P. Jiménez
C. Torras

Institut de Robòtica i Informàtica Industrial
Llorens i Artigas 4-6, E-08028 Barcelona, Spain
pjimenez@iri.upc.es
ctorras@iri.upc.es

An Orientation-Based Pruning Tool to Speed Up Contact Determination between Translating Polyhedral Models

Abstract

Contact determination in terms of edge-face intersection tests permits handling nonconvex polyhedra directly, without decomposing them into convex entities, which saves the decomposition time and avoids having to deal with fictitious features but requires checking all possible pairings. However, by considering only translations and departing from a noninterfering situation, the number of pairings to be checked decreases drastically. The set of critical pairings can be determined efficiently using the spherical face orientation graph (SFOG), a representation developed by the authors. An algorithm to exploit the SFOG in convex settings provides controlled evidence of the pruning potential of this approach: the number of critical pairings grows linearly with the complexity of the polyhedra, instead of quadratically as the total number of pairings does. Experiments with a similar algorithm on nonconvex settings confirm the expected potential of the approach: for workpieces with many concavities moving in close proximity, the contact determination procedure presented in this paper performs one order of magnitude faster than RAPID, at the expense of a much higher preprocessing time.

KEY WORDS—multiple contacts, interference checking, collision detection, nonconvex polyhedra, sphere of orientations

1. Introduction

Most collision detection methods have been devised for complex graphic environments, where speed is the main priority. Such environments typically contain a large collection of objects, modeled as convex polyhedra or polygon soups, and the goal is to determine when and where interference may occur. To this end, many techniques to encapsulate possible collisions within time and space bounds have been developed (Jiménez, Thomas, and Torras 2001; Lin and Gottschalk 1998), such as enclosing boxes (Cohen et al. 1995; García-Alonso et al. 1994), object partitioning for CSG (constructive solid geometry) representations (Cameron 1991), space partitioning (García-Alonso et al. 1994; Thibault and Naylor 1987), hierarchies of bounding volumes (Gottschalk, Lin, and Manocha 1996; Hamlin, Kelley, and Tornero 1992; Hubbard 1993; Hudson et al. 1997; Klosowski et al. 1998; Martínez et al. 1998; van der Bergen 1997), space and time coherence (Lin and Canny 1991; Ponamgi, Manocha, and Lin 1997), four-dimensional space-time bounds (Hubbard 1995), and distance bounding (Gilbert, Johnson, and Keerthi 1988).

This paper deals with an instance of the collision detection problem that has received less attention, namely, exact contact determination between two translating nonconvex polyhedra that are in close proximity and can make contact over wide, possibly disconnected, portions of their boundaries. This situation arises in assembly design and planning within CAD/CAM systems (Thomas and Torras 1992). In a previous paper, we undertook the combinatorial search needed to generate optimal k-directional translational assembly sequences (Jiménez and Torras 2000), and here we address the complementary problem of contact determination for translational assembly. In this context, it is crucial to discard features in the two polyhedra that can never make contact. Back-face culling (Vanecek 1994) is an effective pruning strategy based on the direction of motion. Here, we propose a related strategy based on the relative orientation of the polyhedra.

The paper is structured as follows. For self-containment, Sections 2 and 3 review the items on which the present work is based: the edge-face intersection test, which permits a decomposition-free contact determination between nonconvex polyhedra (Thomas and Torras 1994), and the applicability conditions (Donald 1987), used here to reduce the search

for interfering features under arbitrary translational motion. Section 4 presents the spherical face orientation graph, a suitable representation for handling these conditions. This representation is exploited in Sections 5 and 6 to speed up contact determination in convex and nonconvex settings, respectively. Section 7 presents experimental results, and Section 8 draws some conclusions.

2. Edge-Face Intersection Test

The boundaries of two polyhedra interfere if and only if at least one edge of one polyhedron intersects a face of the other one. Most contact and interference detection procedures require faces to be convex or triangulated. Others, like the classical crossings algorithm (Boyse 1979), work on nonconvex faces but require auxiliary geometric constructs, such as shooting a ray from the point of intersection and counting the number of crossings with the boundary edges.

The edge-face intersection test (Thomas and Torras 1994) works on the vertices of nonconvex faces directly, without introducing any auxiliary entity. This test is a combination of the two basic contact predicates (vertex-face and edge-edge), which correspond to the truth value (the sign) of two types of functions, computed as 4×4 determinants of homogeneous vertex coordinates. A vertex-face (v-face) f predicate $\mathbf{A}_{v,f}$ is true if $A_{v,f} = \|v_i v_j v_k v\| > 0$, where $\{v_i, v_j, v_k\}$ is an ordered set of vertices of f, and an edge-edge predicate $\mathbf{B}_{e_m,e_n}$ is true if $B_{e_m,e_n} = \|v_i v_j v_k v_l\| > 0$, where $v_i = \partial^+ e_m$, $v_j = \partial^- e_m$, $v_k = \partial^+ e_n$, and $v_l = \partial^- e_n$ (∂^+ and ∂^- are the half-boundary operators—in this case, they refer to the endpoints of the edges).

These functions (and their associated predicates) express incidence relationships between the primitives involved. $A_{v,f} = 0$ means that vertex v lies on the plane that supports f, and the sign of this function, when different from zero, indicates on which of the two half-spaces defined by this plane v lies. $B_{e_m,e_n} = 0$ means that the lines supporting the edges are either touching or parallel; other relative positions and orientations are expressed through the sign of this function.

For an edge e to intersect a possibly nonconvex face f, two conditions must simultaneously hold:

- Each endpoint of the edge must lie in a different half-space of those defined by the plane supporting the face.
- The line supporting the edge must intersect the face.

The edge-face test checks these two conditions by evaluating the following composite predicate:

$$(\mathbf{A}_{\partial^+ e,f} \oplus \mathbf{A}_{\partial^- e,f}) \wedge \Bigg[\bigoplus_{e_f \in \partial f} (\mathbf{A}_{\partial^+ e_f,f_e} \oplus \mathbf{A}_{\partial^- e_f,f_e}) \wedge (\mathbf{A}_{\partial^- e_f,f_e} \oplus \mathbf{B}_{e,e_f}) \Bigg],$$

where $\oplus$ is the XOR exclusive OR operator ($(a \oplus b) = (a \wedge \overline{b}) \vee (\overline{a} \wedge b)$) and f_e refers to any plane containing edge e. To identify the right term with the second condition, note that the number of edges in the boundary of f that pierce one of the half-planes of f_e must be odd for the condition to hold. See Figure 1 and Thomas and Torras (1994) for further details. The predicate can be seen as an extension of Canny's (1987) disjunctive form to deal with nonconvex faces.

To test for interference between polyhedra, the above predicate has to be applied to the pairings consisting of edges of one polyhedron and faces of the other one. It suffices to determine one such pairing for which the predicate holds to report interference. But this search has a quadratic worst-case complexity: if no interference exists, all edge-face pairings have to be tested. Therefore, any technique that lowers the number of edge-face pairings to be considered will be welcome. This can be done if one assumes that the polyhedra are initially disjoint and translating, and that only the first edge-face pairing that may intersect needs to be reported.

3. Applicability Conditions and Their Use for Pruning Edge-Face Pairs

Consider two disjoint polyhedra. If any relative motion between them is allowed, many contacts between the features of both polyhedra are possible (in particular, when the polyhedra are convex, every contact is possible). But if only translations are allowed, only certain contacts can arise. These contacts are said to be applicable. In the case of convex polyhedra, necessary and sufficient conditions for vertex-face and edge-edge applicability have been stated (Donald 1987) (see Fig. 2). For a given relative orientation between two polyhedra,

- The contact between a vertex v of one polyhedron and a face f of the other is applicable if and only if $\forall v_i$ adjacent to v, $\langle v_i, f \rangle - \langle v, f \rangle \geq 0$.
- The contact between an edge e_m of one polyhedron and an edge e_n of another polyhedron is applicable if and only if $k_a \neq k_b$, where $k_a = \text{sign}(\langle T_1, f_p \rangle) = \text{sign}(\langle T_2, f_p \rangle)$, and $k_b = \text{sign}(\langle T_3, f_p \rangle) = \text{sign}(\langle T_4, f_p \rangle)$, with $f_p = e_m \times e_n$ (or the opposite direction, the choice is arbitrary) and $T_i = s_i \cdot (f_i \times e_m)$, f_i adjacent to e_m, $T_j = s_j \cdot (f_j \times e_n)$, f_j adjacent to e_n; $s_i, s_j \in \{+1, -1\}$ such that T_l is oriented toward the interior of face f_l.

If the polyhedra are initially disjoint and translating, the interference test in the preceding section needs to be applied only to the edge-face pairings obtained from the applicability conditions as follows:

- From the vertex-face contact, any edge stemming from the vertex should be paired with the face.

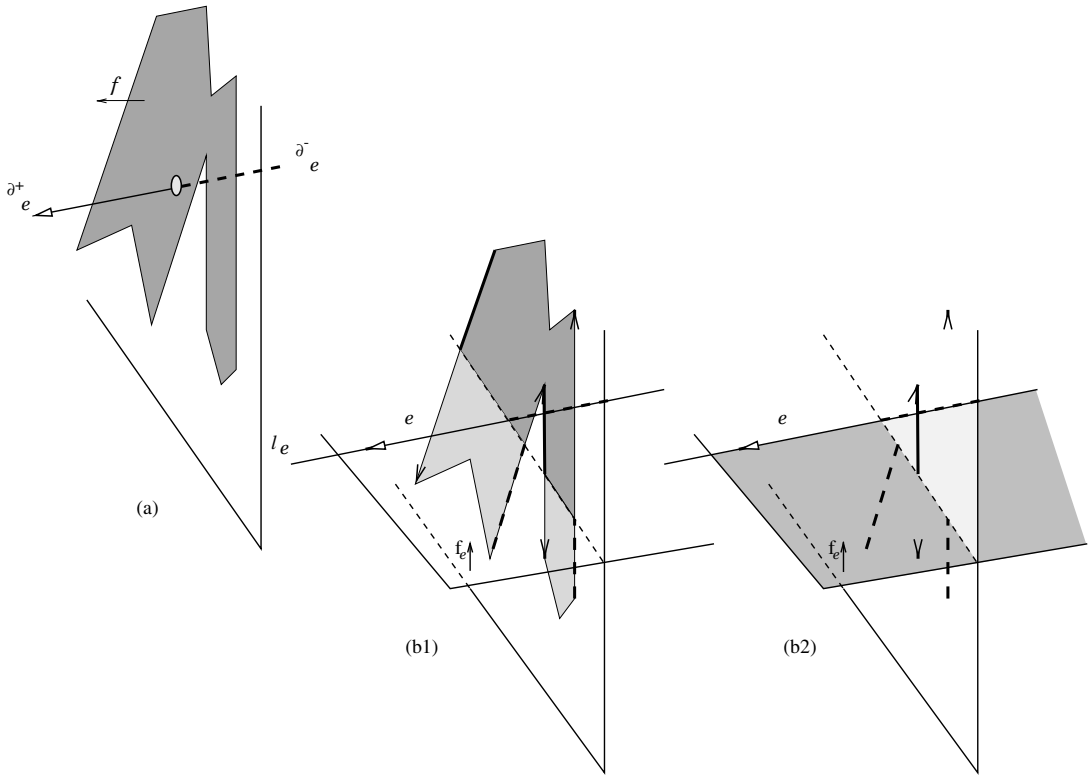

Fig. 1. (a) $\mathbf{A}_{\partial^+ e,f} \oplus \mathbf{A}_{\partial^- e,f}$ is true if and only if both endpoints of edge e lie on different sides of the supporting plane of face f. (b) Testing whether the line l_e supporting e cuts f is done by (b1) determining the set of boundary edges of f that intersect the arbitrary plane f_e, $\mathbf{A}_{\partial^+ e_f,f_e} \oplus \mathbf{A}_{\partial^- e_f,f_e}$ and (b2) counting the number of these edges that intersect the plane to the right (or to the left) of l_e. This number is odd (the XOR operator is used for parity testing) if and only if the second condition of the edge-face intersection test holds.

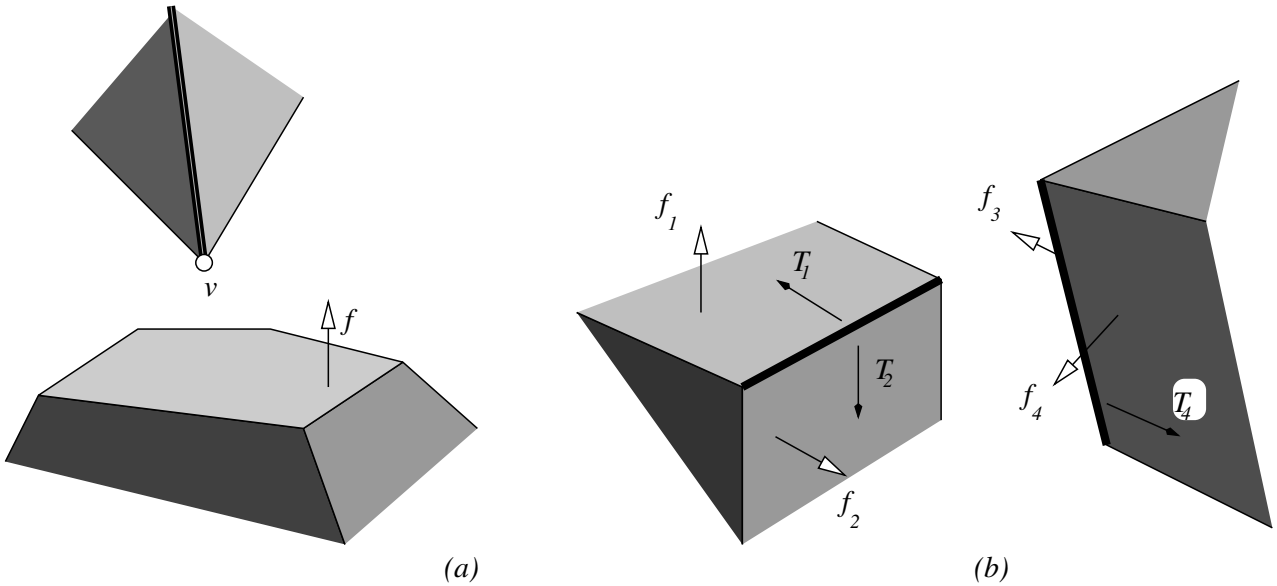

Fig. 2. (a) Applicable vertex-face contact, (b) applicable edge-edge contact.

- From the edge-edge contact, any face adjacent to an edge should be paired with the other edge.

Note that although some repeated pairings may be obtained, both types of applicability need to be considered, because otherwise some candidate pairings could be missed, as depicted in Figure 3.

In the convex case, the applicability conditions guarantee that the contact is possible. If the polyhedra are nonconvex, applicability expresses a necessary but not sufficient condition for contact. One has to talk about local applicability in the nonconvex case, as far as only the adjacent features are considered in the conditions, but other features of the polyhedra may prevent the locally applicable contact from being realized. Therefore, a distinction has to be made between (locally) applicable contacts and those that are also realizable. Figure 4 shows a locally applicable contact that is not realizable. Edge-face pairings arising from such a situation will be called false candidates.

4. A Suitable Representation for Detecting Applicable Contacts

4.1. Spherical Face Orientation Graph

As shown in the preceding section, the applicability of contacts between surface features depends on the relative orientations of these features. Thus, a natural frame for representing applicability is the unit sphere of orientations. The Gaussian map of a given surface represents the surface normals as points on the unit sphere (Hilbert and Cohn-Vossen 1987; Horn 1984). The spherical face orientation graph (SFOG) extends this concept by representing edges and vertices of polyhedra and their adjacencies, as described next.

Faces are represented by nodes on this sphere. In fact, the node represents the orientation of the outward normal of the plane supporting the face, that is, that pointing outside the polyhedron.

Edges are represented by arcs. These arcs join nodes that correspond to faces sharing an edge. Geometric consistency is attained by placing these arcs on great circles of the sphere. In this way, the normals of the planes that define these great circles point in the same directions as the corresponding edges. Convex edges are represented by means of the minor arc, concave edges with the major arc. The representation, thus, characterizes the type of arc and is coherent with the criterion of considering the supplementary angle of the internal dihedrical angle between the faces. The arcs will be called convex or concave depending on the type of edge they represent.

A vertex is represented by the region enclosed within a cycle of convex arcs and nodes, corresponding to adjacent edges and faces. This region is well defined for convex vertices (where all the adjacent edges are convex). Nonconvex vertices have one or more nonconvex adjacent edges. For some nonconvex vertices (pseudoconvex), it is possible to select a subset of adjacent convex edges that define locally a pyramid that contains all other adjacent edges (this pyramid is the local convex hull of the vertex). The convex arcs that correspond to these edges enclose a so-called convex subregion on the sphere.

Figure 5 shows how the three types of features of a polyhedron are represented on the SFOG for a convex vertex, and Figure 6 illustrates the concepts of convex subregion and local convex hull for a pseudoconvex vertex.

Correctness of this representation is proven in Jiménez (1998).

The SFOG is not unambiguous: a polyhedron may not be reconstructed from this representation, neither in size nor completely in shape, as shown in Figure 7. Nevertheless, although this ambiguity may be a drawback for certain applications, it is not a problem for the purposes pursued here. The point is that the SFOG representation preserves those geometric relations that are relevant for the applicability conditions.

4.2. Pairing of Applicable Features Using the SFOG Representation

By superimposing the SFOG of one polyhedron on the central symmetric image of the SFOG of another polyhedron (see Fig. 8), a compact representation is obtained from which the vertex-face and edge-edge applicability relationships can be directly determined:

1. *Convex vertex-face applicability.* A given node falls into a certain region if and only if the contact between the vertex represented by the region and the face represented by the node is applicable (Fig. 9).

2. *Nonconvex vertex-face applicability.* A given node falls into a certain convex subregion if and only if the contact between the vertex whose local convex hull is represented by the convex subregion and the face represented by the node is locally applicable (Fig. 10).

3. *Edge-edge applicability.* Two convex arcs of different SFOGs intersect if and only if the contact between the corresponding edges is (locally, in the nonconvex case) applicable (Fig. 11).

All the properties of applicability relations are adequately reflected in the representation. For example, the existence of at least one applicable vertex per face is captured by the fact that each node falls in at least one region, and the lack of applicable faces for a given vertex shows up in that the corresponding region would contain no nodes. An important difference between the SFOGs of convex and nonconvex polyhedra has to be stressed: in the convex case, the regions corresponding to the vertices constitute a partition of the spherical surface, whereas for nonconvex polyhedra, the convex subregions cover the sphere and may overlap. This means that in

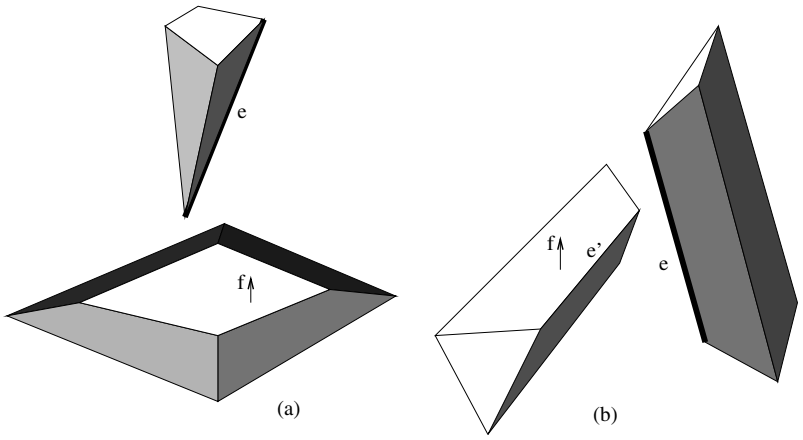

Fig. 3. (a) The candidate pairing e-f can only be obtained from the applicable v-f pair as no edge in the boundary of f can contact the edges stemming from the vertex, (b) the candidate e-f can only come from the applicable e-e' pair as no endpoint of either edge can contact the faces adjacent to the other edge.

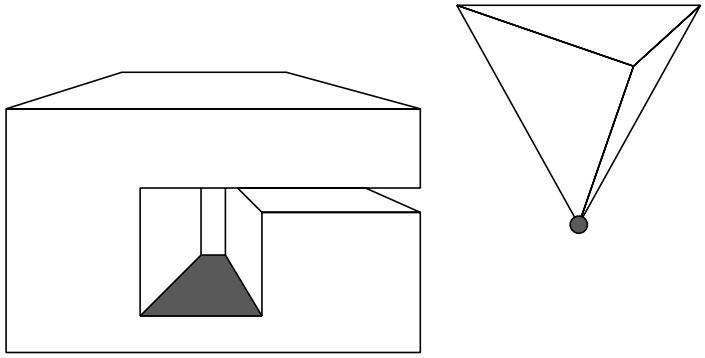

Fig. 4. A locally applicable but not realizable vertex-face contact.

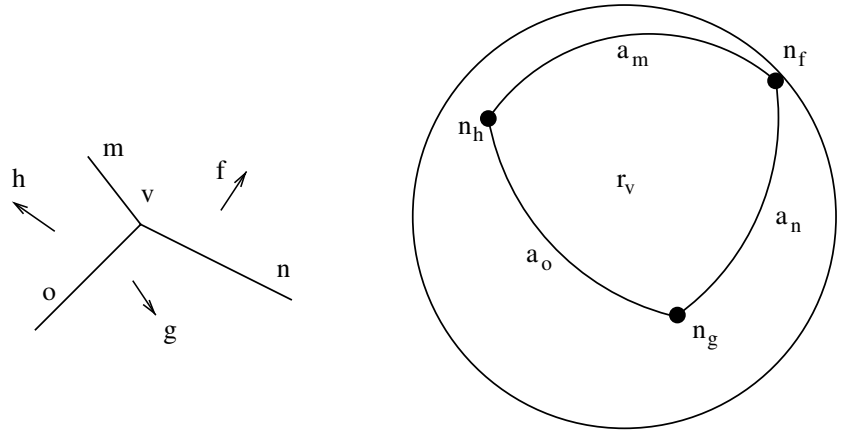

Fig. 5. Representation of polyhedral features on the spherical face orientation graph: a vertex v is represented by region r_v, its adjacent faces f, g, and h by the nodes n_f, n_g, and n_h, and the adjacent edges m, n, and o by the arcs a_m, a_n, and a_o.

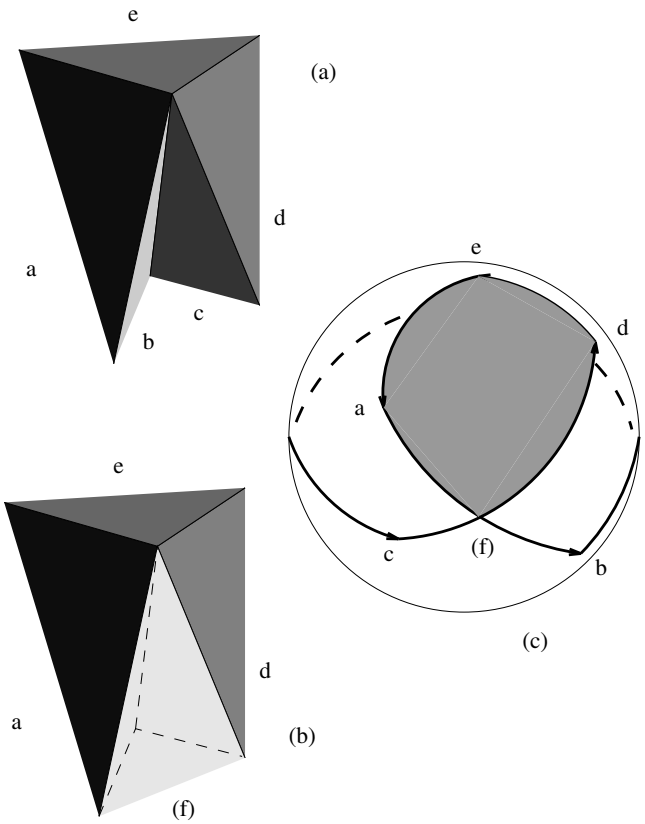

Fig. 6. (a) A pseudoconvex vertex, (b) its local convex hull, and (c) the corresponding convex subregion.

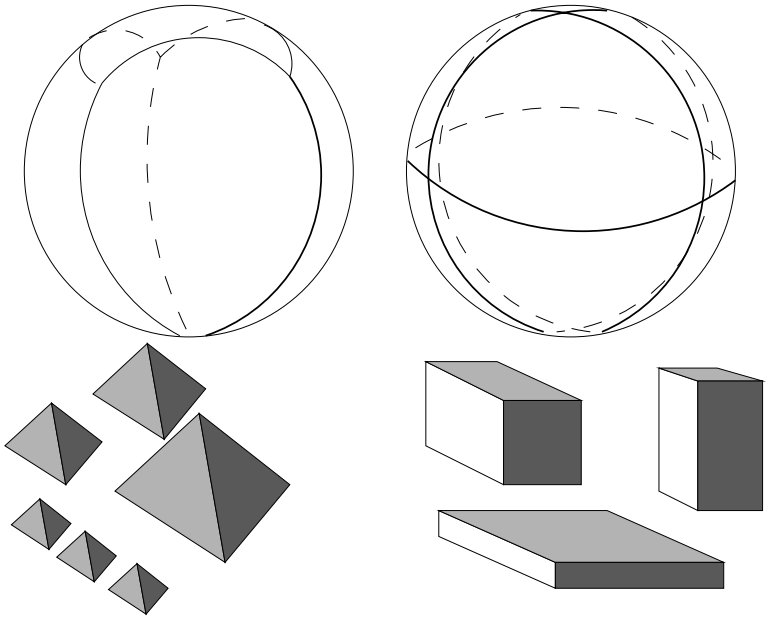

Fig. 7. Different polyhedra will have the same representation on the spherical face orientation graph if the relative orientations of the faces and adjacency relationships are preserved.

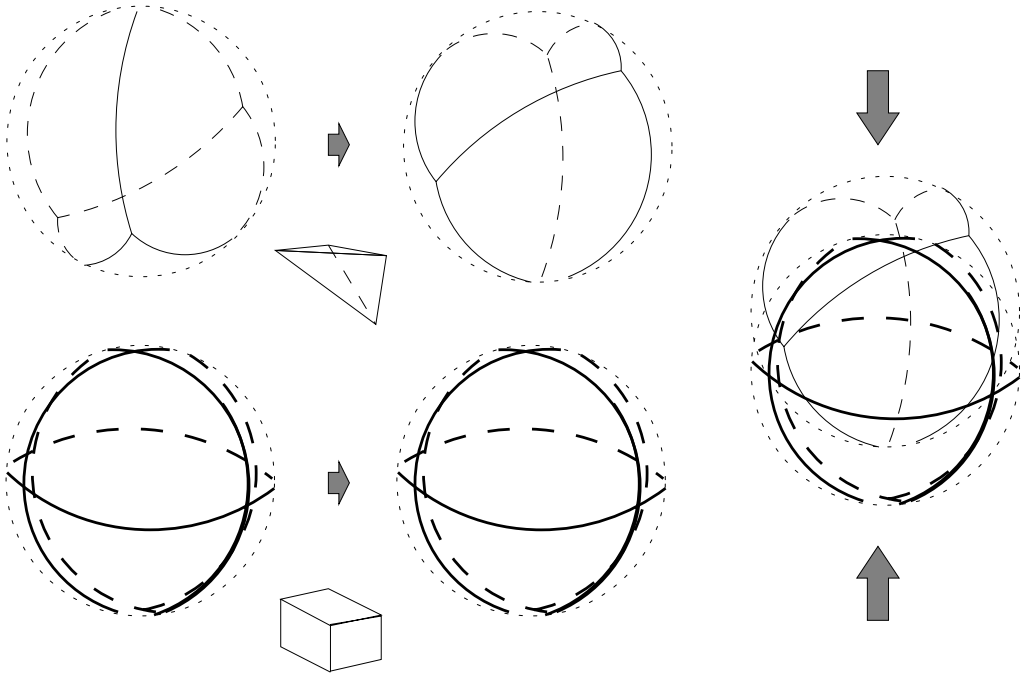

Fig. 8. Overlay of the spherical face orientation graphs (SFOGs) corresponding to two polyhedra in order to obtain a compact representation that allows one to determine the applicability relationships. The SFOG of the rectangular prism below (heavy lines) is combined with the central symmetric image of the SFOG of the tetrahedron above (fine lines).

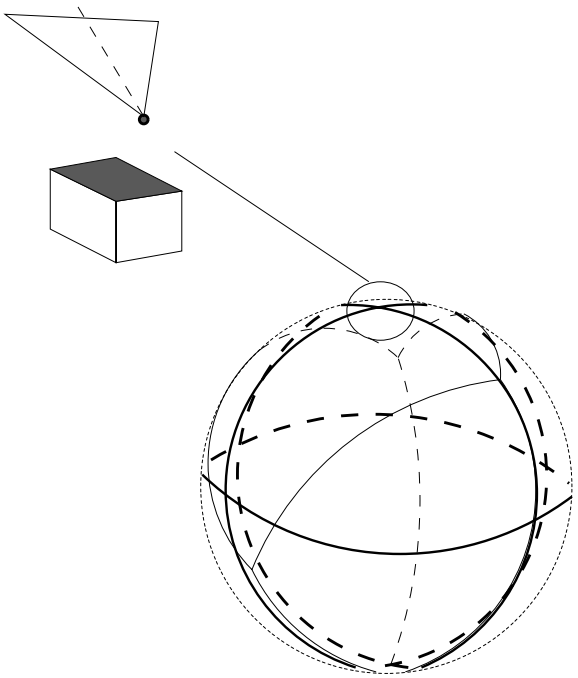

Fig. 9. An applicable vertex-face contact, where the vertex is convex.

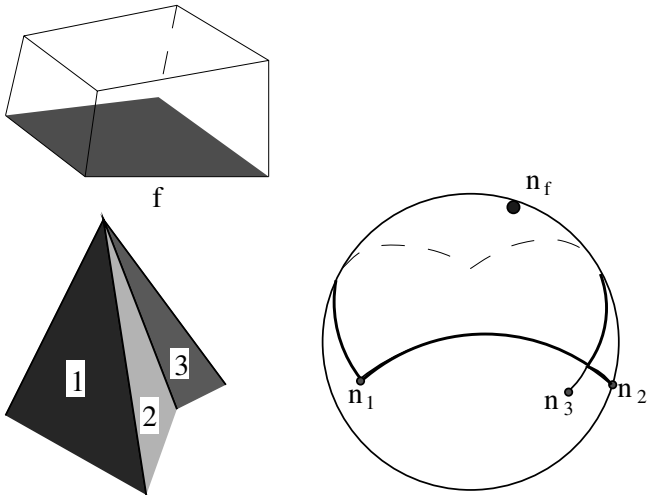

Fig. 10. An applicable vertex-face contact, where the vertex v is pseudoconvex. The node n_f corresponding to the face f is contained in the convex subregion corresponding to the local convex hull of the vertex.

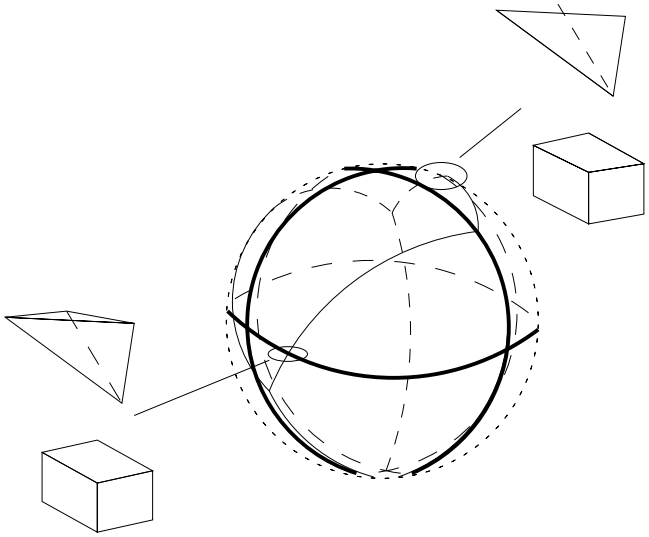

Fig. 11. Two examples of applicable contact between edges. The corresponding arcs intersect.

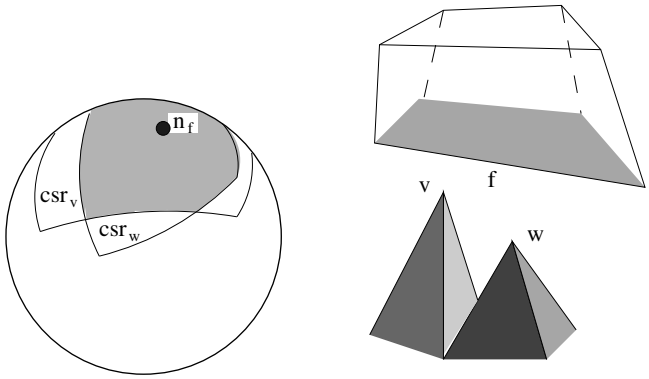

Fig. 12. For nonconvex polyhedra, situations may exist in which the same face is simultaneously applicable to several vertices: the corresponding node is contained in the common area of the overlapping convex subregions of these vertices.

the latter case, a given face may be simultaneously applicable to various vertices if the node that represents this face lies inside the intersection of the corresponding convex subregions. Figure 12 shows one such situation.

The next step is to develop an algorithm to obtain all the applicable feature pairs efficiently.

5. The Convex Case

The amount of pruning that can be done becomes particularly evident in the convex case. Therefore, a simple algorithm has been developed and implemented for the situations in which the polyhedra are known to be convex. This algorithm considers nodes of one SFOG and regions of the other one and performs the node-in-region inclusion and the arc-crossings tests.

The data structures used in the algorithm are as follows:

- Input graphs
  - The SFOGs
  - A cycle graph (in which nodes correspond to vertices of one polyhedron)
- Output vectors
  - FACE_APP[<node>] recording the face-vertex applicability relationships
  - EDGE_APP[<arc>] recording the edge-edge applicability relationships
- Processing lists
  - OPEN_NODES
  - OPEN_ARCS

A brief description of procedures and functions is needed for the clear understanding of the algorithm. The procedure *ordered_intersection (node, &EDGE_APP)* finds every intersection of arcs stemming from *node* with the arcs of the cycle in which *node* lies and appends these intersections to EGDE_APP. The *arc_intersection(arc, cycle, edge)* function finds the intersection between *arc* and the arcs of *cycle* different from *edge*, which is known to have been crossed by *arc* in entering *cycle*. The function *Succ_Arcs(node)* returns the arcs that "point out of" *node*, in the sense that although we are exploring an undirected graph, certain directions of the arcs are implicitly imposed as some nodes are explored before others and we want to avoid exploring a given arc in both directions. The function *succ_node(arc)* returns the unexplored extreme node of *arc*, and *succ_cycle(edge, arc)* returns the cycle that cobounds *edge* and where *arc* is "pointing to" (in the sense that the other cobounding cycle will either contain the node such that *arc* $\in$ *Succ_Arcs(*node*)* or will already have an arc intersected by *arc*). Finally, the function *last(EDGE_APP[arc])* returns the last arc intersected by *arc*.

The algorithm starts at a given point, which can be considered without loss of generality as a "north pole," and travels over the sphere toward the "south" in a spiral-like fashion. Thus, it can be considered a greedy or breadth-first algorithm (Pearl 1984).

SFOG Search Algorithm (Convex Polyhedra)
Choose North-pole;
Find North-region $\supset$ North-pole;
FACE_APP[North-pole]:= North-region;
North-pole $\rightarrow$ OPEN_NODES;
while (OPEN_NODES $\neq \emptyset$) or (OPEN_ARCS $\neq \emptyset$)

```
while (OPEN_NODES ≠ ∅)
  node ← OPEN_NODES;
  ordered_intersection(node, &EDGE_APP);
  for every a ∈ Succ_Arcs(node)
    if EDGE_APP[a] = ∅ then
      if FACE_APP[succ_node(a)]= ∅ then
        FACE_APP[succ_node(a)]:=
            FACE_APP[node];
        succ_node(a) → OPEN_NODES;
      endif
    else
      a → OPEN_ARCS;
    endif
  endfor
endwhile
while (OPEN_ARCS ≠ ∅)
  arc ← OPEN_ARCS;
  edge := last(EDGE_APP[arc]);
  cycle := succ_cycle(edge, arc);
  s := arc_intersection(arc, cycle, edge);
  if s=∅ then
    if FACE_APP[succ_node(arc)] = ∅ then
      FACE_APP[succ_node(arc)] := cycle;
      succ_node(arc) → OPEN_NODES;
    endif
  else
    EDGE_APP[arc] ← s;
    arc → OPEN_ARCS;
  endif
endwhile
endwhile
```

Complexity of this algorithm is linear in the output. A worst case can be found in which the size of this output is quadratic due to edge-edge applicability relationships (when the number of edges bounding a face of polyhedron P is $O(n_P)$ and the number of edges applicable to each of them is $O(n_Q)$), but it is attached to a very specific geometry in which a large number of vertices are coplanar. If vertices are in general position, the number of edges applicable to the boundary of a face is bounded by a constant and the output complexity becomes linear in the input.

The algorithm was implemented and applied to pairs of polyhedra with a given relative orientation. Experimental results displaying the computational savings obtained by running this algorithm previously to the contact determination test are reported in Section 7.1.

6. Nonconvex Polyhedra: A Conservative Pruning

For the case of nonconvex polyhedra, convex subregions have to be identified and arc crossings of two kinds (between convex arcs of the same SFOG and between convex arcs of different SFOGs) have to be distinguished. The algorithm has to perform three main tasks: detect arc crossings, detect regions overlap, and detect node-in-region inclusions. At the base lies the arc-crossings detection procedure. It consists of applying a modified version of segment intersection detection algorithms through plane sweeping, such as those that will be reviewed in Section 6.1. These algorithms have to be adapted for treating arcs on the sphere instead of segments in the plane: the sweep with a vertical line is replaced by a sweep with a meridian. As in the case of line segments, the arcs are monotone in the direction of the sweeping, and every two arcs intersect at most once. The partial ordering that the sweep induces on the arcs is used to keep track of the regions that are being swept, and this in turn allows one to perform the other two tasks of region overlap and node-in-region inclusion detection. In fact, these two steps are merged together, as far as the aim of the region overlap detection step is to allow one to know in which regions a given node lies, that is, which vertices are simultaneously applicable with respect to the same face.

The need to eliminate nonconvex arcs, since they are never applicable, implies that we must deal with the most general setting of the red-blue segment intersection and node-in-region inclusion detection problems: not only monochromatic (i.e., belonging to the same SFOG) arc intersections will appear but also disconnected nodes, arcs, and regions. A simple example, depicted in Figure 13, shows how a region representing a vertex may be entirely disconnected from the neighboring features when nonconvex arcs are discarded.

6.1. Line Sweep Algorithms for Red-Blue Segment Intersection Detection

Consider a set of line segments in the plane. A common strategy for detecting all the intersections between them consists of applying the so-called plane sweep technique (also known as the line sweep or sweep line technique), as generically described in Preparata and Shamos (1985). A sweep line, assumed without loss of generality to be vertical, is swept through the whole plane. At a given instant, the sweep line is intersecting some segments of the considered set. All the segment intersections to the left of the sweep line have already been computed and will not be affected by subsequent intersections to the right. The sweep line introduces in a natural way an adjacency relationship between the segments it intersects. This allows one to consider for intersection only the adjacent segment pairs (contrary to the brute-force approach, which considers all possible pairings). These adjacency relationships change each time a segment endpoint is encountered or two segments intersect. Therefore, the continuous sweep process can be discretized by means of the event point schedule (i.e., the sequence of abscissas that correspond to the segments' endpoints as well as the intersection points) and the structure that maintains the sweep line status will allow for

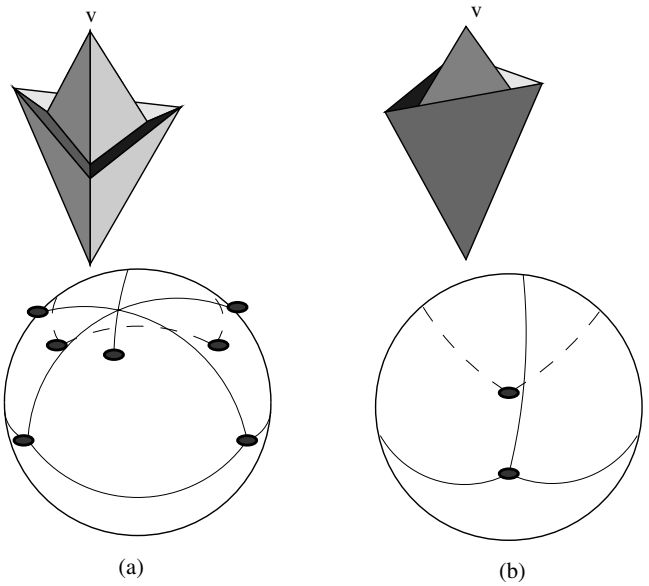

Fig. 13. The nodes and arcs (shown with dashed lines) of the region that represents v are disconnected from the remaining part of the spherical face orientation graph if the nonconvex arcs are eliminated. The nodes that correspond to faces of the polyhedron are shown as small ellipses. (a) Front view, (b) rear view.

queries concerning adjacency relationships (i.e., segment intersection candidates).

The best solution, from the point of view of efficiency, is to make use of algorithms specially devised for the case in which monochromatic intersections exist but do not have to be computed. Agarwal and Sharir (1990) described an algorithm that reports all red-blue intersections between line segments in time $O((n_r\sqrt{n_b} + n_b\sqrt{n_r} + k_{r-b})\log(n_r + n_b))$. This complexity was improved in Agarwal (1990) to overall $O(n^{4/3}\log^{(\omega+2)/3} n + k_{r-b})$ time and using $O(n^{4/3}/\log^{(2\omega+1)/3} n)$ space, where ω is a constant < 3.33 and $n = n_r + n_b$, by applying a divide-and-conquer strategy. This algorithm is deterministic; using random-sampling techniques, an expected time of $O(n^{4/3}\log n + k_{r-b})$ was obtained (Agarwal 1990).

Using recently developed data structures that allow one to maintain a partial order on the line segments, an expected time of $O((n + k_{r-b})\alpha(n)\log^3 n)$ can be obtained for the case in which the sets of red and blue segments are connected (Basch, Guibas, and Ramkumar 1996). If red and blue segments are grouped into c_r and c_b connected components, the time becomes $O((c_b n_r + c_r n_b + k_{r-b})\log^3 n\alpha(n))$. The main idea of this approach consists of considering the red and blue arcs intersecting the sweep line at a given instant grouped into blocks, so that only the bottom segment has to be tested for intersection against the top segment of the contiguous block. The crucial issue is to guarantee that at every instant, the extremes of each block are correctly updated. This means that besides the events attached to the endpoints of each segment and the purple intersections, some monochromatic intersections have to be scheduled. The wise use of data structures that combine the features of binary trees and heaps permits restricting these monochromatic intersections to a small subset of the whole. Further details on this algorithm and its use for detecting the applicable edge-edge pairings can be found in Jiménez (1998), who provided the means of determining efficiently the node-in-region inclusions during the same sweep on the sphere using the same grouping of arcs into blocks along the sweep line.

This solution appears to be very attractive from the point of view of complexity, and the algorithm is not very difficult to implement. Nonetheless, the use and maintenance of the involved sophisticated data structures is highly time consuming, which renders this algorithm less efficient than a naive strategy (described in the following section) when applied to objects of moderate complexities (as those used in our test bed). It is worth mentioning, however, that the savings in arc intersection tests increases faster than the burden of data structure management, implying that there exists a threshold scene complexity beyond which the sophisticated algorithm would always outperform the naive one (Jiménez 1998).

6.2. Naive Sweep Algorithm for Red-Blue Intersection Detection on the Sphere

It is straightforward to adapt the plane sweep principle to the sphere: the vertical sweep line is replaced by a sweep meridian and the sweep begins at an arbitrary point (as we cannot speak of a "leftmost" point) and proceeds eastward (as the plane sweep from left to right). As in the planar case of line segment intersection detection, two arcs will intersect at most at one point, and monotonicity is ensured: one arc cannot intersect the sweep meridian at more than one point simultaneously.

Each time the sweep meridian arrives at the western endpoint of an arc $a[i]$, this arc is tested for intersection with all the active arcs (i.e., arcs currently intersected by the sweep meridian) of opposite color $L_{\bar{c}[i]}$ and included in the list of active arcs $L_{c[i]}$ ($c[i]$ denotes the color of $a[i]$). As soon as the eastern endpoint of an arc is reached, that arc is deleted from the active list. In this way, every purple intersection will be detected exactly once. A pseudocode transcription of this simple algorithm is presented:

Naive SFOG Search Algorithm (Nonconvex Polyhedra)

Preprocessing: Order the 2n endpoints e[i] by increasing meridian value. Let a[i] be the arc with endpoint e[i] and c[i] its color.

for ($i = 1..2n$) do
  if ($e[i]$ is a western endpoint) then
    for (all $a' \in L_{\bar{c}[i]}$) do

```
        if (a[i] ∩ a′) then
            report purple intersection
        endif
      endfor
      insert(a[i], L_c[i])
    else
      delete(a[i], L_c[i])
    endif
endfor
```

As mentioned before, the "first" endpoint is an arbitrary choice and, at this first instant, no lists of active arcs exist. Therefore, a second sweep will have to be performed to take into account all the purple intersections with arcs that are still active after the last endpoint. Figure 14 depicts the purple intersections that can be detected along the first sweep and those that cannot be determined if no second sweep is performed.

As for node-in-region inclusions, they are computed during the same sweeping operation: each region defines an interval on the sweep line, and it has to be determined which intervals of the opposite color include the current endpoint. Each interval is defined by the upper and lower arcs bounding the region at every instant. Regions begin and end at given (not necessarily all) endpoints. The regions a, say, red node belongs to are computed by determining the blue arcs that cut the sweep line above this node and then finding out whether the regions underneath these arcs are also bounded by arcs that cut the sweep line below that node.

7. Experimental Results

Consider the three algorithms:

Algorithm A follows the brute-force approach of testing all possible edge-face pairings.

Algorithm B consists of the specific SFOG search procedure for convex polyhedra (Section 5) followed by interference tests for the resulting edge-face pairings.

Algorithm C consists of the naive SFOG search procedure for nonconvex polyhedra (Section 6) followed by interference tests for the resulting edge-face pairings.

For comparison purposes, these three algorithms were implemented and experiments carried out over two sets of nonintersecting pairs of polyhedra, convex and nonconvex. The data structure that stores the polyhedra is basically a geometric description of faces (coordinates of their vertices and supporting planes), and the topological information is contained in an adjacency matrix. This structure captures implicitly the information about the edges, which leads to certain inefficiency when this information has to be made explicit. Mainly, this means that every edge is counted twice when the whole polyhedron is examined, which could be avoided using another structure such as the doubly connected edge list (Preparata and Shamos 1985). Nonetheless, the results are suitable for comparison purposes, as the same data structure constitutes the input in all three cases.

7.1. Convex Polyhedra

Despite the fact that general polyhedra (i.e., both convex and nonconvex) are the target of our pruning strategy, we first restrict our test bed to convex solids. This permits testing the algorithm in Section 5 and assessing the pruning potential of the strategy in a controlled setting. Convex polyhedra allow one to design situations that cover the range from a high degree of pruning (such as in polyhedra that approximate spheres) to almost none (such as in prisms of a high degree) without the distortions due to false candidates. The convex polyhedra used in the experiments range from the tetrahedron to a polyhedral approximation of the sphere with 128 triangular faces, and include other regular and semiregular polyhedra, pyramids, prisms, and so on.

Experiments show that it is worthwhile to perform pruning when the polyhedra become complex and that the threshold lies at a relatively low level. Results are displayed in Figures 15, 16, and 17 (note that a logarithmic scale is used). In Figure 15, the number of candidate edge-face pairs before and after pruning are displayed for different relative orientations of the polyhedra. Although this number grows quadratically (linearly in the logarithmic scale) if all possible pairings have to be considered, the savings derived from pruning reduce this growth to a linear one. Execution times in Figure 16 correspond to a SG O2 workstation, 180 Mhz, 192 MB RAM, SPEC int95 4.8, SPEC fp95 5.4. As can be seen, savings in computational time attain various orders of magnitude. Computation of the applicable feature pairings and pruning needs to be done only once, whereas the polyhedra maintain their relative orientations. Even if the time to perform these computations is included, the total time is less than that needed by algorithm A as the polyhedra become more complex, as shown in Figure 17.

7.2. Nonconvex Polyhedra

Algorithms A and C were tested on a set of nonconvex polyhedra ranging from a pentahedron (with only one concave edge, i.e., a pyramid with a V-shaped base) to an hour-glass-shaped polyhedron (160 edges, including 32 concave ones). The results are displayed below. Figure 18 shows a drastic reduction in the number of edge-face intersection tests to perform for the considered objects. Figure 19 displays the execution times of algorithms A and C, the latter without and with the preprocessing step. We have to insist that this preprocessing has to be done only once. Once again, a logarithmic scale is used in both figures.

Next, we carried out some experiments to assess the scalability of the approach and to compare its performance

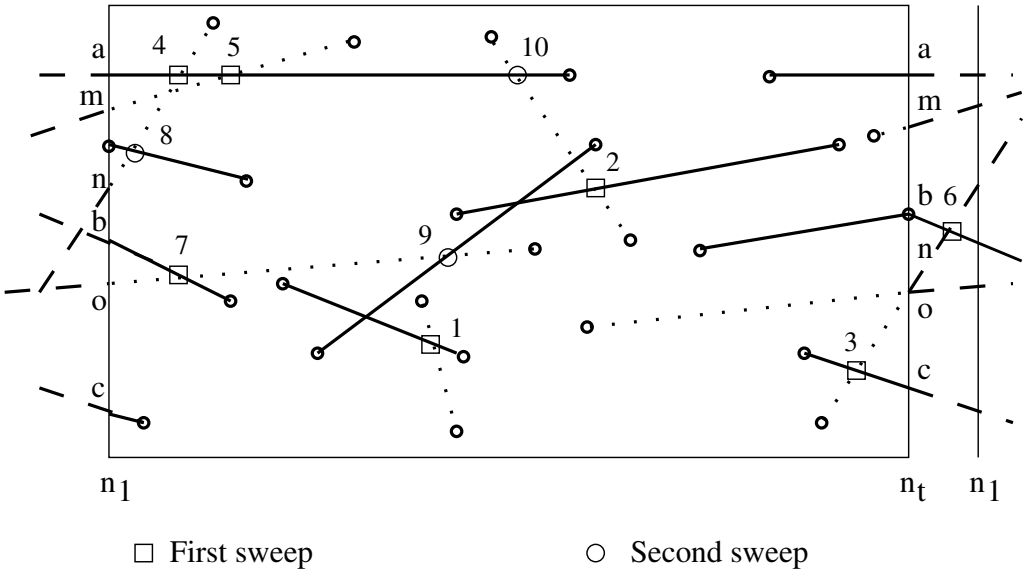

Fig. 14. Plane analog of the spherical sweep. Red and blue arcs are depicted as dotted and solid segments. Some of them (a, b, c, m, n, o) begin before or at the last event point n_t and end after the first one n_1. These segments are activated during the first sweep from $e[1]$ to $e[2n]$, where some purple intersections can already be detected (marked with a small square), and their intersections with the segments that had ended before they were generated (marked with an empty circle) can only be detected during a second sweep. Numbers indicate the order in which the intersections are computed.

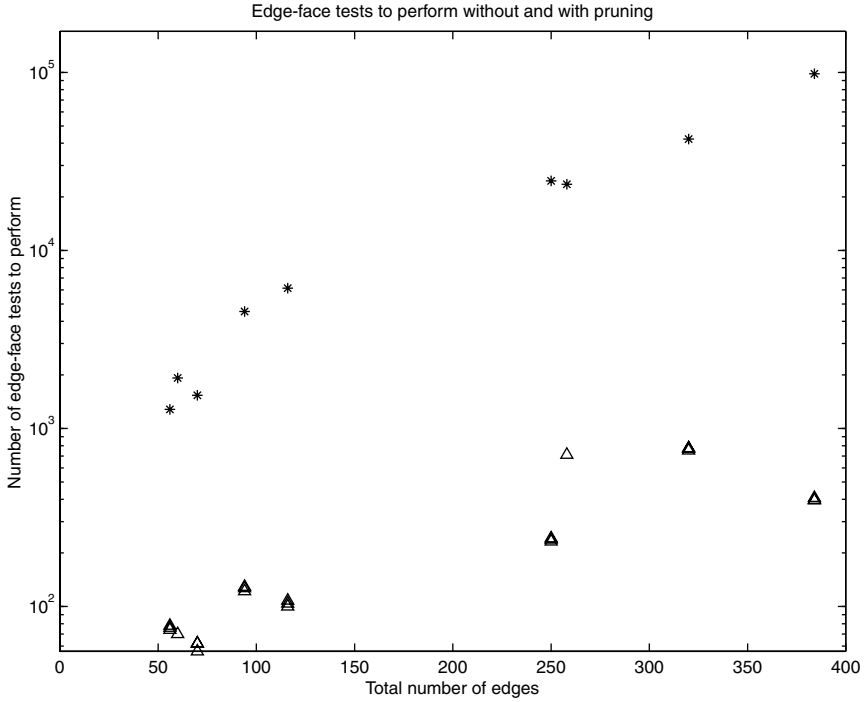

Fig. 15. Comparison between the total number of edge-face tests to perform without applicability pruning, that is, algorithm A (*), and with a previous pruning step, that is, algorithm B (△).

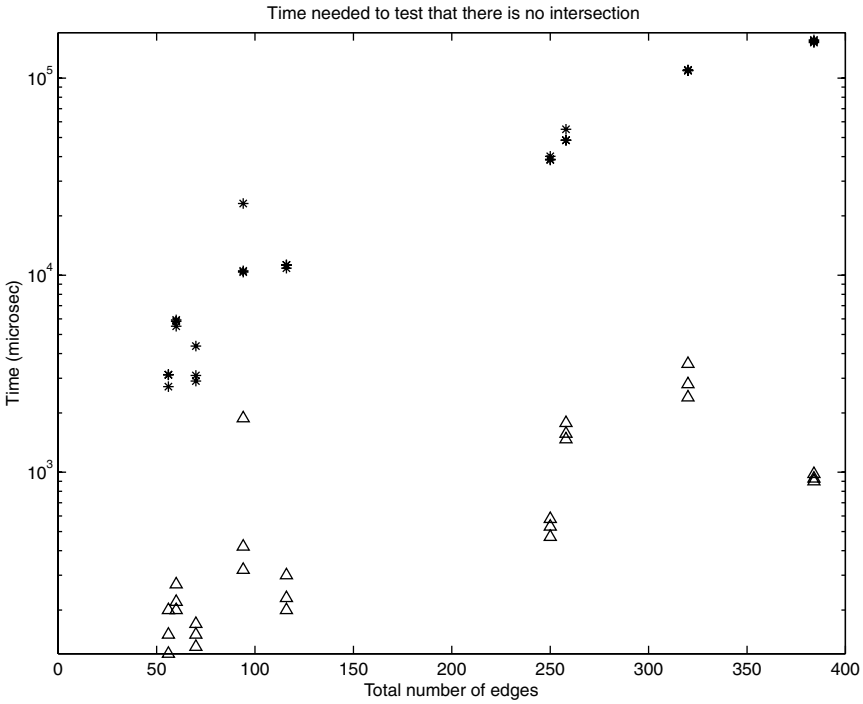

Fig. 16. Comparison of execution times of algorithm A (*) and the interference part of algorithm B (△).

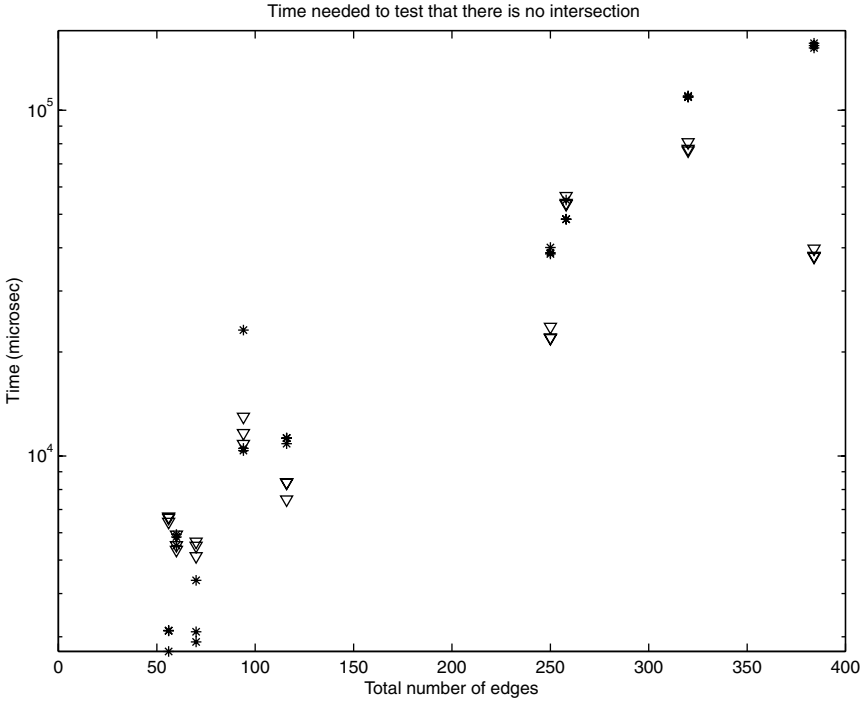

Fig. 17. Comparison of execution times of algorithm A (*) and algorithm B including preprocessing time (∇). Note that with preprocessing added, B outperforms A even for low complexities. The different results for each pair of polyhedra correspond to different relative orientations.

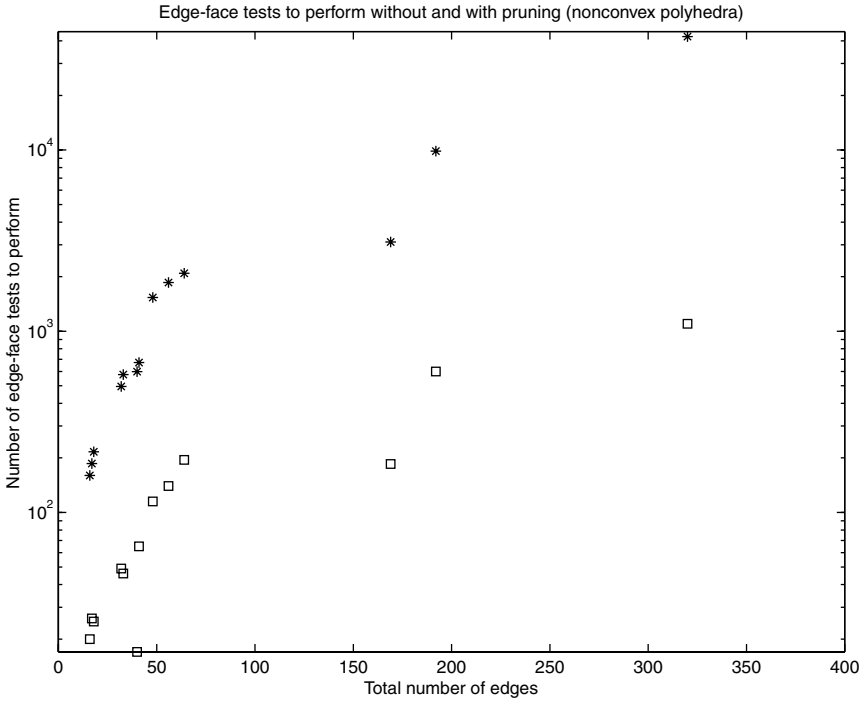

Fig. 18. Comparison between the total number of edge-face tests to perform without applicability pruning, that is, algorithm A (*), and with a previous pruning step, that is, algorithm C (□), in settings that include only nonconvex polyhedra.

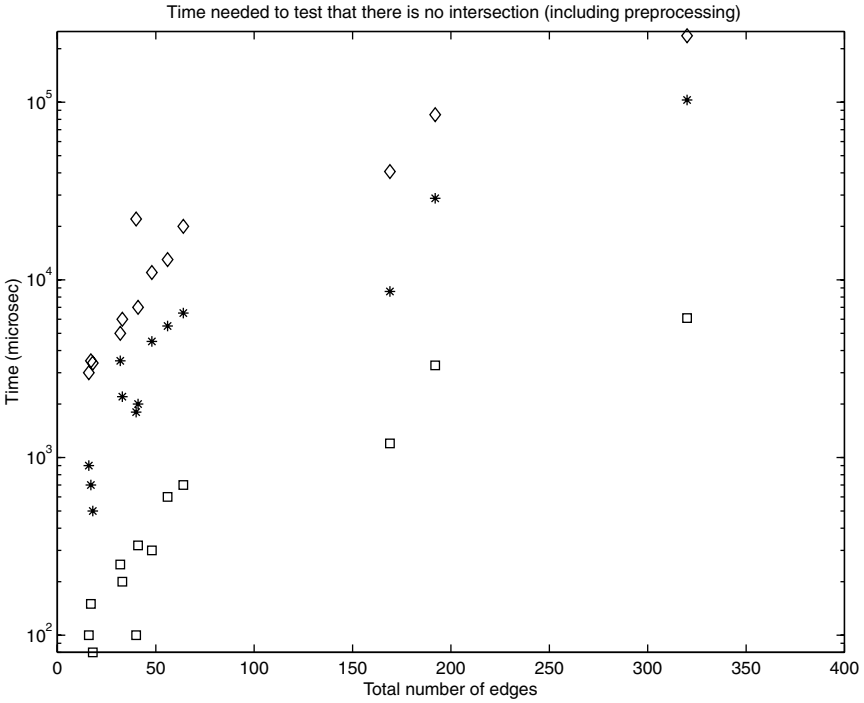

Fig. 19. Comparison of the execution times of algorithm A (*) and algorithm C (□) with a previous pruning step (◇) in settings that include only nonconvex polyhedra.

against that of a well-known collision detection package (RAPID, v2.01, publicly available at http://www.cs.unc.edu/~geom/OBB/OBB.html). As a test bed, we used pairs of cubes having an increasing number of square prismatic holes arranged on square grids, as shown in Figure 20.

The two cubes are located and oriented such that the algorithms are forced to perform a high number of elemental intersection tests (e.g., for the case in which both cubes have 12×12 holes, RAPID reports 22,947 contacts out of 576,267 box tests whereas algorithm C performs 171,400 edge-face intersection tests).

Figure 21 shows the increasing power of applicability pruning as the complexity of the setting grows. Again, we observe a linear growth of the number of edge-face tests to perform versus the quadratic growth of the total number of edge-face pairings.

The experimental results displayed in Figure 22 show that for such kinds of settings, our algorithm is one order of magnitude faster than RAPID in carrying out contact determination. For example, in the case of two cubes with 12×12 holes each (e.g., having 3480 edges), algorithm C detects all contacts in 620 ms whereas RAPID needs 3051 ms. The gain in speed is attained at the expense of a much higher preprocessing time (Fig. 23). In the latter figure, times that are accounted for are searching the SFOG representation for determining all pairs of applicable features and extracting all possibly intersecting edge-face pairings for algorithm C, whereas for RAPID the measured time corresponds to Seidel's triangulation plus the building up of the hierarchical data structures.

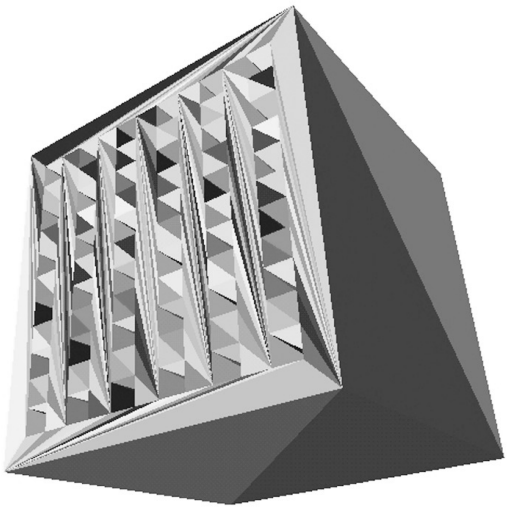

Fig. 20. The type of workpiece used to compare the performances of algorithm C and RAPID. The faces of the polyhedron (displayed here for the case of 6×6 holes) were triangulated with an implementation of Seidel's algorithm (the code was obtained at http://www.cs.unc.edu/~dm/CODE/GEM/chapter.html). Remember that triangulation is a necessary previous step for RAPID but not for our algorithm.

It should be noted that the purpose of the comparison with RAPID is just to show the potential of our approach. Thus, we have chosen the most favorable setting for our algorithm, namely, two objects with many concavities, placed in a way that many simultaneous contacts occur. Both algorithms are in some way complementary in the sense that pruning is based on volumetric aspects in RAPID whereas the algorithm presented here exploits orientation information. Thus, both approaches are not mutually exclusive, and possibilities exist for their integration.

8. Conclusion

An orientation-based pruning strategy for reducing the computational effort in contact determination between general polyhedra was described. This strategy is based on applicability conditions and consists basically of reducing the number of edge-face pairs to be considered for intersection by taking into account that only a subset of all contacts between the features of two polyhedra is actually possible, if the relative orientation of the polyhedra does not change. It has to be stressed that in our approach, nonconvex polyhedra can be directly tested for interference without decomposing them into convex entities. Therefore, applicability pruning is a preprocessing step, like the usual practice of decomposing the polyhedra into convex entities, but it does not increase the complexity of the setting by introducing fictitious features, as the convex decomposition does.

Experiments carried out in settings with convex and nonconvex polyhedra show important savings in computational effort: contact determination based on the edge-face intersection test performs 10 to 100 times faster if a previous selection of candidate pairs based on the applicability conditions is performed. Although highly dependent on the specific geometry of the involved polyhedra, it can also be stated that these savings increase proportional to the complexity of the setting.

Our approach used alone, without a bounding volume hierarchy or any other encapsulation method, can outperform interference detection packages such as RAPID only under very restrictive conditions, namely, when polyhedral models with many concavities translate in close proximity, leading to the occurrence of many simultaneous contacts. This is a typical situation arising in assembly design and planning within CAD/CAM systems.

Moreover, both the edge-face intersection test and the proposed orientation-based pruning can be seen as general tools, which can be combined with other space and time bounding strategies and integrated in a given collision detection scheme.

Two issues deserve further attention. First, because many computer graphics applications depend on surface normals, several data structures have been proposed for encapsulating

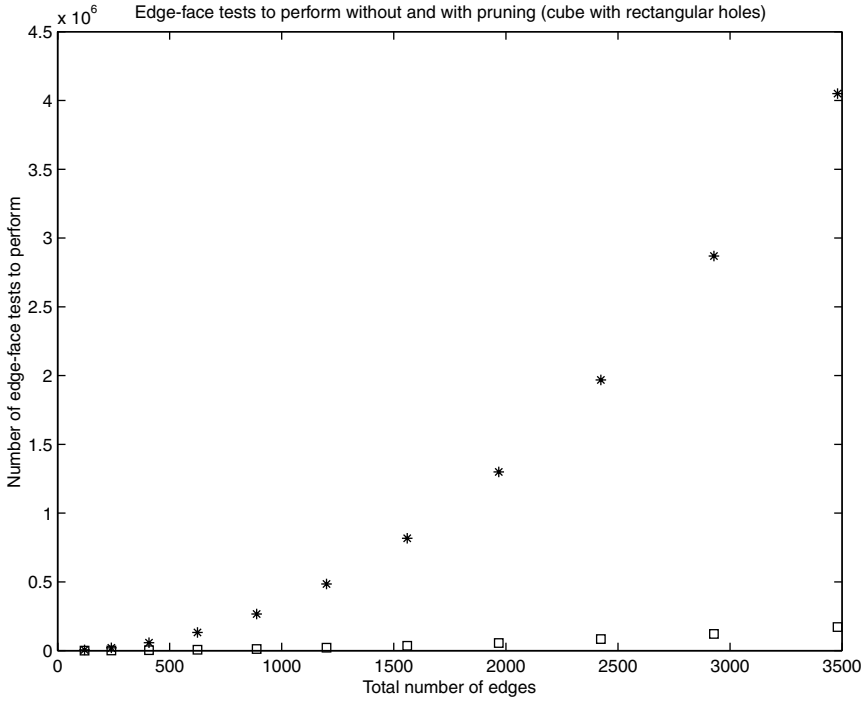

Fig. 21. Number of edge-face intersection tests to perform by algorithm C (□) against the total number of edge-face pairings (*) for the setting in which two cubes have an increasing number of holes.

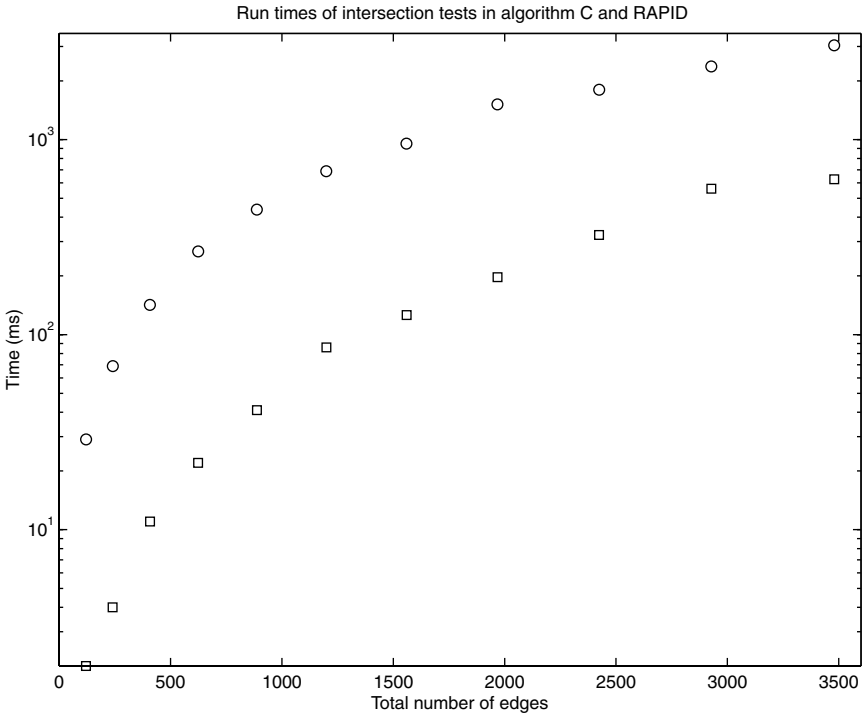

Fig. 22. Comparison of the execution times of the interference detection phase in algorithm C (□) and RAPID (○).

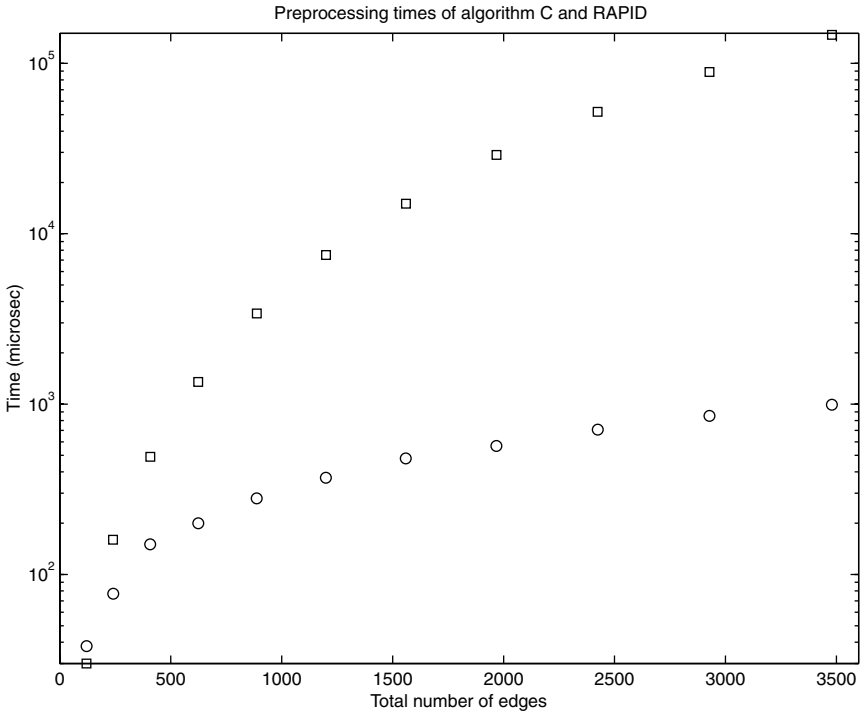

Fig. 23. Comparison of preprocessing times of algorithm C (□) and RAPID (○). Preprocessing is various orders of magnitude slower for the former. However, it has to be performed only once in settings where objects are only allowed to translate.

surface orientations. Most of them were developed to tackle interactions between a point and a polyhedral model, but recently one such structure, namely the spatialized normal cone hierarchy, was applied in the computation of local distances in polyhedral settings (Johnson and Cohen 2001). We see this as the orientation-based counterpart effort to that of establishing hierarchies of bounding volumes, and therefore we would like to explore the possibility of integrating such type of hierarchy into our approach.

The other issue is that the applicability conditions are orientation dependent, which means that along trajectories that entail a change in the relative orientation of the polyhedra, new edge-face candidates arise while others become no longer valid. Therefore, one must be able to determine the intervals of isoapplicability, that is, the ranges of relative orientations—along the trajectory—for which the same applicability conditions hold. In a trajectory parameterization approach, this means determining the values of the parameter where these changes occur, whereas in a multiple interference detection approach, a discretization of time based on isoapplicability will have to be considered (besides the standard discretization based on distance and relative velocities). The SFOG representation can be used to this end: the intervals of isoapplicability are delimited by the rotation events each time a node of one SFOG crosses an arc of the other one. These questions are addressed in Jiménez (1998), but devising an efficient method for computing all the rotation events for arbitrary changes in the relative orientation of the polyhedra is still an open issue.

Acknowledgments

This paper provides a revised and extended account of results presented at the IEEE International Conference on Robotics and Automation (Jiménez and Torras 1996, 1999). This research was partially supported by the Spanish Science and Technology Commission (TAP99-1086-C03-01) and the Catalan Research Commission. We would like to thank two anonymous reviewers for their helpful comments.

References

Agarwal, P. 1990. Partitioning arrangements of lines: ii. Applications. *Discrete Computational Geometry Siam Journal of Computing* 5:533–573.

Agarwal, P., and Sharir, M. 1990. Red-blue intersection detection algorithm, with applications to motion planning and collision detection. *Siam Journal of Computing* 19:297–321.

Basch, J., Guibas, L. J., and Ramkumar, G. D. 1996. Reporting red-blue intersections between connected sets of line segments. *4th European Symposium on Algorithms*, pp. 302–319.

Boyse, J. W. 1979. Interference detection among solids and surfaces. *Communications of the ACM* 22(1):3–9.

Cameron, S. A. 1991. Efficient bounds in constructive solid geometry. *IEEE Computer Graphics and Applications* 11:68–74.

Canny, J. 1987. *The Complexity of Robot Motion Planning.* Cambridge, MA: MIT Press.

Cohen, J. D., Lin, M. C., Manocha, D., and Ponamgi, M. K. 1995. I-collide: An interactive and exact collision detection system for large-scale environments. *Proceedings of the ACM International 3D Graphics Conference*, Vol. 1, pp. 189–196. Available: http://www.cs.unc.edu/˜geom/LCOLLIDE.html.

Donald, B. R. 1987. A search algorithm for motion planning with six degrees of freedom. *Artificial Intelligence* 31:295–353.

García-Alonso et al. 1994.

Gilbert, E. G., Johnson, D. W., and Keerthi, S. 1988. A fast procedure for computing the distance between complex objects in three dimensional space. *IEEE Journal of Robotics and Automation* 4:193–203.

Gottschalk, S., Lin, M. C., and Manocha, D. 1996. Obbtree: A hierarchical structure for rapid interference detection. *Proceedings of ACM Siggraph'96*. Available: http://www.cs.unc.edu/˜geom/OBB/OBBT.html

Hamlin, G. J., Kelley, R. B., and Tornero, J. 1992. Efficient distance calculation using the spherically-extended polytope (s-tope) model. *Proceedings of the IEEE Conference on Robotics and Automation*, Vol. 3, Nice, France, pp. 2502–2507.

Hilbert, D., and Cohn-Vossen, S. 1987. *Geometry and the Imagination.* New York: Chelsea.

Horn, B. K. 1984. Extended Gaussian images. *Proceedings of the IEEE* 72:1671–1686.

Hubbard, P. M. 1993. Interactive collision detection. *Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality*, Vol. 1, pp. 24–31.

Hubbard, P. M. 1995. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics* 1:218–230.

Hudson, T. C., Lin, M. C., Cohen, J. D., Gottschalk, S., and Manocha, D. 1997. V-collide: Accelerated collision detection for vrml. *Proceedings of VRML.* Available: http://www.cs.unc.edu/˜geom/V_COLLIDE.html

Jiménez, P. 1998. Static and dynamic interference detection between nonconvex polyhedra. Ph.D. thesis, Universitat Politècnica de Catalunya. Available: http://www-iri.upc.es/people/jimenez/phdthesis.html

Jiménez, P., Thomas, F., and Torras, C. 2001. Collision detection: A survey. *Computers and Graphics* 25:269–285.

Jiménez, P., and Torras, C. 1996. Speeding up interface detection between polyhedra. *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 2, Minneapolis, MN, pp. 1485–1492.

Jiménez, P., and Torras, C. 1999. Benefits of applicability constraints in decomposition-free interference detection between nonconvex polyhedral models. *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 3, Detroit, MI, pp. 1856–1862.

Jiménez, P., and Torras, C. 2000. An efficient algorithm for searching implicit and/or graphs with circles. *Artificial Intelligence* 124(1):1–30.

Johnson, D., and Cohen, E. 2001. Spatialized normal cone hierarchies. *Proceedings of the 2001 ACM Symposium on Interactive 3D Graphics*, Vol. 2, Research Triangle Park, NC, pp. 129–134.

Klosowski, J., Held, M., Mitchell, J., Sowizral, H., and Zikan, K. 1998. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics* 4(1):21–36.

Lin, M. C., and Canny, J. F. 1991. A fast algorithm for incremental distance calculation. *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 2, Sacramento, CA, pp. 1008–1014.

Lin, M. C., and Gottschalk, S. 1998. Collision detection between geometric models: A survey. *IMA Conference on Mathematics of Surfaces*, Vol. 1, San Diego, CA, pp. 602–608.

Martínez et al. 1998.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Reading, MA: Addison-Wesley.

Ponamgi, M. K., Manocha, D., and Lin, M. C. 1997. Incremental algorithms for collision detection between polygonal models. *IEEE Transactions on Visualization and Computer Graphics* 3(1):51–64.

Preparata, F. F., and Shamos, M. I. 1985. *Computational Geometry: An Introduction.* Texts and Monographs in Computer Science. New York: Springer-Verlag.

Thibault, W. C., and Naylor, B. F. 1987. Set operations on polyhedra using binary space partitioning trees. *ACM Computer Graphics* 21(4):153–162.

Thomas, F., and Torras, C. 1992. Inferring feasible assemblies from spatial constraints. *IEEE Transactions on Robotics and Automation* 8:228–239.

Thomas, F., and Torras, C. 1994. Interference detection between non-convex polyhedra revisited with a practical aim. *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 1, San Diego, CA, pp. 587–594.

van der Bergen, G. 1997. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphic Tools* 2(4):1–13.

Vanecek, G. 1994. Back-face culling applied to collision detection of polyhedra. *Journal of Visualization and Computer Animation* 5(1):55–63.