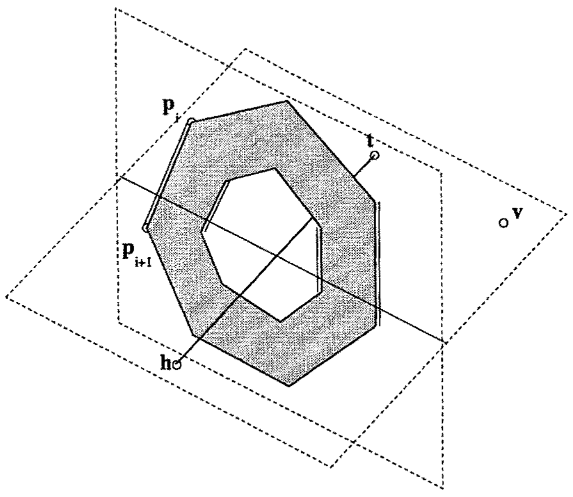


Fig. 7. *Basic edge-face intersection test (general faces).*

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 [7].

It is also interesting to point out that, if the arbitrary point **v** corresponds to a point on one of the two faces in which the edge lies, different from its endpoints, the above approach is a generalization of Canny's predicates, by simply noting that they can also be expressed in terms of signs of determinants involving vertex locations [58].

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 determinants involved 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 [57] through a formulation of the problem in terms of oriented matroids.

3 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 intrincate versions require finding the time and features involved in the first collision, or even the time intervals over which objects would be intersecting if they were adhering to the predefined motions. 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 quasi-linear.

The four main approaches that have been proposed to deal with the different instances of the collision detection problem are described in Section 3.1. After this description, it becomes clear that tests for static interference lie at the base of most approaches. However, the efficiency of a basic interference test does not guarantee that a collision detection algorithm based on it is in turn efficient. The other key factor is the number of times that this test is applied. Therefore, it is important to restrict the application of the interference test to those instants and object parts at which a collision can truly occur. Section 3.2 reviews the different strategies for time and space bounding that have been developed, among them distance computation, orientation-based pruning criteria, and prioritizing collision pairs.

3.1 Four main approaches

Collision detection algorithms can be grouped into four approaches: multiple interference detection, swept volume interference, extrusion in 4D space, and trajectory parameterization. As we will see, some approaches are linked to a particular object representation scheme (e.g. , extrusion is particularly suited to a CSG representation), while others do not.

Multiple interference detection The simplest way to tackle collision detection consists in sampling the trajectories followed by the objects and repeatedly applying a static interference test. This is called the multiple interference detection approach.

The way sampling is performed is crucial for the success of the approach. A too coarse sampling may lead to accepting a trajectory as safe when it actually leads to collision (see Fig. 8), 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 that relating a lower bound on the distance between objects to an upper bound on their relative velocities [10,15].

More sophisticated strategies take not only distance into account, but also directional information. One such strategy [23] requires computing the closest

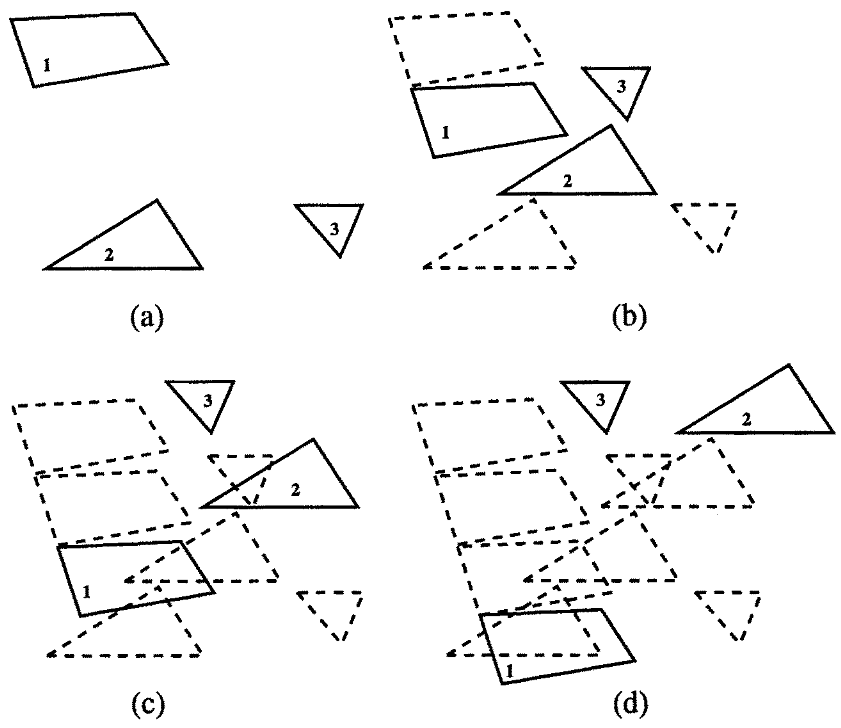


Fig. 8. *Multiple interference detection approach. As the time step is too large, the collision between the polygons 1 and 2, which takes place between instants* (b) *and* (c)*, is missed. At instant* (d) *polygons 1 and 2 have attained their final positions, whereas polygon 3 had already attained it between instants* (b) *and* (c)*. The polygons and their trajectories are the same as those in the next three figures.*

points from the objects 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. 9).

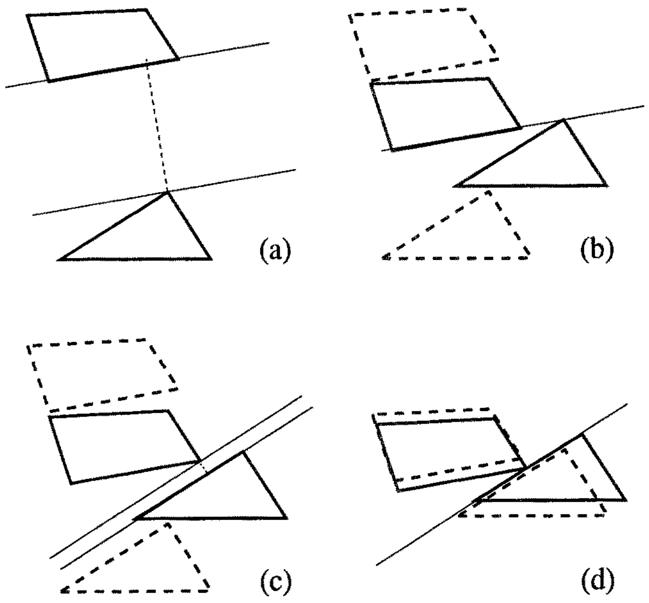


Fig. 9. *Adaptive time sampling. Starting position is depicted in* (a), *where the closest points and the line joining them are computed. The projections of the objects on this line meet at instant* (b), *which is taken as the next time sample. At this instant, the new closest points are computed* (c), *and the next time sample, where the polygons do actually collide, is determined in the same way* (d).

Since the closest points between two objects lie always in their boundaries, it is usual practice to resort to boundary representations (*B-rep*) when following a multiple interference detection approach. However, 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.

Swept volume interference Given an object and a description of its motion over a time period, the volume containing all the points occupied by the object at some time instant 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 (see Fig. 10).

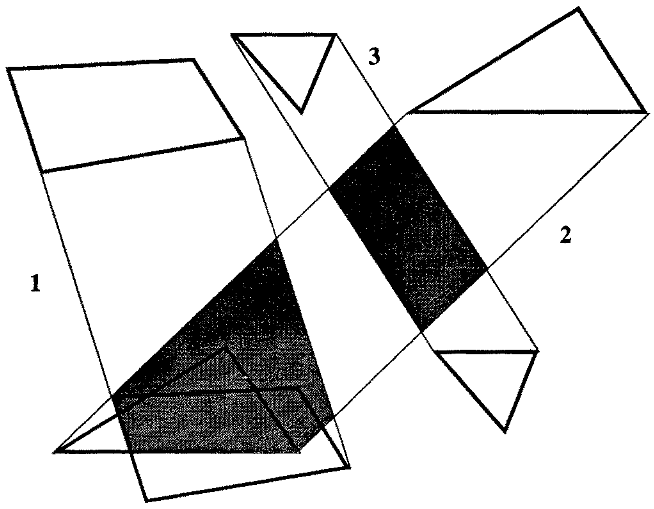


Fig. 10. *Swept volume interference. Polygons 1 and 2 collide, and their swept areas interfere. However, interference exists between the swept areas of polygons 2 and 3, but they do not actually collide.*

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. This can be computationally very costly.

The generation of the swept volume per se is also computationally expensive. This is the reason why most works in this area 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 [19].

The union of the convex approximations for the several trajectory pieces constitutes a much finer approximation to the real swept volume than the initial global approximation for the entire trajectory. For convex objects, Foisy and Hayward [19] have proved that the approximations obtained in the successive splittings of the trajectory converge to the real swept volume.

Simplifying alternatives consist in restricting the kind of shapes and trajectories to very simple ones [29], or creating implicitly the swept volume from the volumes swept out by the primitives of the *B-rep* [7].

Extrusion in 4D space Probably the collision detection approach most attractive from a theoretical viewpoint is that based on the extrusion operation [9]. Given an object and a description of its motion over a time period, the *extruded volume* is the spatio-temporal set of points representing the spatial occupancy of the object along its trajectory.

The intersection of two extruded volumes is a necessary and sufficient condition for the occurrence of a collision between the corresponding objects as they move along their respective trajectories (see Fig. 11). Therefore, this approach obviates a priori all the problems derived from sampling and from having to consider relative motions between pairs of objects. The problem that remains, however, is that of generating the volumes, which are 4D in this case.

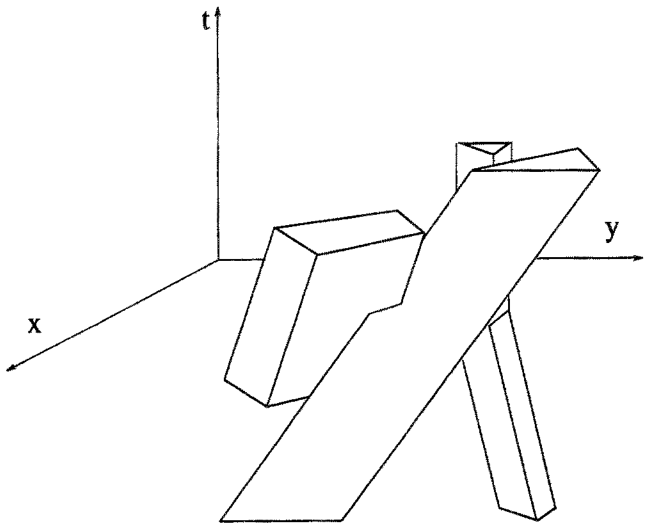


Fig. 11. *Interference between extruded volumes. Time is explicitly taken into account and therefore collision situations can be clearly identified (such as that of polygons 1 and 2). Note the change in the shape of the volume extruded by polygon 3, corresponding to the change in its velocity (it has stopped moving earlier than the other polygons).*

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 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. 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 [9].

Trajectory parameterization The collision instant can be analytically determined if the object trajectories are expressed as functions of a parameter (time) and the collision condition is formulated as a semialgebraic set involving the locations of object features (faces, edges and vertices). This requires to perform a change of variable in order to obtain an algebraic expression for rotation, instead of equations in terms of trascendent functions. By replacing those locations by the corresponding parameterized trajectories, a semialgebraic set in terms of a single variable (time) is obtained. Once this set is explicitly computed, the time instants at which objects establish and lose contact are known.

The trajectory parameterization approach has been followed in [12,33,54], where a polyhedra interference test is expressed as a combination of parameterized basic contact functions. These functions reflect 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.

3.2 Strategies for space and time bounding

The first three approaches described in the preceding section eventually require to apply a static interference test between either 3D volumes or 4D ones. However, even if a basic interference test is made very efficient, 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. 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 the objects. On the other hand, if the direction of motion is known, the search for possible collisions can be restrained to those object parts which may first come into contact. 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 discussed in the next subsections.

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 [59] the 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*. The distance between two spherical cones is determined in two steps: first, compute the direction where the minimum distance occurs, then compute the involved spheres (locate their centers on the axes of the spherical cones). The distance between a sphere and a spherical plane is found by projecting the sphere perpendicularly on the plane, and calculating the sphere on the plane that corresponds to this projection. In any case, once the spheres are located, the distance is easily found as the distance between their centers minus the sum of their radius.

Most distance computation algorithms have been developed for convex polyhedra. Some exploit specific features of the polyhedra and therefore cannot be used for other type of geometric models. Others, like the method explained in [24], can be used with spherical [26] or other non-polytopal surface descriptions [22].

There are two main streams in the way that the distance computation problem is treated, namely the *geometric* and the *optimization* approaches.

The geometric approach

The closest points of two polyhedra are obtained, under this approach, by expanding a hierarchical (incremental) representation in a given direction or by navigating along the boundaries of the polyhedra. The euclidean distance between these closest points is then computed. The methods differ in the way that the closest points are obtained:

- An adequate representation may justify the effort spent in obtaining it, as a preprocessing step is done only once, if it allows important computational savings in subsequent operations. This is the idea behind Dobkin and

Kirkpatrick's hierarchical polyhedral representation, already mentioned in Section 2. Using their representation leads to distance computation in optimal worst-case $O(\log n \log m)$ time [17]. Every step of the closest points search procedure corresponds to a level in the construction of the hierarchical representation. In the first step the closest points of two tetrahedra (the lowest level in the hierarchical representation), have to be determined, which is trivial. Now consider the direction of the segment that joins the closest points found at a given step. The two planes which are perpendicular to this direction and touch each polyhedron (in the hierarchy expanded so far) bound the zone where the next closest pair has to be searched for. This zone consists, for each polyhedron, in the intersection of the next hierarchy level polyhedron and the negative halfspace defined by the plane (the normal of the plane points towards the polyhedron expanded so far). Thus, it is either a 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 intersection simplices. Therefore, every search step is restricted to at most two simplices. The number of steps is bounded by $\log n \cdot \log m$. Figure 12 may help understand this procedure.

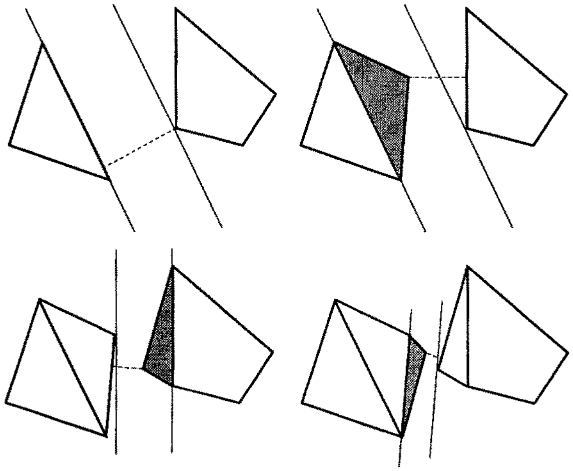


Fig. 12. *The hierarchical representation allows to build up and search only those parts of the polygons where the closest points can be found.*

- The Minkowski difference $M_{P,Q} = \{p - q | p \in P, q \in Q\}$ of two polytopes P and Q has been used in distance computation algorithms, since the distance between two polytopes is equal to the distance of their Minkowski difference to the origin (Fig. 13). This result is proved by Cameron and Culley (1986), and they provide also a procedure for computing $M_{P,Q}$, as well as the *minimum translational distance* (the minimum translation to be applied to one of the polyhedra in order to attain a situation where both polyhedra just touch). If $M_{P,Q}$ contains the origin of coordinates, the polyhedra are intersecting, and the minimum translational distance is negative.
- Efficiency is greatly increased in the procedure described in [24]. Complexity of $M_{P,Q}$ is, in general, quadratic, and therefore an algorithm that avoids generating the whole Minkowski difference would be desirable. Here, a directed sequence of subsets of the Minkowski difference polyhedron is generated, converging to a subset that contains the point that is closest to the origin. The convex hull of a subset of the vertices of the Minkowski difference is taken, and vertices are added which lie in the direction of interest, closer to the origin. At the same time non-relevant vertices are deleted, so that the search of the closest point to the origin is always done on a simplex, as can be seen in Fig. 14. 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.
- If a given point of a polyhedron is the closest one to a given feature (a vertex, an edge, or a face) of another polyhedron, it must be contained in the *Voronoi region* of this feature. The first step in this direction was done in [46] for rectangular boxes, but it was formalized and extended to any convex polyhedra in [37]. In their *incremental distance algorithm*, two arbitrary features are selected and the closest points that belong to them are obtained. In order to be actually the closest points of both polyhedra, these points have to belong to the Voronoi region of the other feature. If not, each point has to be closer to another neighboring feature, which is selected, and these steps are repeated until the condition of point-in-Voronoi-region-inclusion is met. The Voronoi regions for the three kinds of features are characterized in the mentioned reference (see Fig. 15).

  In their work, 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 considered for initialization for every step. Simple preprocessing of the polyhedra, so that every feature has a constant number of neighboring features,

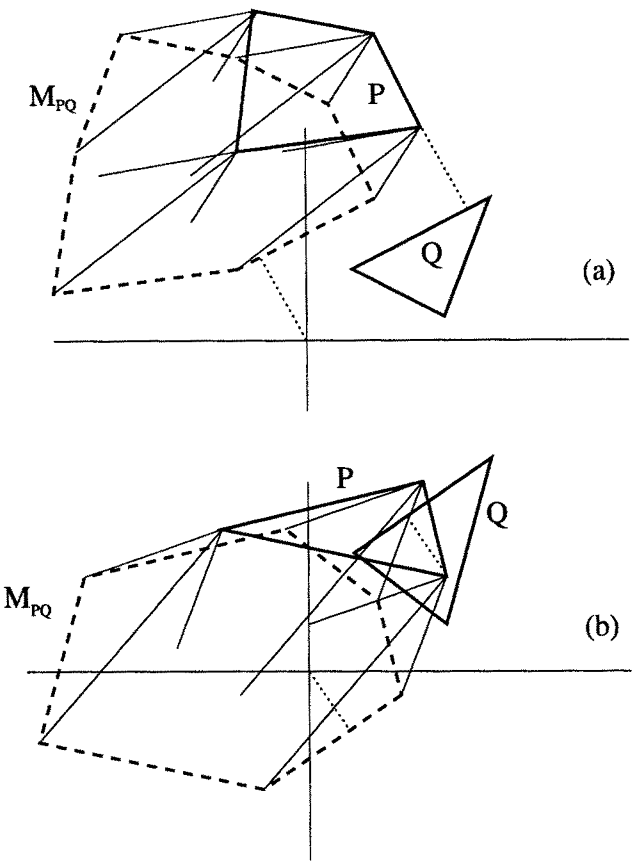


Fig. 13. (a) *The distance between the polygons is the same as the distance from the origin to their Minkowski difference.* (b) *If the polygons are interfering, the origin will be contained in the interior of their Minkowski difference. A hint is given for the construction of the Minkowski difference as the convex hulls of the points resulting from the subtraction of the vertices of Q from the vertices of P (thin lines).*

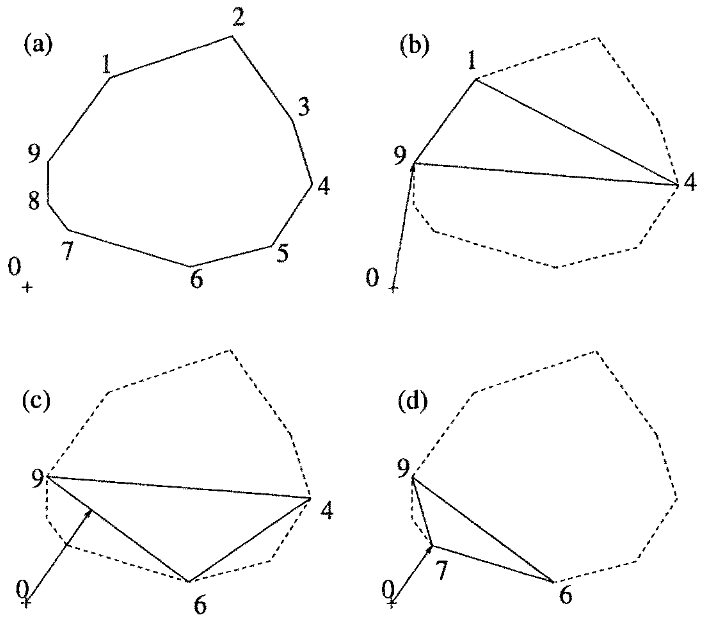


Fig. 14. *The closest point of the original Minkowski polygon* (a) *to the origin (0) has to be determined. The first simplex* (b) *has been chosen arbitrarily. A subset of vertices, whose convex hull contains the closest point of the simplex to the origin, is taken (4 and 9, although in this case also 1 and 9 could have been chosen), and a new vertex is selected* (c). *Note that this new vertex, 6, has the minimum projection onto the direction to the closest point found in* (b). *At the same time, the non-relevant vertex 1 is deleted. The direction to the closest point of the simplex in* (c) *is computed, and in the next step the closest point of the polygon to the origin (7 in this case) is determined* (d).

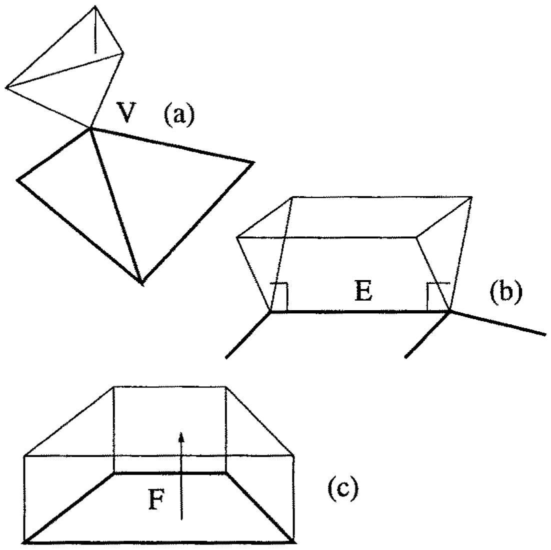


Fig. 15. *Voronoi regions of a vertex* (a), *an edge* (b), *and a face* (c).

allows the distance computation algorithm to run in expected constant time, once initialized (the global initialization step is typically linear in the total number of features).

To overcome the difficulty associated to the basic assumption that the two polyhedra have to be separated (if the objects actually penetrate each other, the algorithm goes into a cyclic loop), some authors have extended the space partition to the interior of the object, defining *pseudo*-Voronoi regions whose boundaries are faces determined by the centroid of the object and its edges (or the edges of its convex hull) [14,51,38]. These pseudo-Voronoi regions are only used to determine if the objects interpenetrate or not.

The optimization approach

Distance is viewed as a quadratic function to be minimized, under linear constraints due to the convexity of the polyhedra.

- The minimization of the non-linear function $f(p,q) = \|p-q\|^2/2$ subject to the linear constraints $\langle p, n_i^P \rangle \leq d_i^P, i = 1, \cdots, k^P$ and $\langle q, n_j^Q \rangle \leq d_j^Q, j = 1, \cdots, k^Q$ (these constraints mean that $p \in P$ and $q \in Q$, where the polyhedra P and Q are described as intersections of halfspaces) is solved in [5] by means of a gradient projection algorithm. At each step, the active constraints are determined (those where equality holds, with certain tolerance) and Kuhn-Tucker conditions are used to test if 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 perform a simplex minimization subalgorithm along the direction given by the centroids of the polyhedra, or by considering the intersection points of the polyhedra boundary and the segment that joins the centroids.

- In applying Rosen's gradient projection method as Bobrow did, a convergence problem may occur, as stated in [62]. 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 provided by these authors to this problem consists in considering as search direction for this polyhedron the projection of the search direction of the other polyhedron on the active constraints of the first one, instead of the zero vector, as shown in Fig. 16.

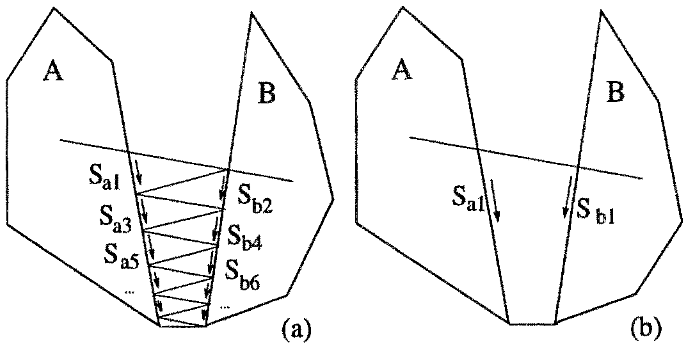


Fig. 16. *The zig-zag phenomenon* (a) *is avoided if the projection of S_{a1} on the active constraint of B is taken as the search direction S_{b1}* (b).

- Certain quadratic optimization problems can be solved in linear time, as shown in [44]. In [36] the distance computation problem between convex polyhedra is stated as described in Section 2, reducing it to a linear programming problem. It can be shown that the distance computation problem between non-intersecting convex polyhedra can be solved in $O(n)$ (n is the total number of vertices) by using a quadratic programming algorithm.

The linear constraints are here formulated in terms of the convex hull of the vertices of each polyhedron. In [53] the complexity associated with the intersection intensity computation between two polyhedra is also discussed.

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 2. Unfortunately, this solution may be inefficient, because of the complexity increment associated with the convex decomposition step. Moreover, 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.

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

- 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 [61]. The basic idea consists in 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. 17. On the average, half of the faces of the two polyhedra are eliminated in this way. An algorithm that performs culling is described in the above reference.
- The incremental minimum distance realization technique [36] has already been mentioned in Section 3.2. At a given instant, the boundary elements that realize the minimum distance must be close to those realizing it at the previous instant, 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.
- A third possibility to avoid having to perform unnecesary intersection tests arises in the context of convex polyhedra where only translational motions are allowed. The *applicability constraints* [18] permit detecting those vertex-face and edge-edge pairings which can really come into contact (Fig. 18). The vertex-face applicability condition expresses the fact that a vertex can touch a face only if every adjacent edge projects positively on the

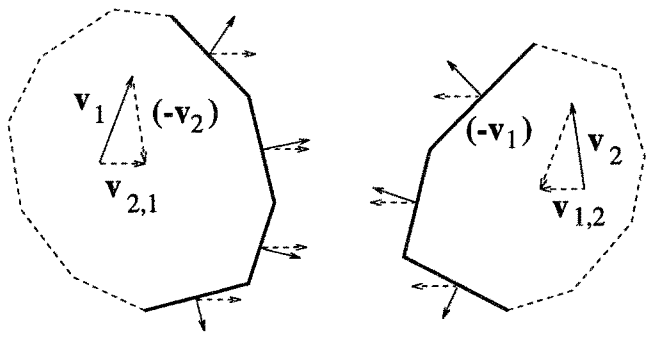


Fig. 17. *Only the faces (shown as heavy lines) whose normals have positive projections on the relative motion vectors ($v_{2,1}$ and $v_{1,2}$) need to be considered.*

face's normal (taking the vertex as origin of every edge interpreted as a vector). Two edges can touch only if there exist a separating plane between their respective *wedges*, as formally stated in the edge-edge applicability condition.

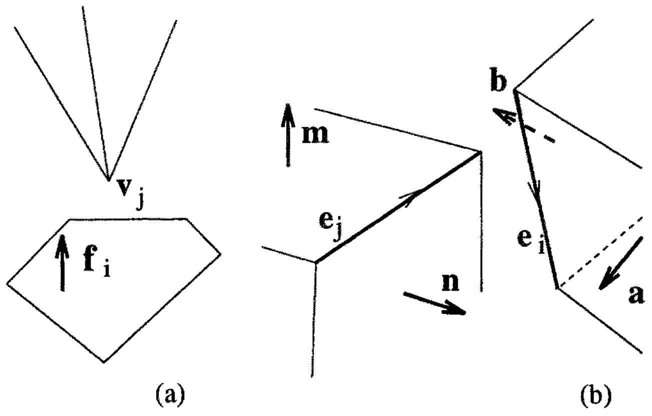


Fig. 18. (a) *An applicable vertex (V_j) - face (F_i) pairing.* (b) *Edges E_i and E_j are also applicable.*

The applicability constraints may be used as a preprocessing step in a collision detection scheme based on performing edge-face intersection tests. In general, if the contact between a vertex of a convex polyhedron and a face of another polyhedron is applicable, only one of the edges which are adjacent to the vertex have to be considered for intersection with this face to report collision. Any other edge-face test with this face can be cut off. In a similar way one can bound the number of edges to be considered with respect to a given face resulting from edge-edge applicability constraints. In [32] an efficient algorithm for geometric pruning based on applicability constraints for convex polyhedra is described. Experimental results show that, by using this pruning technique, collision detection based on the edge-face intersection test has an expected $O(n)$ complexity, where n is the total number of edges, and the constant of linearity is close to 1. The algorithm is based on a face orientation graph representation, where face adjacency relations are explicitly depicted. The authors are currently working on extensions of the algorithm to non-convex polyhedra.

Prioritizing collision pairs The algorithms that try to avoid having to test for collision every possible pairing between solids in a given workspace are only useful if there is a large number of solids that may collide. Candidates for collision checking are prioritized in order to test only those pairs which are more likely to collide. The first criterion one may consider is distance, but it is not enough if the relative velocities are not taken into account, as pointed out in [20]. These authors introduce the concept of *awareness* or imminence of a collision. The shortest possible time at which a collision may occur is computed, considering mutual distance, instantaneous relative velocity, and velocity and acceleration bounds. This calculation is initially done for every pair, and afterwards the updating is done more frequently for those pairs whose awareness is larger. According to their value of awareness, the pairs are partitioned into equivalence classes whose collision imminence is similar. A binary partition scheme is used, where the cardinality of each class (called "bucket") is an increasing power of 2, and the value of the measure of awareness for all elements of a given class is greater than that for any other element in a lower bucket (of greater cardinality). At every time step, only one pair within each bucket is updated. Since the higher the bucket, the less pairs it contains, higher buckets are updated more frequently than lower ones. As their measures of awareness change, pairs can percolate from bucket to bucket.

In [47] a heap is used to store object pairs and soonest possible collision times, so that the pair on the top is the nearest to collide. This soonest collision time is computed from the distance between the closest points of the objects, current velocities and accelerations, and acceleration bounds assuming a ballistic trajectory for the objects. At each time step, integration of the dynamic state is done up to the time of collision 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 frame period 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 is the ratio of largest to smallest box size), as shown in [48].

The same idea is followed in [35,14,51,38], where the concept of geometric and temporal coherence is emphasized, not only to speed up pairwise intersection detection (as done in [36] and whose algorithm is also used here) but also to perform less of these pairwise tests. If time steps (frames) are small enough, the position and orientation of the objects undergo only small changes, and it has already been mentioned how this fact can be used to keep track for the closest feature pair of two convex polyhedra. But it also means that there will be little changes in the position of the bounding boxes[1], and, of course, in the sequence of intervals that these bounding boxes project onto the coordinate axes, and which overlap (in the three axes) if and only if the corresponding bounding boxes intersect. Interval sorting techniques exist that take into account the sorted lists of interval endpoints in the previous frame, and allow to lower the effort to determine the projection intervals overlap to expected linear instead of $O(n \log n)$ (for n boxes). The computational cost of keeping track of changes in overlap status of interval pairs, following this line, is $O(n+s)$ (where s is the number of pairwise overlaps).

The so called *sweep algorithms* in [60] are along the same line: at a given instant, a plane is swept through the scene and only pairs of objects simultaneously intersecting the plane are tested for possible interference, thus avoiding to test every pair. The algorithm mentioned in this reference due to [30] does not find all intersections, although it reports at least one intersection if any exists, in $O(n \log^2 n)$ time between n spheres.

It is also possible to use such a sweep algorithm in 2D for bounding collision pairs in 3D, as done in [31]. 4D hyper-trapezoids are used to bound the object during 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 succesive projections, to a 2D segment intersection detection problem. The 2D sweep algorithm is described in [4] 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%).

[1] Two kinds of axis aligned bounding boxes are used in [14], fixed bounding cubes and dynamically rectangular boxes; for the latter, object orientation changes translate into changes in the dimensions of the bounding box.

4 Collision detection in motion planning

The goal of motion planning is to generate a collision-free path for a robot. Thus, collision-free trajectory planners must be able to perform some kind of geometric reasoning concerning collision detection between the robot and the obstacles [5,12]. In the generation of the path from the initial to the final robot configuration other criteria than mere collision avoidance may intervene, in order to optimize the resulting path in terms of its length, distances to obstacles, or orientation changes. Not to speak about extensions of the basic motion planning problem, that include uncertainty, kinematic constraints, or movable objects [34].

Depending on whether the static interference or collision detection tests are performed in a preprocessing step or during the path planning process, three kinds of planners can be distinguished: global, incremental, and local planners.

4.1 Global planners

In general, the configuration of a robot is given by a set of parameters, or degrees of freedom, that determine its location and orientation. The space defined by the ranges of allowed values for these parameters is usually called C-space.

An obstacle in C-space (C-obstacle, for short) is defined as the connected set of configurations where a given mobile object intersects with an obstacle in workspace. C-obstacles can be interpreted as the intersection of halfspaces bounded by C-surfaces, each C-surface being associated with a basic contact (see Section 2.2).

It can be shown that when working with polyhedra (and vertex, edge and face locations are expressed in terms of the degrees of freedom of the moving polyhedron), expressions (1) and (2) in Section 2.2 lead to the above-mentioned halfspaces, and using the predicate formalism in expression (3) a proper description of the C-obstacles can be obtained.

The collection of all C-obstacles constitutes the C-obstacle region. Some properties of the C-obstacles concerning compactness, connectedness and regularity are shown in [34]. C-obstacle generation can be viewed as a further generalization of the static interference and collision detection problems: here objects are not tested for interference at a particular configuration nor even along a given parameterized trajectory, but rather at all possible configurations in the workspace. Thus, once the C-obstacles are obtained, all information concerning interferences is captured.

Global planners construct a complete representation of the connectivity of free space (the complementary of the C-obstacle region) for their planning purposes. Several techniques have been devised to this end, depending on the degrees of freedom of the robot as well as on its shape and the shape of the obstacles. Nevertheless, they are only of practical interest in low-dimensional configuration spaces. Pioneer work in this direction was done in [42,40] for polytopal environments. As a result of applying these techniques, a graph-based representation of free space is obtained: a roadmap or the connectivity graph of a cell decomposition. Afterwards a graph search algorithm can be applied in order to find the path that connects the initial and the final configurations.

In some simple cases, the configuration of a robot can be expressed in terms of the workspace coordinates of a given robot's point: for example, if the robot is a sphere (a disc in 2D) this reference point will be its center. The C-obstacles are trivially obtained from the obstacles in the workspace by performing an isotropic growth by the radius of the robot. Another simple case consists in a polytopal robot translating amidst polytopal obstacles. Any vertex of the robot can be taken as reference point. In this particular case, C-obstacles can also be interpreted as the Minkowski difference between the obstacle and the robot at a fixed orientation (as already mentioned in the context of distance computation in Section 3.2). This fact can be used for constructing the C-obstacle itself. This alternative representation is obviously related to the general predicate-based one, in the sense that the differences between the vertices corresponding to the features related to basic contacts are vertices of the C-obstacle. Both representations have been developed for convex polytopes. Non-convex obstacles can be treated in the same way by representing them as overlapping convex parts.

When the robot polytope is allowed to rotate, the computation of the C-obstacles becomes much more difficult. Although an approximate solution can be readily obtained by sampling the involved rotations, in general C-obstacles can only be accurately described using the aforementioned predicates, which can be formally interpreted as semialgebraic sets (see [11] for more details). Note that when all degrees of freedom but one are sampled, the problem becomes one of detecting intervals of interference, as many times as needed, depending on the sampling rate. This technique is used with up to three degrees of freedom in [41].

4.2 Incremental planners

While global path planners generate a detailed description of the connectivity of the whole free space, incremental path planners avoid this costly computation by obtaining this description in an incremental fashion. In this case, the construction of the free-space representation is carried out simultaneously with the path planning process. A paradigmatic example of this strategy can be found in [21], where a restricted visibility graph in C-space is built up iteratively. This subset of the whole visibility graph is granted to contain the optimal path. It is constructed by determining which C-obstacles intersect the segments of the shortest path found so far (a straight line joining the initial and final positions at the first step), and rearranging the visibility graph with these new C-obstacles.

Randomized path planning methods might work in a similar way: points are randomly generated and those lying in free space are retained. Then, attempts are made to link these points by means of collision-free line segments. In this way, a representation of free space is gradually built up by locally testing for collisions, while generating a path from the initial to the final configurations. The same applies to those techniques that combine a potential field approach with randomization to escape from local minima. More details on this kind of algorithms can be found in Chapter 5.

4.3 Local planners

Local planners use collision detection as a subroutine whose output is used on-line to guide the search of a collision-free trajectory. The main difference with respect to incremental strategies is that local methods perform path planning by applying motion operators that act locally. In [18] these operators are used for sliding on C-surfaces and along their intersections without computing an explicit representation of free-space. They also allow to jump in free-space between obstacles. In any case, each time a C-surface is traversed, an interference test is performed to ensure that the motion is collision-free. These operators are the building blocks of more sophisticated *local experts*, which are strategies for deciding which trajectory to follow, based on the local geometry as well as on the history of the current planning process. This combination of motion in free space with motion in contact (or up to a safety distance from the obstacles) is used by other local motion planners. This is the case of the algorithms developed for planar articulated and 3D cartesian robot arms in [43,55]. The intersection points of the obstacles with the *main line* joining the initial and final positions are found, and motions along the obstacle boundaries between these points are computed.

Some local planners, as well as some incremental ones, can be applied in a recursive way: starting from an initial guess for a path between the initial and the final position, intermediate points are determined as collisions are detected, and the algorithm is recursively applied to the resulting path segments until a collision-free trajectory is detected or the conclusion is drawn that no such path exists.

While global path planners are always complete, that is, they are able to find a solution if one exists, those based on local techniques only ensure completeness at a resolution level. In [18] a partition of C-space based on *neighborhoods* is adopted, which are marked as visited if they are traversed by a trajectory generated during the path planning process. As a consequence, if neighborhoods are made arbitrarily small, the algorithm becomes arbitrarily slow.

5 Conclusions

The different approaches to collision detection lie within two main categories: algebraic and geometric. The first try to solve equations that describe collision situations. These equations are expressed in terms of one parameter which is time or a variable related to time, and collision instants are determined. The trajectory parameterization approach corresponds to this strategy.

The geometric approaches compute geometric entities where time is treated as one more dimension, and try to determine intersections between them using methods developed within Computational Geometry. The most general approach is spatio-temporal volume intersection. However, no techniques exist for solving this problem directly, except for simple particular cases. The other two approaches can be viewed as particular techniques that simplify the resolution of the problem: the multiple interference detection approach applies sampling, whereas the swept volume interference approach uses projection. The drawbacks of these simplifying techniques have already been mentioned: sampling is complete only up to a given resolution, and projections may lead to report false collisions. Combinations of these techniques may allow to avoid these drawbacks, as in the adaptive sampling approach.

This perspective permits to formulate extensions for further work in a straightforward way: simplifying techniques have always been formulated considering time as a privileged dimension. Time is discretized by sampling or obviated by projection, but both techniques may also apply as well to the other dimensions or to combinations of them. To identify the classes of situations where sampling or projecting along other dimensions will ease the computation of collisions is more than an interesting theoretical exercise and may open new promising trends in collision detection algorithms.

Algebraic methods can also be viewed as a simplification of the general spatio-temporal problem formulation, as a projection on the time coordinate axis. Other dimensions instead of time could be used as parameters of the contact equations. However, the degree of the equations cannot be lowered in this way, and thus the efforts in looking for more efficient algorithms have to point in another direction, namely reducing the number of equations to be considered. This can be done by applying the complexity reduction techniques already mentioned in Section 3.2. In particular, orientation-based pruning may be applied to subdivide the trajectory into intervals where the same boundary primitives have to be considered for possible intersection, reducing drastically the number of contact equations to be considered within each interval.

In the line of developing complexity reduction techniques for interference detection we have centered our contribution to the PROMotion Project. Little work had previously been done on algorithms that deal directly with non-convex polyhedra, without decomposing them into convex parts. We have developed one such algorithm, based on a boolean combination of signs of vertex determinants [58]. Thus, neither line-plane intersections, nor ficticious edges and faces arising from a decomposition are required. Only the signs of determinants, for which there exist very efficient and robust algorithms, need to be computed. Moreover, we have developed a representation that captures the applicability relationships between the boundary features of two general polyhedra, that is, it allows to determine quickly which contacts can arise under translational motions [32]. In the case of non-convex polyhedra a large subset of all contacts that cannot take place for a given relative orientation are pruned off (all of them in the case of convex polyhedra). As these relationships hold over whole ranges of orientations, this technique can also be used to perform pruning along a trajectory that includes rotation[33], as mentioned above in the context of algebraic techniques for collision detection.

The speed-up of the basic interference and collision detection tests will necessarily improve the performance of motion planners, thus making the famous bottleneck a little bit wider.

References

1. N. Ahuja, R. T. Chien, R. Yen and N. Bridwell, "Interference detection and collision avoidance among three dimensional objects" in *I Annual National Conference on AI* (Stanford University) pp. 44–48 (Aug. , 1980).
2. F. Avnaim, "Evaluating signs of determinants using single-precision arithmetic" (Tech. Rep. INRIA 2306) (1994).
3. C. Bajaj and T. Dey, "Convex decomposition of polyhedra and robustness" in *SIAM J. Comput.* 21(2) pp. 339–364 (Apr. , 1992).
4. J. L. Bentley and T. A. Ottmann, "Algorithms for reporting and counting geometric intersections" in *IEEE Trans. Comput.* 28 (9) pp. 643–647 (Sept. , 1979).
5. J. E. Bobrow, "A direct optimization approach for obtaining the distance between convex polyhedra" in *Internat. J. Robotics Res.* 8 (3) pp. 65–76 (June, 1983).
6. S. Bonner and R. B. Kelley, "A representation scheme for rapid 3-D collision detection" in *Proceedings IEEE International Symposium on Intelligent Control* (Arlington (VA)) pp. 320–325 (Aug. , 1988).
7. J. W. Boyse, "Interference detection among solids and surfaces" in *Comm. ACM* 22 (1) pp. 3–9 (Jan. , 1979).
8. S. A. Cameron, "Efficient intersection tests for objects defined constructively" in *Internat. J. Robotics Res.* 8 pp. 3–25 (Feb. , 1989).
9. S. A. Cameron, "Collision detection by four-dimensional intersection testing" in *IEEE Trans. Robotics Automat.* 6 (3) pp. 291–302 (June, 1990).
10. S. A. Cameron, "A study of the clash detection problem in robotics" in *IEEE Proc. Int. Conf. Robotics Automat.* 1 (Saint Louis (MO)) pp. 488–493 (Mar. , 1985).
11. J. F. Canny, "The complexity of robot motion planning" (PhD Thesis, The MIT Press, Cambridge (MA)) (1988).
12. J. F. Canny, "Collision detection for moving polyhedra" in *IEEE Trans. Patt. Anal. Mach. Intell.* 8 (2) pp. 200–209 (Mar. , 1986).
13. B. Chazelle, "Convex partitions of polyhedra: A lower bound and a worst-case optimal algorithm" in *SIAM J. Comput.* 13 pp. 488–507 (1984).
14. J. D. Cohen, M. C. Lin, D. Manocha and M. K. Ponamgi, "I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments" in *Proceedings of ACM Int. 3D Graphics Conference* 1 pp. 189–196 (1995).
15. R. K. Culley and K. G. Kempf, "A collision detection algorithm based on velocity and distance bounds" in *IEEE Proc. Int. Conf. Robotics Automat.* 2 (San Francisco (CA)) pp. 1064–1069 (Apr. , 1986).
16. D. Dobkin and D. Kirkpatrick, "Fast detection of polyhedral intersections" in *Lect. Notes in Comp. Sci.* (Springer-Verlag, New York–Heidelberg–Berlin) (140) pp. 154–165 (1982).
17. D. Dobkin and D. Kirkpatrick, "Determining the separation of preprocessed polyhedra -a unified approach" in *ICALP-90, Lect. Notes in Comp. Sci.* (Springer-Verlag, New York–Heidelberg–Berlin) (443) pp. 400–413 (1990).
18. B. R. Donald, "Local and global techniques for motion planning" (Masters Thesis, Massachusetts Institute of Technology) (1984).

19. A. Foisy and V. Hayward, "A safe swept volume method for collision detection" in *The Sixth International Symposium of Robotics Research* (Pittsburgh (PE)) pp. 61–68 (Oct. , 1993).
20. A. Foisy, V. Hayward and S. Aubry, "The use of awareness in collision prediction" in *IEEE Proc. Int. Conf. Robotics Automat.* 1 (Cincinnati (OH)) pp. 338–343 (May, 1990).
21. L-C. Fu and D-Y. Liu, "An efficient algorithm for finding a collision-free path among polyhedral obstacles" in *J. Robotic Sys.* 7 (1) pp. 129–137 (Feb. , 1990).
22. E. G. Gilbert and C-P. Foo, "Computing the distance between general convex objects in three-dimensional space" in *IEEE Trans. Robotics Automat.* 6 (1) pp. 53–61 (Feb. , 1990).
23. E. G. Gilbert and S. M. Hong, "A new algorithm for detecting the collision of moving objects" in *IEEE Proc. Int. Conf. Robotics Automat.* 1 (Scottsdale (AR)) pp. 8–14 (May, 1989).
24. E. G. Gilbert, D. W. Johnson and S. Keerthi, "A fast procedure for computing the distance between complex objects in three dimensional space" in *IEEE J. Robotics Automat.* 4 (2) pp. 193–203 (Apr. , 1988).
25. S. Gottschalk, M. C. Lin and D. Manocha, "OBB-Tree: A Hierarchical Structure for Rapid Interference Detection" in *Proc. of ACM Siggraph'96* (1996).
26. G. J. Hamlin, R. B. Kelley and J. Tornero, "Efficient distance calculation using the spherically-extended polytope (S-tope) model" in *IEEE Proc. Int. Conf. Robotics Automat.* 3 (Nice (France)) pp. 2502–2507 (May, 1992).
27. V. Hayward, "Fast collision detection scheme by recursive decomposition of a manipulator workspace" in *IEEE Proc. Int. Conf. Robotics Automat.* 2 (San Francisco (CA)) pp. 1044–1049 (Apr. , 1986).
28. P. Heckbert (ed.), "Graphic Gems IV" (Academic Press, 1994).
29. M. Herman, "Fast, three-dimensional, collision-free motion planning" in *IEEE Proc. Int. Conf. Robotics Automat.* 2 pp. 1056–1063 (Apr. , 1986).
30. J. E. Hopcroft, J. T. Schwartz and M. Sharir, "Efficient detection of intersections among spheres" (Tech. Rep. 59, Dept. of Computer Science, Courant Inst. of Math. Sciences. , N.Y.University) (Feb. , 1983).
31. P. M. Hubbard, "Interactive collision detection" in *Proc. IEEE Symp. on Research Frontiers in Virtual Reality* 1 pp. 24–31 (Oct. , 1993).
32. P. Jiménez and C. Torras, "Speeding Up Interference Detection Between Polyhedra" in *IEEE Proc. Int. Conf. Robotics Automat.* pp. 1485–1492 (Minneapolis (MN) (Apr. , 1996).
33. P. Jiménez and C. Torras, "Collision detection: a geometric approach" in *Modelling and Planning for Sensor Based Intelligent Robot Systems* (H. Bunke, H. Noltemeier, T. Kanade (eds.)) World Scientific Pub. Co. (Nov. , 1995).
34. J-C. Latombe, "Robot Motion Planning" Kluwer Academic Publishers (SECS 0124, Boston/Dordrecht/London) (1991).
35. M. C. Lin, "Efficient Collision Detection for Animation and Robotics" (PhD Thesis, University of California, Berkeley) (1993).
36. M. C. Lin and J. F. Canny, "An efficient algorithm for incremental distance computation" (to appear in IEEE Trans. Robotics Automat.).

37. M. C. Lin and J. F. Canny, "A fast algorithm for incremental distance calculation" in *IEEE Proc. Int. Conf. Robotics Automat.* 2 (Sacramento (CA)) pp. 1008–1014 (Apr. , 1991).
38. M. C. Lin, D. Manocha and J. F. Canny, "Fast Contact Determination in Dynamic Environments" in *IEEE Proc. Int. Conf. Robotics Automat.* 1 (San Diego (CA)) pp. 602–608 (May, 1994).
39. Y-H. Liu, S. Arimoto and H. Noborio, "A new solid model HSM and its application to interference detection between moving objects" in *J. Robotic Sys.* 8 (1) pp. 39–54 (1991).
40. T. Lozano-Pérez, "Spatial planning: a configuration space approach" in *IEEE Trans. Comput.* 32 (2) pp. 108–120 (Feb. , 1983).
41. T. Lozano-Pérez, "A simple motion-planning algorithm for general robot manipulators" in *IEEE J. Robotics Automat.* 3 (3) pp. 224–238 (June, 1987).
42. T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles" in *Comm. ACM* 22 (10) pp. 560–570 (Oct. , 1979).
43. V. J. Lumelsky, "Effect of kinematics on motion planning for planar robot arms amidst unknown obstacles" in *IEEE J. Robotics Automat.* 3 (3) pp. 207–223 (June, 1987).
44. N. Megiddo and A. Tamir, "Linear time algorithms for some separable quadratic programming problems" in *Operations Research Letters* 13 (4) pp. 203–211 (1993).
45. K. Mehlhorn and K. Simon, "Intersecting two polyhedra one of which is convex" in *Fundamentals of Computation Theory 85, Lecture Notes in Computer Science* 199 pp. 534–542 (1985).
46. W. Meyer, "Distance between boxes: applications to collision detection and clipping" in *IEEE Proc. Int. Conf. Robotics Automat.* 1 (San Francisco (CA)) pp. 597–602 (Apr. , 1986).
47. B. Mirtich and J. F. Canny, "Impulse-based dynamic simulation" in *Tech. Rep.* CSD-94-815 (University of California) (1994).
48. M. Overmars, "Point location in fat subdivisions" in *Information Processing letters* 44 pp. 261–265 (1992).
49. M. Pellegrini, "Stabbing and ray shooting in 3-space" in *Proceedings of the 6thACM Symposium on Computational Geometry* pp. 177–186 (1990).
50. A. P. del-Pobil, M. A. Serna and J. Llovet, "A new representation for collision avoidance and detection" in *IEEE Proc. Int. Conf. Robotics Automat.* 1 (Nice (France)) pp. 246–251 (May, 1992).
51. M. K. Ponamgi, D. Manocha and M. C. Lin, "Incremental algorithms for collision detection between solid models" in *Proceedings of ACM/Siggraph Symposium on Solid Modelling* 1 pp. 293–304 (1995).
52. S. Quinlan, "Efficient Distance Computation between Non-Convex Objects " in *IEEE Proc. Int. Conf. Robotics Automat.* 4 (San Diego (CA)) pp. 3324–3329 (1994).
53. N. K. Sancheti and S. S. Keerthi, "Computation of certain measures of proximity between convex polytopes: a complexity viewpoint" in *IEEE Proc. Int. Conf. Robotics Automat.* 3 (Nice (France)) pp. 2508–2513 (May, 1992).

54. E. Schömer and C. Thiel, "Efficient collision detection for moving polyhedra" (Tech. Rep. MPI-1-94-147, Max Plank Inst. Informatik Saarbr.) (1995).
55. K. Sun and V. Lumelsky, "Path planning among unknown obstacles: the case of a three dimensional cartesian arm" in *IEEE Trans. Robotics Automat.* 8 (6) pp. 776–786 (Dec. , 1992).
56. W. C. Thibault and B. F. Naylor, "Set operations on polyhedra using binary space partitioning trees" in *ACM Computer Graphics* 21 (4) (July, 1987).
57. F. Thomas, "An approach to the movers problem that combines oriented matroid theory and algebraic geometry." in *IEEE Proc. Int. Conf. Robotics Automat.* 1 (Nagoya (J)) (May, 1995).
58. F. Thomas and C. Torras, "Interference detection between non-convex polyhedra revisited with a practical aim" in *IEEE Proc. Int. Conf. Robotics Automat.* 1 (San Diego (CA)) pp. 587–594 (May, 1994).
59. J. Tornero, J. Hamlin and R. B. Kelley, "Spherical-object representation and fast distance computation for robotic applications" in *IEEE Proc. Int. Conf. Robotics Automat.* 2 (Sacramento (CA)) pp. 1602–1608 (Apr. , 1991).
60. G. Turk, "Interactive collision detection for molecular graphics" (Master Thesis, University of North Carolina) (1989).
61. G. Vanecek, "Back-face culling applied to collision detection of polyhedra" (to appear in "Journal of Visualization and Computer Animation").
62. S. Zeghloul, P. Rambeaud and J. P. Lallemand, "A fast distance calculation between convex objects by optimization approach" in *IEEE Proc. Int. Conf. Robotics Automat.* 3 (Nice (France)) pp. 2520–2525 (May, 1992).