Reducing feasible contacts between polyhedral models to red-blue intersections on the sphere

P. Jiménez *, C. Torras

Institut de Robòtica i Informàtica Industrial (CSIC - UPC)
Llorens i Artigas 4-6, E-08028 Barcelona, Spain

Abstract

Orientation-related problems in geometric design can be naturally expressed on the spherical surface S^2. A wide subset of such problems can be solved directly on the sphere by adapting well-known planar data structures and algorithms. This paper shows that the detection of feasible contacts between two translating polyhedral models can be formulated as a problem of this type. First, a dual spherical representation of polyhedra is introduced, which reduces the contact detection above to finding intersections between two sets of spherical polygons. Next, the red-blue blocks plane sweep algorithm is adapted to obtain both edge intersections and point-in-polygon inclusions in the spherical setting. An experimental comparison of this algorithm against a naïve one shows an increasing advantage of the former as the complexity of the setting grows. The obtained edge-edge and vertex-face polyhedral contacts provide the relevant feature pairs to be tested for interference, leading to considerable savings in collision detection between polyhedral models, as shown in the experimental test performed.

Key words: contact determination, orientation-based preprocessing

1 Introduction

Collision detection is essential to design processes that involve moving parts. A machine design is validated by simulating its operational sequence of movements, and possible collisions have to be checked between the moving components or with workpieces. Likewise, the design parameters of an assembly may have to be changed if it is not possible to find a collision free assembly plan. And immersive CAD systems do also require collision detection between the models and the virtual image of the designer's hands.

* Corresponding author.
Email addresses: pjimenez@iri.upc.es (P. Jiménez), ctorras@iri.upc.es (C. Torras).
URLs: www-iri.upc.es/people/jimenez (P. Jiménez), www-iri.upc.es/people/torras (C. Torras).

A wide class of basic problems in geometric design entails the computation of orientation-dependent relationships between different objects. The set of possible relative orientations of a tool and the surface to be machined, the alignment of specific objects' features during assembly, or the restriction of displacement of pieces in some directions are just some examples. Such information is naturally expressed on the sphere of orientations (i.e., the surface of the unit sphere, S^2 in the 3D case). This representation is complete and provides the setting where typical computational geometry algorithms can be adapted to solve efficiently the problems at hand.

One such problem is that of computing which features (vertices, edges or faces) of two translating polyhedral models may possibly collide, attending to their orientations. In a previous paper [1] we showed that confining the application of a collision checker to only these feature pairs leads to considerable savings. Thus, we concentrate here on how to preprocess the models so as to obtain the possibly colliding features efficiently. To this end, we describe a suitable spherical representation and two alternative algorithms to carry out this preprocessing. Moreover, we show experimentally that, in the assembly domain, the resulting collision checker is competitive, as compared to state-of-the-art packages such as RAPID [2] and PQP [3], therefore opening up the possibility of combining these radically different strategies (i.e., their bounding volume hierarchies and our orientation-based preprocessing).

The paper is structured as follows: first, a brief survey of applications which require solving efficiently geometric problems on S^2 is presented, in order to contextualize our contribution. Next, we describe the problem of checking interference between two general polyhedra, based on the edge-face intersection test, and show how to reduce the number of such elemental tests to perform by precomputing feasible contacts (Section 3). The spherical representation of polyhedra and of feasible contacts, as well as the algorithms that compute them efficiently are explained in Sections 4, 4.1 and 5, respectively. Experimental results are presented in Section 6 and, finally, some conclusions are drawn in Section 7.

2 Computing relationships between 3D objects on S^2

We focus on problems where 3D objects and some type of relationship between them are mapped on S^2, because of their practical interest. However, the literature includes other theoretical problems on the sphere that may be inspiring for some applications, like the construction of spherical Voronoi diagrams [4,5] and minimax localization problems [6]. See also [7] for computational geometry problems on nonplanar surfaces.

2.1 Representations on the sphere

The sphere is the natural representation frame for orientation-dependent problems. The surface of the unit sphere can be viewed as the set of all possible 3D directions, and mappings of any surface on the sphere can be defined in an obvious way:

- **The Gaussian Map** (G_{Map}) [8,9] assigns each point of a surface F to the surface's normal at this point. Normals are depicted as points on the unit sphere. These points can be seen as the intersections of the spherical surface with the rays centered at the sphere and pointing in the directions of the surface's normals. Formally, $G_{Map} : F \to S^2$ so that $\forall x \in F, G_{Map}(x) = N(x)$, with $N(x) \in S^2$ the normal of F at x.
- **The Visibility Map**(V_{Map}) assigns to a given surface the set of directions based at infinity from which the whole surface is visible (i.e., unobstructed rays can be drawn to every point on the surface). Alternatively, the V_{Map} is the set of points on S^2 that differ from every point of a G_{Map} by at most $\pi/2$ [10,11].

Both maps and their relationship are depicted in Figure 1.

2.2 Applications

Applications where such representations are useful span such different areas as machineability of workpieces, assembly planning, and robot path planning.

Machineability of objects

Many problems of this type are related to the machineability [1] of a surface, which is a generalization of the notion of visibility. The set of directions along which a tool can machine a given surface entirely without being obstructed by the surface itself can be represented as a spherical polygon [12–14]. If the tool is a ball-end cutter, the directions along which a surface element (or a planar surface) is machineable are those in the hemisphere centered on its normal; therefore, the spherical polygon corresponding to the entire surface is the intersection of those hemispheres, which is exactly the V_{Map}. Thus, for this type of cutter, the notions of machineability and visibility coincide.

In general, suppose that a workpiece with several surfaces to be machined is placed on an NC machine equipped with a tool. The *workpiece-tool orientation problem* is that of determining how the workpiece should be oriented so that the maximum number of surfaces can be machined. Again in the case of a ball-end cutter, and for an NC machine with three translational degrees of freedom (dof), any orientation contained in the maximum number of V_{Map}s would do. This is equivalent to finding the densest hemisphere of the G_{Map} [13]. If the NC machine has four dofs (respectively, five dofs) the solution is to find the great arc (resp., the spherical band) intersecting as many V_{Map}s as possible [13]. Furthermore, finding a *separator* of the set of spherical polygons corresponding to the surfaces' machineabilities, permits machining all the surfaces under only two opposite orientations of the workpiece.

In the case of a fillet-end cutter, $V_{Map}s$ are now *arc-polygons*, i.e., intersections of small circles on S^2, as shown in Figure 2. An algorithm is provided in [14] for solving the five (and implicitly the four) axis machining problem.

[1] Here machineability is a local concept, in the same way as the notion of visibility in the preceding section.

Assembly planning

Consider two pieces in an assembly, and consider a given direction along which one piece blocks the other as an infinitesimal translation is attempted in 3D. This blocking relationship between two pieces can be depicted as an arc pointing from the node representing the blocked piece to the node corresponding to the blocking one. The blocking relationships between all the pieces in an assembly for a given direction can be represented by a *directional blocking graph*. These relationships hold over a whole set of directions, which correspond to a region on S^2. The arrangement of all such regions, each one with its associated directional blocking graph, constitutes a *non-directional blocking graph (ndbg)* [15]. Figure 3 displays the *ndbg* corresponding to a simple planar assembly of three pieces (in this case, regions are points and arcs on S^1).

As only face-face contacts are considered, such regions are clearly V_{Maps} in the sense expressed above: recall that the V_{Map} corresponding to a planar face is the hemisphere whose pole is the face normal, and the set of all non-blocking directions in a face-face contact is exactly such hemisphere. Furthermore, the regions in the *ndbg* are the spherical polygons that result from the intersection of the hemispheres associated to all the contacting faces of two subassemblies, i.e., V_{Maps} of sets of faces.

If infinite translations (in 3D) are considered instead, the arrangement on S^2 arises now from the intersection of the spherical surface with the polygonal cone of all the rays drawn from the origin and intersecting the Minkowski difference of the two subassemblies [15,16].

Robot path planning

A well-known concept in Robotics is that of *Configuration space (C-space)*, which allows to reduce the geometric motion planning problem of a robot amidst obstacles to determining a collision-free path of a point amidst *C-obstacles*. When the robot is only allowed to translate, the C-obstacle corresponding to all the configurations where the robot intersects an obstacle can be obtained by computing their Minkowski sum. Efficient algorithms exist for computing such sum in the case of polygonal and polyhedral robots and obstacles. One such algorithm [17] in fact computes implicitly the Minkowski sum by performing the *convolution*[2] of two so called *polyhedral tracings*. A tracing extends the concept of polyhedron by adding a *whisker map*, which assigns to each feature a point, an arc or a region on the spherical surface (i.e., a set of directions) [18,17]. Thus, again, S^2 is the space where a relationship between two objects is most clearly expressed, for the convolution of two polyhedral tracings involves determining which features are coincident in their whisker maps. The kind of CG problems that arise in this context are those of determining red-blue arc intersections and node-in-region inclusions.

The problem presented in this paper is formulated in similar, although simpler terms than in the case of the convolution. As explained below (Section 3), some polyhedral features like concave edges and a wide class of non-convex vertices do not need to be considered in our application, while they need to in [17].

[2] The convolution subsumes the Minkowski sum of two polyhedra when they are convex, and it is combinatorially much simpler to compute.

The previously described problems are solved either by projecting on the euclidean plane, and then applying well-known planar algorithms, or directly on the spherical surface. In the latter case, it is also often possible to adapt a planar algorithm to the particularities of the spherical setting.

2.3.1 Projection on the euclidean plane

Central projection allows to map spherical features on the euclidean plane, where most computational geometry algorithms have been developed. After projecting, two possible strategies may be followed: either the problem is completely solved in the plane, where the output obtained from running the planar algorithm admits an immediate interpretation in terms of the spherical equivalent, or, after a preprocess in the plane, the setting is backprojected onto S^2 and the solving process is completed there.

Examples of the first strategy are to be found in the algorithms that solve the four-axis NC machine problem, either by reducing it to finding the line that intersects the maximum number of (planar) polygons, passing through given points [12] (total $O(vn^2 + n^3 log n)$ time for n polygons and v vertices), or by combining duality and topological sweeping [14] ($O(v^2)$).

The second strategy is followed in [13], also in the context of the three- and four-axis NC machining problems. First, the spherical polygons are projected onto the plane, where a simple test indicates whether the original spherical polygon is completely contained or not in a hemisphere, and where classical algorithms can be applied for computing convex hulls of sets of points [10]. These convex hulls are then backprojected on the sphere, thus obtaining spherical convex hulls. Arrangements of their duals serve as input to the algorithms that solve the specific NC machining problems.

2.3.2 Solving directly on the sphere

An arrangement of regions is built directly on the sphere if the nature of the involved geometric entities renders its projection on the plane unmanageable. This happens in the case of the five-axis NC machine problem, as small circles (like those limiting the spherical band) do not project on straight lines [14]. Here, an arrangement of regions defined by intersections of disks (the surface of spherical sectors) is computed, and algorithms are provided for determining efficiently the number of regions covering each element of the arrangement. Four and five-axis machining problems with fillet-end cutters have obviously the same limitation, and [14] provides the means for computing efficiently the arrangement of the *feasible regions* attached to the arc-polygons. A note concerning implementation: [19] provide an exact representation of points, arcs and regions on the sphere, as well as exact formulas for basic computations like the intersection of two circles or the circular ordering of arcs around a circle.

There are also algorithms devised to run directly on arrangements on the sphere, despite the fact that the type of involved geometric entities would allow to resort to central

projection (like minor arcs on great circles). They are based on the observation that planar algorithms can be adapted to S^2 with little additional effort. An example of this approach can be found in [20], where a planar arrangement computation package, based on trapezoidal decomposition, is used with minor modifications to handle arrangements on the sphere. Another example is the use of a plane sweep algorithm [21] for solving the convolution computation problem [17] described at the end of the preceding section.

Our strategy belongs to this last kind. In fact, we describe an alternative application of the algorithm in [21], to lower the computational effort needed for determining interference between the boundaries of two polyhedra.

3 A new application: pairing of features for interference detection

Interference between two initially disjoint polyhedra may be checked by testing whether the edges of one polyhedron intersect the faces of the other one. Most interference tests require faces to be convex [22]. The edge-face intersection test we use [23,24] works for non-convex faces, thus avoiding not only the costly preprocessing step of decomposing them into convex ones, but also having to consider for interference all the edges and faces arising from this decomposition. Although there still remain many edge-face intersection tests to perform, their number can be considerably lowered, as we show next.

Consider two initially disjoint polyhedra which are allowed to translate. As their relative orientation remains unchanged, only certain pairs of features of their boundaries (vertices, edges and faces) may eventually get in contact. Such contacts are said to be *applicable* [25]. It is not difficult to see that applicability and machineability are two interpretations of the same concept, that of accessibility or reachability between the surfaces of two objects. If polyhedron T is regarded as a tool, and polyhedron W as a workpiece, then the set of orientations of T for which a given vertex can machine a given face of W is obviously the same for which the contact between these two features is applicable.

The edge-face pairs that need to be considered for intersection are restricted to those which are related through applicable contacts:

- If a given vertex-face contact is applicable, a candidate pair for intersection test consists in the face and any one of the edges adjacent to the vertex.
- If the applicable contact is an edge-edge type, any one of the edges and any one of the faces adjacent to the other edge is a suitable candidate pair.

Note that this is an infinitesimal approach, and time sampling may have an effect on its accuracy. However, since only translations are allowed, applicable contacts can be parameterized [25], leading to linear equations that can be analytically solved to obtain tentative, but very accurate, collision times.

Applicable contacts can be determined in a preprocessing step. The results obtained from this computation are valid as long as the relative orientation between the polyhedra does not change or changes within certain limits. The local nature of applicability leads to a conservative strategy in the non-convex case, in the sense that edge-face candidates may arise that can never intersect first when the polyhedra collide. However, except for very

specific polyhedra whose shapes will hardly arise in practical applications, the amount of pruning will be much more significative than the number of false candidates.

It remains to perform this preprocessing step efficiently. To determine all the applicable contacts by testing the applicability constraints for all vertex-face and edge-edge pairs has a quadratic complexity. A more efficient solution is attained by using a proper representation, as described in the next section.

4 The proposed representation: the Spherical Face Orientation Graph

A compact representation of all applicable contacts is obtained by using the **Spherical Face Orientation Graph** (SFOG for short), developed by the authors. The SFOG extends the concept of G_{Map} by representing not only faces but also edges and vertices, and their adjacencies:

- Faces are represented by **nodes** on the sphere of orientations. As in the G_{Map}, the node represents the orientation of the outward normal of the plane supporting the face.
- Edges are represented by **arcs**. These arcs join nodes that correspond to faces sharing an edge, and lie on great circles of the sphere (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 (which will be called *convex* arc), concave edges by the major arc (*concave* 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.
- A vertex is represented by the **region** enclosed by 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). Non-convex vertices have at least one adjacent concave edge. For a class of non-convex vertices, a pyramid can be defined locally by selecting a subset of adjacent convex edges, such that every other adjacent edge is contained inside. This pyramid is called the *local convex hull* of a *pseudo-convex* vertex. The intersecting convex arcs that correspond to this subset bound a so-called *convex subregion (csr)* on the sphere. Note that, for non-convex polyhedra, regions corresponding to different vertices may intersect.

The SFOG is quite similar to the *whisker map* [17] mentioned at the end of Section 2.2. Common features are duality, the use of the spherical surface as representation space, and the possiblity of multiply covering S^2 (overlapping of regions corresponding to different vertices). However, there are also significant differences: in the whisker map, arcs are always less than π, and need to be labelled in order to be identified as convex or concave. Furthermore, all types of vertices have an associated region enclosed by a ring of edges, not only convex or pseudo-convex ones as in the SFOG. These differences arise from the fact that convolution is performed over the whole set of polyhedral features, whereas applicability makes no sense with concave edges and most types of non-convex vertices.

By superimposing the SFOG of one polyhedron on the central symmetric image of the SFOG of another polyhedron (see Figure 4 (I)), the vertex-face and edge-edge applicability relationships can be directly determined:

(1) (Convex case) 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 (Figure 4 (II-1)).
(2) (Non-convex case) A given node falls into a certain convex subregion (*csr*) if and only if the contact between the vertex whose local convex hull is represented by the *csr* and the face represented by the node is *locally* applicable (Figure 4 (II-2)).
(3) Two convex arcs of different SFOGs intersect if and only if the contact between the corresponding edges is (*locally*, in the non-convex case) applicable (Figure 4 (II-3)).

Observe again that a spherical region can also be seen as the set of possible directions in which a face can be machined by a tool with the geometry of the corresponding vertex.

The problem is reduced to determining all these node-in-region inclusions and intersections between arcs efficiently. If both polyhedra are convex, an *ad hoc* algorithm that exploits connectivity and which is linear both in the input and the output can be used [26]. In the general case, concave arcs are better eliminated, as no useful information concerning applicability is attached to them, and connectivity cannot be exploited as in the convex case. Thus, the following setting has to be dealt with: a spherical surface covered by *red* and *blue* regions, nodes and arcs, where all bichromatic intersections (between nodes and regions, and between arcs) have to be found (each colour stands for one polyhedron). The regions of each colour cover completely the sphere and may even overlap. On the other hand, arcs may be disconnected from other arcs, and even isolated nodes may appear.

Computing the convolution of two polyhedral tracings does also mean to detect all these intersections [17]. Nonetheless, both computations differ in various aspects: first, other types of pairings (such as vertex-vertex or vertex-edge) have to be computed in the convolution case as well. Furthermore, concave edges (and all types of vertices) have also to be considered. And this leads to the final difference, namely that connectedness is preserved and exploited, whereas it could be lost in the applicability computation case due to the elimination of concave arcs.

In Section 2.3, different strategies were presented for dealing with computational geometry problems on the sphere. The geometrical nature of the objects involved in our application (arcs on great circles, regions delimited by such arcs, and nodes) would allow to use central projection for mapping them onto the plane, and solving the problem as the computation of the intersection between the corresponding mapped segments, regions and points. However, besides the cost associated with the projection (considering also that arcs traversing the equator are splitted into two semiinfinite lines), roundoff errors may appear for arcs which are nearly coincident with the equator, and possible intersections may be missed (or falsely reported) due to these errors.

The alternative consists in solving the problem directly on the sphere, by adapting a planar algorithm. This is the approach followed here. In particular, we have developed

and adapted planar sweep line strategies to the sphere.

5 Line sweep algorithms on the sphere

5.1 Naïve algorithm

It is straightforward to adapt the plane sweep principle to the sphere: the vertical sweep-line is replaced by a sweep-meridian, the sweep begins at an arbitrary point (as we cannot speak of a "leftmost" point), and proceeds eastwards (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 (recall that arcs larger than π have been removed, since they correspond to concave edges), 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 colour $L_{\overline{c}[i]}$, and included in the list of active arcs $L_{c[i]}$ ($c[i]$ denotes the colour 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.

As said 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 5 depicts the purple intersections that can be detected along the first sweep and those which 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 colour 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 if the regions underneath these arcs are bounded by arcs that cut the sweep-line below that node.

5.2 Red-blue blocks algorithm

The algorithm shown in the previous section is easy to implement and works well in practice. However, as complexity of the setting grows, small increases in efficiency using more sophisticated data structures and procedures lead to large benefits in the overall performance. Attempts at lowering the computational effort of computing the intersections of two sets of line segments have been reported in the literature, which could be translated into the spherical setting.

Two basic formulations can be distinguished, whether each set of segments is disjoint or not. For settings without monochromatic intersections, sweep-based algorithms [27,28] as well as algorithms based on segment-tree-like data structures [29,30] exist, that perform in optimal $O(k_{r-b} + n_t \log n_t)$ time and need $O(n_t)$ space ($n_t = n_r + n_b$).

The harder problem of determining all bichromatic *purple* intersections, avoiding the costly computation of all monochromatic intersections (in the worst case, the number of monocromatic intersections can be quadratic) is treated in [31], where an algorithm is described that combines line sweep with ray-shooting techniques, reporting all purple intersections in time $O((n_r\sqrt{n_b} + n_b\sqrt{n_r} + k_{r-b}) \log(n_r + n_b))$. This was improved in [32] to overall $O(n_t^{4/3} \log^{(\omega+2)/3} n_t + k_{r-b})$ time and using $O(n_t^{4/3} / \log^{(2\omega+1)/3} n_t)$ space, where ω is a constant < 3.33, by using a divide-and-conquer strategy. These algorithms are deterministic; using random-sampling techniques, an expected time of $O(n^{4/3} \log n + k_{r-b})$ is obtained [32].

The wise use of a recently developed data structure -called *heater*- permits performing a line sweep in expected time $O((n + k_{r-b})\alpha(n) \log^3 n)$ in the case where the sets of red and blue segments are connected [21] (this result can even be lowered by a logarithmic factor [33,34], although the associated data structures may be hard to implement [35]). In our case, due to the loss of connectivity as concave arcs are eliminated, the complexity is $O((c_b n_r + c_r n_b + k_{r-b}) \log^3 n\alpha(n))$, as reported in [21] when red and blue segments are grouped into c_r and c_b connected components. *Heaters* combine the features of binary trees and heaps: they are heaps in element keys (which, in the present case, encode the heights at which the segments intersect the sweep line), and search trees on random search keys [3] . Their use is explained below.

As in every line sweep strategy, the endpoints of the arcs (BEGIN and END events) are scheduled first. At a given instant, sets of red and blue arcs will be intersecting the sweep line. The novelty and main contribution of this approach consists in considering these red and blue arcs grouped into blocks, implemented as heaters, so that only the bottom segment has to be tested for intersection against the top segment of the contiguous block. If such a test reports intersection, a bichromatic (PURPLE) event is scheduled, which belongs to the output of the algorithm. It must be guaranteed 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, also some monochromatic intersections have to be scheduled (called INTERNAL events). The key point is that only those monochromatic intersections that affect parental relationships within the heaters have to be scheduled, and they are only a subset of the whole.

The scheduling of bichromatic intersections (PURPLE events) triggered by the different events along the sweep is shown in Figure 6.

The event point schedule, implemented as a global priority queue, contains all four types of events, and is updated conveniently (for example, a BEGIN event may schedule a PURPLE event somewhere in the future). As the next element (the next event) is extracted from the queue, it is classified according to its type and the corresponding subroutines are triggered (a "case of" structure is the most natural implementation):

[3] As pointed out in [21], this data structure is very similar to the random treap [36].

- BEGIN events mean the insertion of the beginning arc in an existing block of the same colour or the creation of a new block and possibly the splitting of an opposite colored block. In the latter case, a PURPLE event may be scheduled, and also if the new arc is on top or bottom of a block.
- An END event means the deletion of the arc from the corresponding block. Possibly a new PURPLE event has to be scheduled if this arc was at the top or bottom of its block, or, if the block consisted only in this ending arc, then possibly the neighboring blocks have to be merged together. These arcs insertions (in BEGIN events) and deletions (in END events) schedule also possibly new INTERNAL events.
- INTERNAL events require the performance of *rotations*, i.e. parental relationship modifications inside the heater, and possibly the scheduling of PURPLE events if the INTERNAL event affects the block's roots.
- PURPLE events store the crossing arcs in a list for output. Different situations arise depending on whether the size of the blocks to which these arcs belong is one or greater, and the presence or not of neighbor blocks. Some of these situations possibly mean the scheduling of new PURPLE events. The crossing arcs may create new blocks, or be included in existing ones (and deleted from others), which implies the scheduling of INTERNAL events.

The essential information about the blocks where a new arc has to be inserted, as well as which blocks are above and below a given one, is quickly retrieved by maintaining a balanced search tree that stores top and bottom elements of each block. Obviously, this information may have to be updated when some events occur.

This algorithm can be adapted to the sphere following the same guidelines as for the naïve algorithm, in order to compute all purple arc intersections. The node-in-region inclusions can be managed within the same sweep, by keeping, for each block, a list of regions currently including it (along the sweep meridian). Nodes lie at the endpoints of the arcs, and they can be included in the regions corresponding to the block where the arc is going to be inserted (BEGIN event) or where it has been deleted from (END event). New blocks may appear at BEGIN and PURPLE events, they inherit the regions lists of the adjacent blocks of the same colour, with some modifications that can be computed locally from the adjacent blocks of the other colour.

Further details and pseudo-code are given in [37].

6 Experimental results

6.1 Comparison between the naïve and the red-blue blocks algorithms

A first set of experiments was aimed at assessing the benefits and drawbacks of using the intrincate red-blue blocks algorithm, instead of the naïve one, in practice.

Since the goal of both algorithms is to determine all the bichromatic intersections between arcs, a natural performance measure is the total number of arc intersection tests that have to be performed in order to determine such bichromatic intersections.

As described above, the naïve algorithm requires to test each arc with all the arcs in the list of currently active arcs of the opposite colour, while the search of bichromatic intersections is restricted, in the case of the red-blue blocks algorithm, to the top and bottom arcs of the neighbouring blocks. However, the naïve algorithm avoids completely to test monochromatic intersections, while the maintenance of the data structures of the red-blue blocks algorithm requires to compute some monochromatic intersections inside each block. These monochromatic intersection tests arise during the insertion and deletion operations, as well as in the INTERNAL events themselves. Thus, the number of bichromatic intersection tests of the naïve algorithm has to be compared with the number of bichromatic plus monochromatic intersection tests of the red-blue blocks algorithm.

Both algorithms have been implemented in C, and tested on several pairs of polyhedra, yielding the results presented in Figure 7. The number of all possible edge - edge pairings has been chosen as a measure of the complexity of the setting (in abscissae). Only a subset of the corresponding arc intersections (plus some monochromatic intersections in the blocks approach) have to be computed by the two algorithms. For objects of low complexity and a relatively high number of INTERNAL events, as is the case of two star-shaped hexadecahedra (on the left in Figure 8) - a total of 576 edge pairings - the naïve algorithm performs better. At the other extreme, there are two hourglass-shaped polyhedra (Figure 8, right) - a total of 26244 edge pairs - where, despite being non-convex, no INTERNAL events occur (overlapping arcs are not considered as intersecting). Nonetheless, it is important to notice that although no monochromatic intersections occur, the tests have actually to be performed during insertions and deletions in the heaters. Therefore, the increasing difference in performance between the two algorithms as the number of edge-edge pairings increases is clear.

It is worth noting that the performance differences are not so high in terms of CPU time, due to the computational cost of creating and maintaining the data structures that are necessary in the red-blue blocks algorithm. To determine precisely the point beyond which the blocks algorithm outperforms the naïve one would require an optimized implementation and a careful experimentation that is beyond the scope of this paper. In our implementation, the turning point occurs around settings with ten thousand edge-edge pairs.

6.2 Performance of the collision checker with orientation-based preprocessing

Previous experiments showed that interference detection between polyhedral models based on the edge-face intersection test performs 10 to 100 times faster if a previous selection of candidate pairs based on applicability is carried out [1]. Although highly dependent on the specific geometry of the involved polyhedra, it can also be stated that these savings increase proportionally to the complexity of the setting.

Here we wanted to determine how our collision checker with orientation-based preprocessing compared to publicly available state-of-the-art packages, such as RAPID 2.01 [2] and PQP 1.2 [3] in the assembly domain. Thus, we devised an assembly setting with scalable pieces, such as those shown in Figure 9. Table 1 shows the runtimes of the three checkers for a situation where the protuberances of polyhedron A are deeply inserted (with a given tolerance) but not touching the inner walls of the holes in polyhedron B. Note that

our checker compares favourably to RAPID 2.01 and is compareable to PQP 1.2. These and similar results obtained for other configurations lead us to the conclusion that our orientation-based approach may be advantageous for workpieces with many concavities that are in close proximity and can give rise to multiple contacts, and it is advisable to explore its combination with other strategies based on bounding volume hierarchies.

7 Conclusions

To identify the feasible contacts between two polyhedral models with a given relative orientation is a basic step in the design process. Furthermore, this computation can be viewed as a preprocess to reduce the number of edge-face intersection tests needed to detect interference, when the polyhedra are allowed to translate freely. This preprocessing can be formulated as a computational geometry problem on the sphere, like a wide class of machining and assembly operations. To this end, a representation of polyhedral features and their relationships on the unit sphere of orientations, named SFOG, has been developed. By combining two SFOGs, a compact structure that captures all feasible (applicable) contacts between them is obtained. Expressed in this way, the detection of feasible contacts belongs to the kind of problems that can be solved directly on the sphere by adapting planar algorithms.

Two such algorithms, based on a line sweep, are described and compared here. On the one hand, there is the naïve algorithm where each arc introduced at a BEGIN event is tested for intersection with all the arcs of the opposite colour which are currently active (i.e., intersecting the sweep line). On the other hand, the red-blue blocks approach uses sophisticated data structures to determine quickly where the arc has to be introduced in the blocks structure, thus confining the tests for intersection with this arc to the neighbouring blocks. Experimental comparison shows an increasing advantage of the red-blue blocks approach, as the complexity of the setting grows.

Moreover, the collision checker with orientation-based preprocessing has shown its potential in tight assembly situations where multiple contacts may arise, as compared to state-of-the-art approaches. Thus, a promising line of future research concerns the inclusion of orientation-related information in a hierarchical structure, in an analogous way as the *spatialized normal cone hierarchy* [38]. Combining both volumetric and orientation-dependent information may multiply the benefits attached to each approach.

Continuing with future work, note that the applicability of contacts is orientation-dependent, which means that if the polyhedra move along trajectories that entail a change in the relative orientation, new feasible contacts arise, while others disappear. Therefore, one must be able to determine the *intervals of isoapplicability*, i.e., the ranges of relative orientations –along the trajectory– for which the same applicability relations hold. In a trajectory parameterization approach, this entails 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*, i.e., points in time when a node of one SFOG crosses an arc of the other one. This question is ad-

dressed in [26], 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.

Aknowledgments

This research has been partially supported by the Spanish Science and Technology Commission (CICYT) under contract TAP99-1086-C03-01 (project "Constraint-based computation in robotics and resource allocation") and the Catalan Research Commission, through the "Robotics and Control" group.

References

[1] P. Jiménez, C. Torras, An orientation-based pruning tool to speed up interference detection between translating polyhedral models, International Journal of Robotics Research 20 (6) (2001) 466–483.

[2] S. Gottschalk, M. C. Lin, D. Manocha, Obb-tree: A hierarchical structure for rapid interference detection, in: Proc. of ACM Siggraph'96, New Orleans (LA), 1996, pp. 171–180, http://www.cs.unc.edu /~geom/OBB /OBBT.html.

[3] E. Larsen, S. Gottschalk, M. Lin, D. Manocha, Fast proximity queries with swept sphere volumes, Tech. Rep. TR99-018, Dep. of Comp. Sci., UNC Chapel Hill, http://www.cs.unc.edu/~geom/SSV/ (1999).

[4] K. Brown, Geometric transformations for fast geometric algorithms, Ph.D. thesis, Dept. of Computer Science, Carnegie Mellon Univ. (1980).

[5] A. Okabe, B. Boots, K. Sugihara, S. N. Chiu, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, John Wiley and Sons, 1992.

[6] V. Sacristan, Geometric optimization and applications in visibility (in spanish), Ph.D. thesis, Dept. de Matemàtica Aplicada II, Universitat Politècnica de Catalunya (1997).

[7] C. Grima, A. Marquez, Computational Geometry on Surfaces, Kluwer, (to appear).

[8] D. Hilbert, S.Cohn-Vossen, Geometry and the Imagination, Chelsea, 1987.

[9] B. K. Horn, Extended gaussian images, in: Proc. of the IEEE, Vol. 72, 1984, pp. 1671–1686.

[10] L.-L. Chen, T. C. Woo, Computational geometry on the sphere for automated machining, ASME J. Mech. Des. 114 (2) (1992) 288–295.

[11] J. G. Gan, T. C. Woo, K. Tang, Spherical maps: their construction, properties and approximation, ASME J. Mech. Des. 116 (2) (1994) 357–363.

[12] K. Tang, T. C. Woo, J. G. Gan, Maximum intersection of spherical polygons and workpiece orientation for 4- and 5-axis machining, ASME J. Mech. Des. 114 (3) (1992) 477–485.

[13] L.-L. Chen, S.-Y. Chou, T. C. Woo, Separating and intersecting spherical polygons: computing machinability on three-, four, and five-axis numerically controlled machines, ACM Transactions on Graphics 12 (4) (1993) 305–326.

[14] P. Gupta, R. Janardan, J. Majhi, T. Woo, Efficient geometric algorithms for workpiece orientation in 4- and 5-axis nc-machining, in: S. G. Akl, F. K. H. A. Dehne, J.-R. Sack, N. Santoro (Eds.), Algorithms and Data Structures, 4th International Workshop, WADS '95, Vol. LNCS,955, Springer Verlag, Kingston, Ontario, Canada, 1995, pp. 171–182.

[15] R. Wilson, J.-C. Latombe, Geometric reasoning about mechanical assembly, Artificial Intelligence 71 (2) (1994) 371–396.

[16] D. Halperin, J.-C. Latombe, R. H. Wilson, A general framework for assembly planning: The motion space approach, in: Proc. 14th Annual Symp. on Computational Geometry, Minneapolis, Minnesota, 1998, to appear in *Algorithmica*.

[17] J. Basch, L. J. Guibas, G. D. Ramkumar, L. Ramshaw, Polyhedral tracings and their convolution, in: Workshop on Algorithmic Foundations of Robotics, 1996.

[18] L. Guibas, R. Seidel, Computing convolution by reciprocal search, in: Proc. of the ACM Symp. on Comp. Geom., Yorktown Heights (NY), 1986.

[19] M. V. A. Andrade, J. Stolfi, Exact algorithms for circles on the sphere, in: Symposium on Computational Geometry, 1998, pp. 126–134.
URL `citeseer.nj.nec.com/326577.html`

[20] M. Goldwasser, An implementation for maintaing arrangements of polygons, in: Proc. 11 th Symp. on Comp. Geometr, ACM, Vancouver, B.C., Canada, 1995, pp. C32–C33.

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

[22] P. Jiménez, F. Thomas, C. Torras, Collision detection: a survey, Computers and Graphics 25 (2) (2001) 269–285.

[23] F. Thomas, C. Torras, Interference detection between non-convex polyhedra revisited with a practical aim, in: Proc. IEEE Int. Conf. on Robotics and Automation, Vol. 1, San Diego (CA), 1994, pp. 587–594.

[24] F. Thomas, C. Torras, A projectively invariant intersection test for polyhedra, The Visual Computer 18.

[25] B. R. Donald, A search algorithm for motion planning with six degrees of freedom, Artificial Intelligence 31 (3) (1987) 295–353.

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

[27] H. G. Mairson, J. Stolfi, Theoretical Foundations of Computer Graphics and CAD, Vol. F40, 1988, Ch. Reporting and counting intersections between two sets of line segments, pp. 307–325.

[28] T. M. Chan, A simple trapezoid sweep algorithm for reporting red/blue segment intersections, in: 6th Canadian Conference on Comput. Geom., 1994, pp. 263–268.

[29] L. Palazzi, J. Snoeyink, Counting and reporting red/blue segment intersections, in: Lect. Notes in Comp. Sci., Vol. Proc. 3rd Workshop Algorithms Data Struct., 1993, pp. 530–540.

[30] B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir, Algorithms for bichromatic line segment problems and polyhedral terrains, Algorithmica 11 (1994) 116–132.

[31] P. Agarwal, M. Sharir, Red-blue intersection detection algorithms, with applications to motion planning and collision detection, Siam J. Comput. 19 (2) (1990) 297–321.

[32] P. Agarwal, Partitioning arrangements of lines. ii: Applications, Discrete Comput. Geom. 5 (1990) 533–573.

[33] T. M. Chan, Dynamic planar convex hull operations in near-logarithmic amortized time, in: IEEE Symposium on Foundations of Computer Science, 1999, pp. 92–99.
URL citeseer.nj.nec.com/chan99dynamic.html

[34] S. Har-Peled, M. Sharir, Line point location in planar arrangements and its applications, in: Proc. 12th ACM-SIAM Sympos. Discrete Algorithms, Vol. 1, Washington (DC), 2001, pp. 57–66.
URL citeseer.nj.nec.com/har-peled99line.html

[35] Y. Aharoni, D. Halperin, I. Hanniel, S. Har-Peled, C. Linhart, On-line zone construction in arrangements of lines in the plane, in: Lect. Notes in Comp. Sci., Vol. Proc. 3rd Workshop on Algorithm Engineering (WAE'99), London (UK), 1999, pp. 139–153.
URL citeseer.nj.nec.com/299664.html

[36] C. R. Aragon, R. G. Seidel, Randomized search trees, in: Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci., 1989, pp. 540–545.

[37] P. Jiménez, Orientation-based pruning for collision detection, http://www-iri.upc.es/people/jimenez/mycoldec.html (2002).

[38] D. E. Johnson, E. Cohen, Spatialized normal cone hierarchies, in: ACM Symposium on Interactive 3D Graphics, ACM SIGGRAPH, 2001.
URL citeseer.nj.nec.com/446423.html

[39] A. Narkhede, D. Manocha, Fast polygon triangulation based on Seidel's algorithm, in: A. Paeth (Ed.), Graphics Gems 5, Academic Press, 1995, Ch. VII: Utilities, pp. 394–397, http://www.cs.unc.edu/~dm/CODE/GEM/chapter.html.

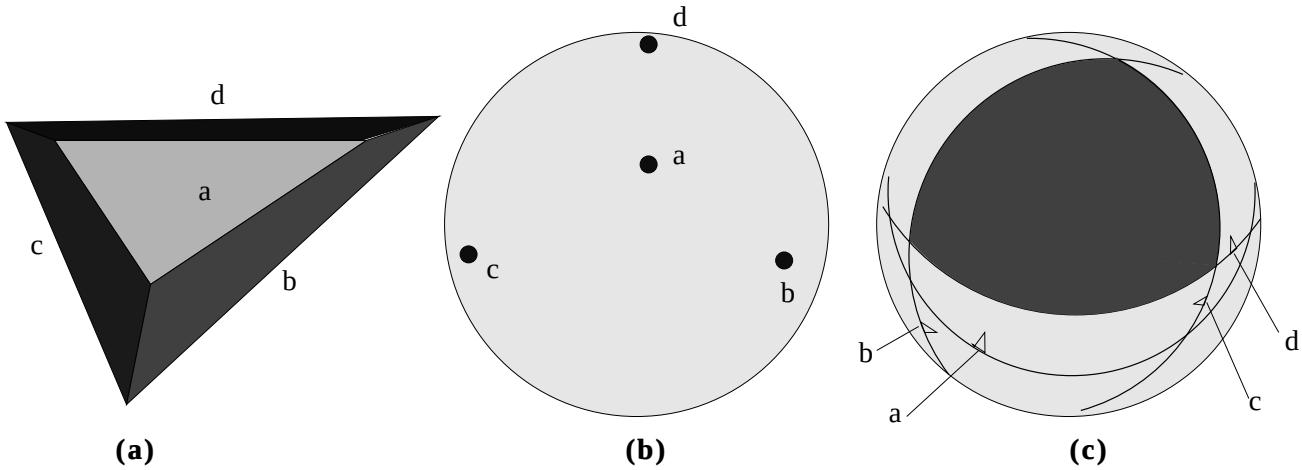

Fig. 1. *a) Four planar faces, b) their G_{Map} and c) their V_{Map}. The G_{Map} of a single face is a point on the sphere, and its V_{Map} is the hemisphere centered at that point (denoted by a small triangle on the limiting great circle). The vertices of the convex hull of the G_{Map} are "b", "c", and "d", and the V_{Map} of the four faces can be computed as the dual of the spherical convex hull [11]: the arcs on the V_{Map} are on the dual great circles of these vertices.*

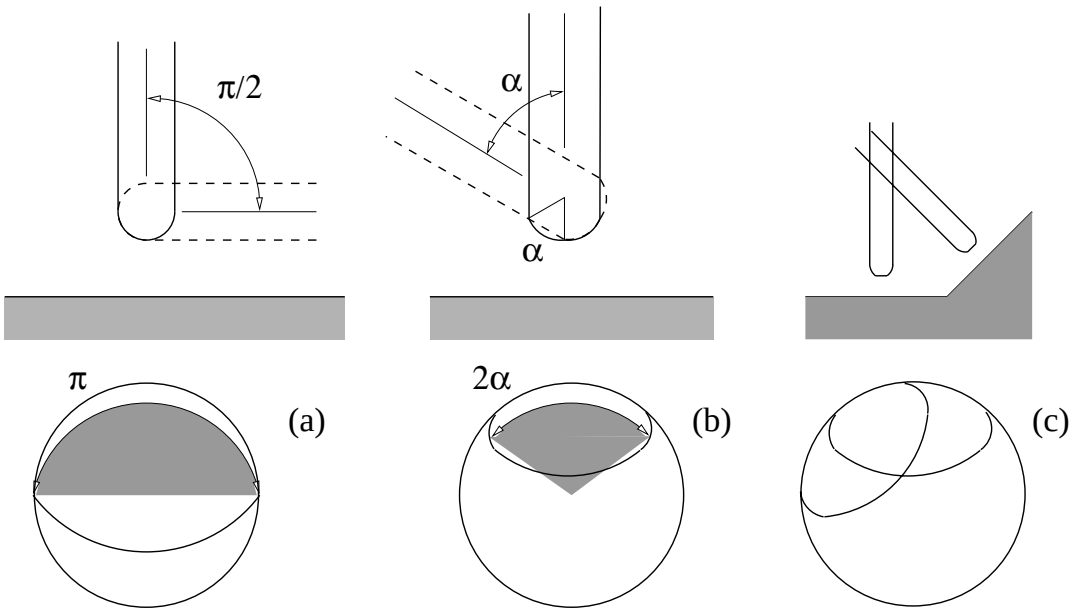

Fig. 2. *(a) The set of possible orientations along which a ball-end cutter can machine a flat surface (or a surface element) covers the whole hemisphere, (b) whereas the sector corresponding to a fillet-end cutter is smaller, depending on the tool's angle α. These regions are delimited, respectively, by a great and by a small circle. (c) The set of directions along which the fillet-end cutter can machine either of the two faces is restricted to the intersection of the sectors delimited by the two small circles.*

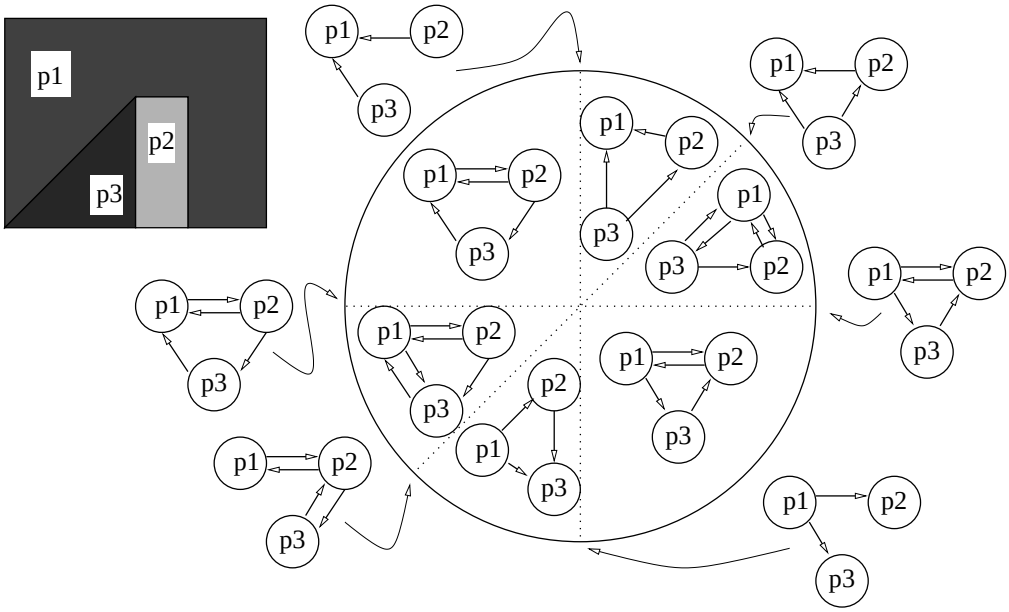

Fig. 3. *The non-directional blocking graph corresponding to a planar assembly of three pieces. In the plane, the ndbg partitions the circumference into arcs and nodes.*

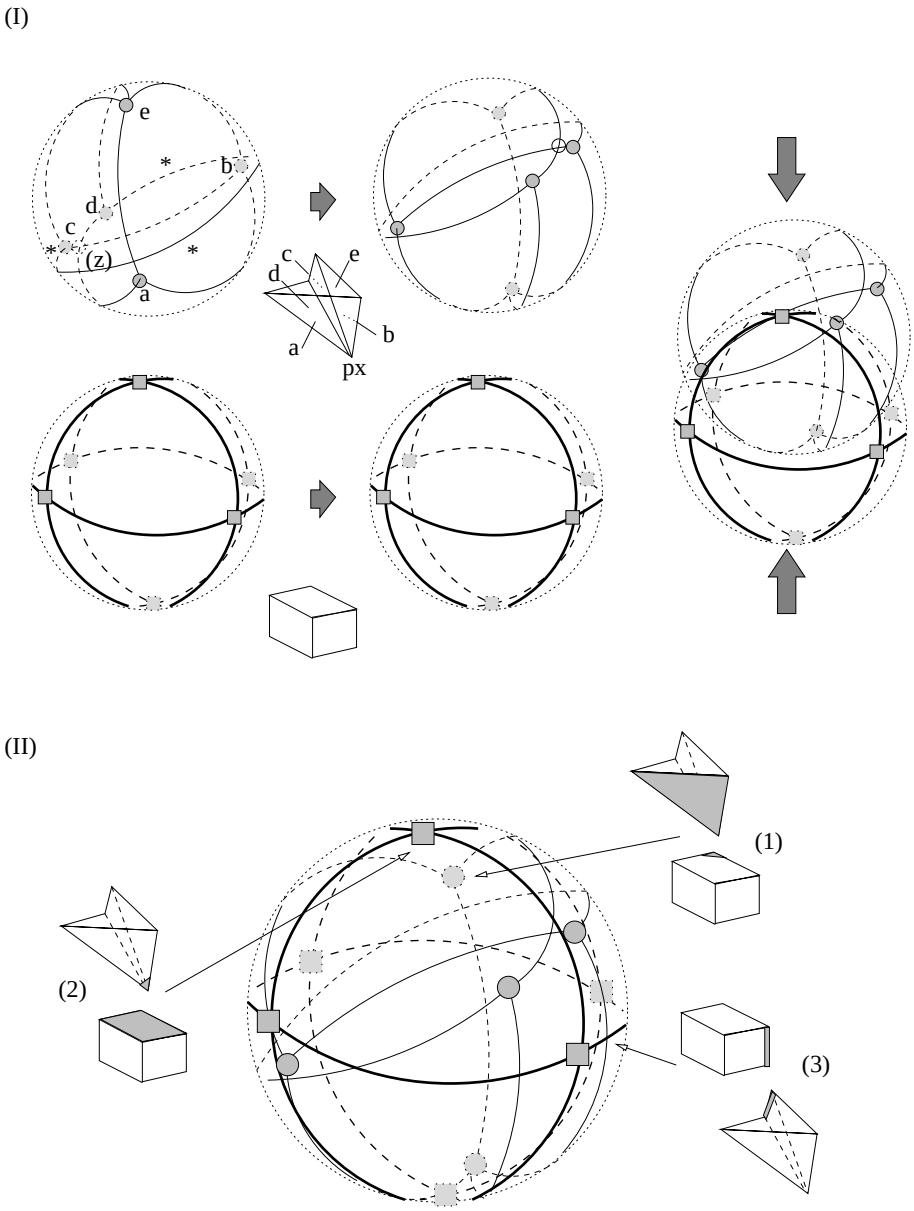

Fig. 4. *(I) Overlay of the SFOGs corresponding to two polyhedra in order to obtain a compact representation that allows to determine the applicability relationships. The SFOG of the rectangular prism below (heavy lines) is combined with the central symmetric image of the SFOG of the non-convex pentahedron above (fine lines). Observe the convex subregion attached to the pseudo-convex vertex (px), delimited by the convex arcs between the nodes a, b, and the fictitious node z (which is the intersection of arcs ad and bc). The arc representing the concave edge is marked with an asterisk. (II) Node-in-region inclusions correspond to applicable face-vertex contacts (1) and (2), and intersections between arcs to applicable edge-edge contacts (3).*

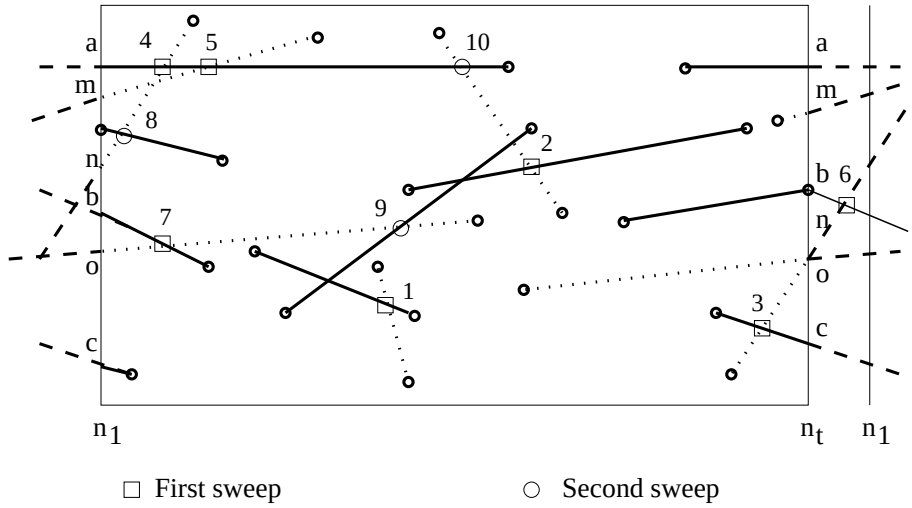

Fig. 5. *Plane analogue of the spherical sweep. Red and blue arcs are depicted as dotted and solid segments, respectively. Some of them (a, b, c, m, n, o) begin before or at the last event point n_t and end after the first one n_1. These segments are activated during the first sweep from e[1] to e[2n], where some purple intersections can already be detected (marked with a small square), and their intersections with the segments that had ended before they where generated (marked with an empty circle) can only be detected during a second sweep. Numbers indicate the order in which the intersections are computed.*

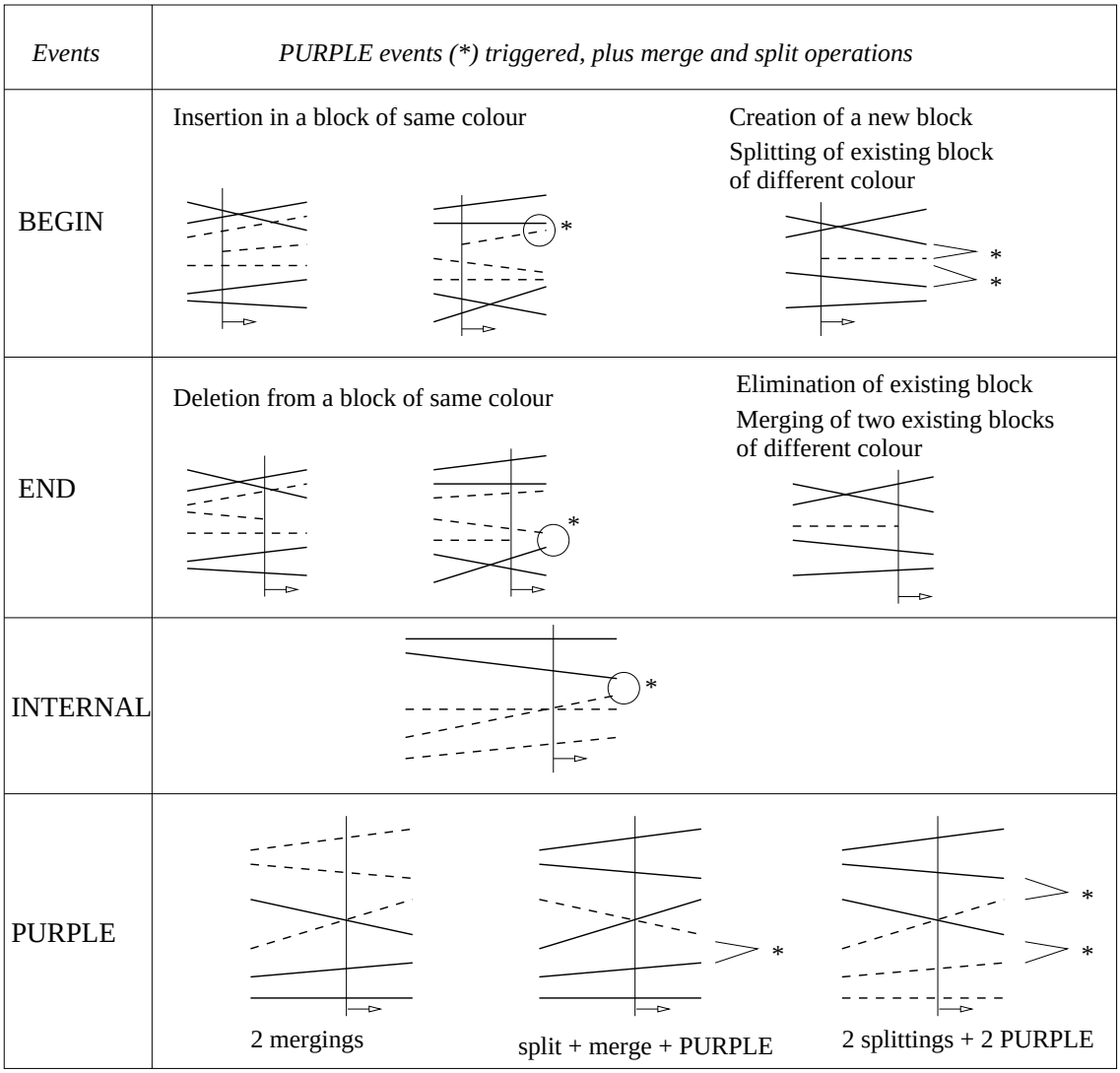

Fig. 6. *PURPLE events (*) scheduled by the endpoints of the segments (BEGIN and END events) as well as by some monochromatic (INTERNAL) and other bichromatic (PURPLE) intersections. Arcs and blocks are assumed to be in general position, the cases where they are at uppermost or lowermost position along the sweep line have to be considered specifically.*

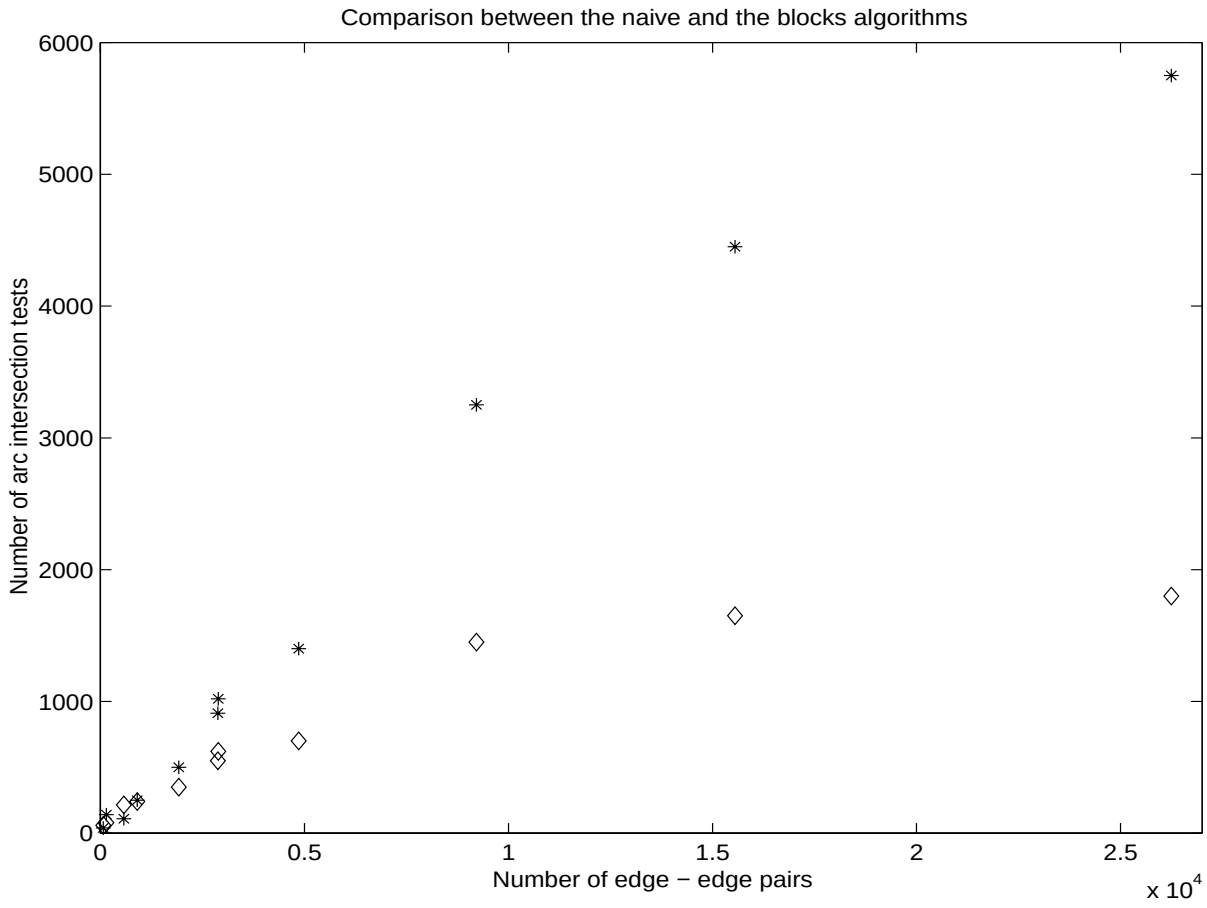

Fig. 7. *Number of arc intersection tests performed by the naïve algorithm(*), and the red-blue blocks algorithm(◇).*

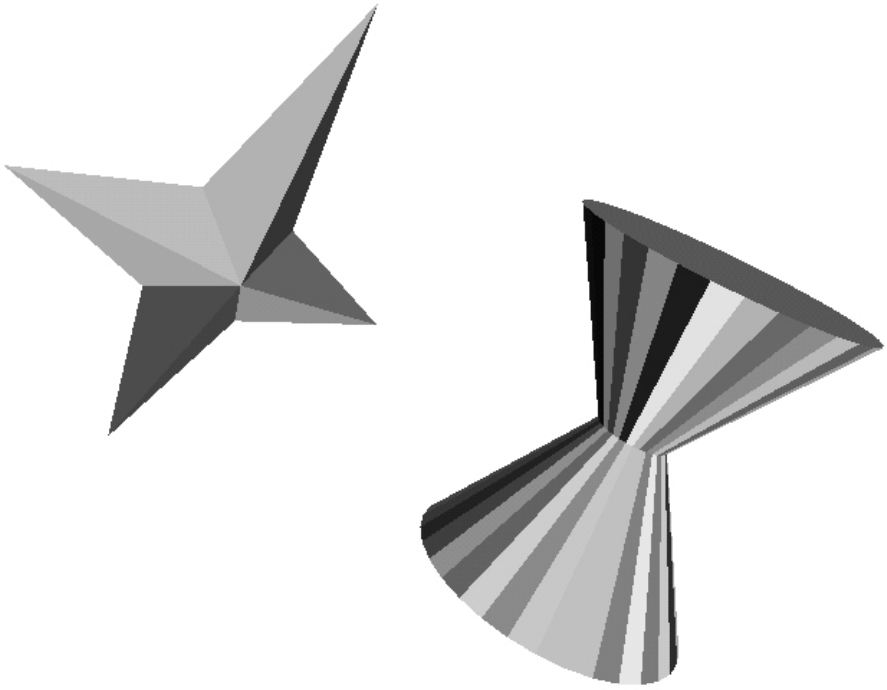

Fig. 8. *Two types of non-convex polyhedra used in the testbed: star and hour-glass.*

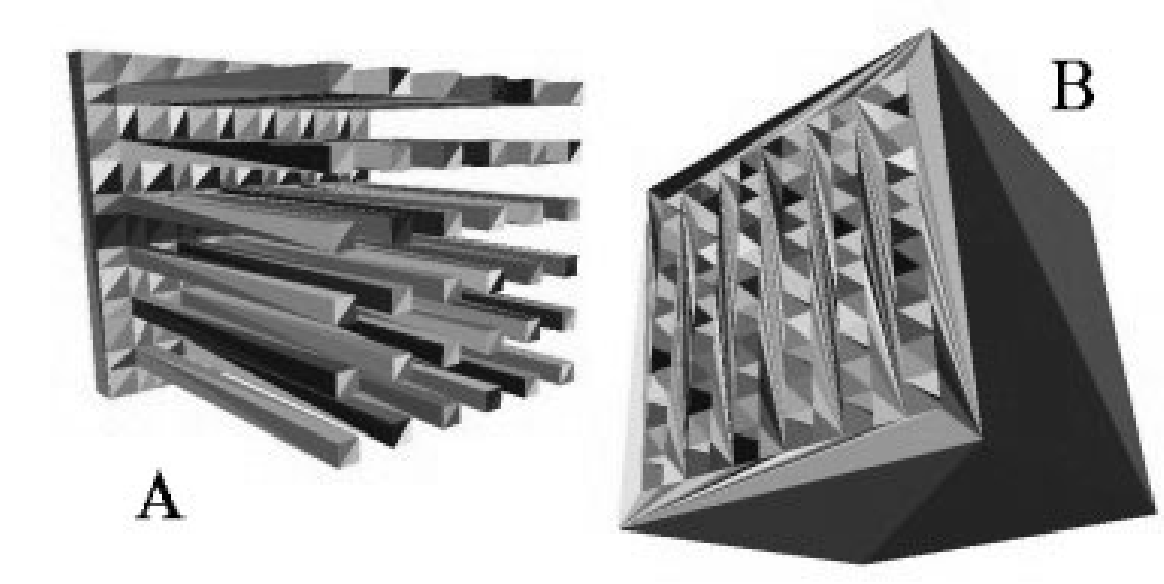

Fig. 9. *Polyhedron A with 6x6 protuberances is tested against polyhedron B (6x6 holes) in an insertion position. The polyhedra are displayed after the triangulation that has to be performed as a preprocess in order for the polyhedra to be a suitable input to RAPID and PQP, which is not necessary in our algorithm. Polyhedron A shows an ad hoc triangulation, whereas polyhedron B displays the narrow triangles produced by a standard generic triangulation algorithm [39].*

complexities	Our checker	RAPID	PQP
6x6	25	33	27
7x7	35	49	37
8x8	60	116	84
9x9	110	152	109

Table 1

Runtimes (in milliseconds) of our collision checker, RAPID 2.01 and PQP 1.2 on a Sun Ultra 80 workstation (2 ULTRASPARC II processors at 450 MHz, 1Gb RAM), for different complexities of the polyhedra. Only the interference detection part is taken into account, i.e., neither the orientation-based preprocessing, nor the triangulation and building up of the hierarchical representations are included. An ad hoc triangulation has been employed for the polyhedra in RAPID and PQP, as a standard generic one [39] generates problematic triangles (from a numerical point of view) when the complexity is bigger than 6x6. This triangulation favours the performance of RAPID and PQP, as the execution times, for the 6x6 case, after the generic triangulation, are of about 60 ms for both algorithms.