Computing the exact arrangement of circles on a sphere, with applications in structural biology

Frédéric Cazals, Sebastien Loriot

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Computing the exact arrangement of circles on a sphere, with applications in structural biology

Frédéric Cazals — Sébastien Loriot

N°6049

Décembre 2006

Thème SYM

ISSN 0249-6399

Computing the exact arrangement of circles on a sphere, with applications in structural biology

Frédéric Cazals *, Sébastien Loriot *

Thème SYM — Systèmes symboliques
Projet Geometrica

Rapport de recherche n° ???? — Décembre 2006 — 52 pages

Abstract: Given a collection of circles in a sphere, we adapt the Bentley-Ottmann algorithm to the spherical setting to compute the *exact* arrangement of the circles. Assuming the circles are induced by balls, we also extend the algorithm to report the balls covering each face of the arrangement. The algorithm consists of sweeping the sphere by a meridian, which is non trivial because of the degenerate cases and the algebraic specification of event points. The paper focuses on the construction of the arrangement, the predicates and algebraic questions being developed in a companion paper.

This construction is motivated by the calculation of parameters describing multi-body contacts in structural biology, so as to go beyond the classical exposed and buried surface areas. As an illustration, statistics featuring spectacular changes wrt traditional observations on protein - protein complexes are provided.

Key-words: Arrangement of circles, molecular surfaces, Van der Waals models

* firstname.namesop.inria.fr

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Calcul de l'arrangement exact induit par des cercles sur une sphère. Applications à la biologie structurale

Résumé : Étant donnée une collection de cercles sur une sphère, nous adaptons l'algorithme de Bentley-Ottmann pour calculer l'arrangement exact des cercles sur la sphère. En supposant que les cercles sont induit par des boules, nous indiquons comment adapter l'algorithme afin de reporter toutes les boules recouvrant une face donnée de l'arrangement. L'algorithme consiste à balayer la sphère par un méridien, ce qui est non trivial en raison des cas dégénérés et de la spécification algébrique des évenèments. Ces travaux sont motivés par le calcul de paramètres mesurant des contacts à plusieurs corps en biologie structurale, pour aller au delà des modèles classiques de surface enfouie et exposée. Dans cette perspective, deux statistiques spectaculaires sont données sur une famille de complexes protéine - protéine.

Mots-clés : Arrangement de cercles, surface moléculaire, modèle de Van der Waals

1 Introduction

1.1 Atoms, spheres, and multi-body interactions

Balls and their boundaries i.e. spheres are amongst the simplest geometric objects, and are ubiquitous in science and engineering. Among their many uses, maybe the most interesting and surprising one is found in bio-chemistry, as in modeling (macro-)molecules with Van der Waals (VdW) models, an atom is represented by a ball whose radius depends on the atomic properties and its covalent environment. The physical accuracy of a ball to model an atom is certainly limited, but as collections of thousands of atoms are not amenable to quantum mechanic calculations, VdW models provide a compromise between accuracy and tractability. Such models are actually instrumental in a number of structural biology problems, amongst which characterizing *hot spots* [RTVC04], modeling solvation and the hydrophobic effect [EM86, Cha05], investigating electrostatic properties [LFSB03], defining *scoring functions* [MJ85, AT97, GK01], etc. Not surprisingly, algorithms required by these applications were developed by bio-chemists and computer scientists, and the reader is referred to section 2 for an overview.

As most (if not all) of the classical algorithms on spheres required by the afore-mentioned applications on VdW models have reached their geometric maturity, the next stage consist of improving the bio-chemistry models, which will in turn require new geometric developments. One such cycle was possibly initiated in a recent paper dedicated to the investigation of interfaces in protein - protein complexes [CP$^+$06]. To state the problem briefly, consider the HP model where each atom is tagged as Hydrophobic or Polar. In investigating *interfaces* of protein - protein complexes in this model, its turns out that pairwise contacts across the interface follow a random distribution given the propensities of the two species, while one naturally expects more H-H (P-P) contacts from the hydrophobic *core* (hydrophilic *rim*) of the interface. This simple observation calls for the development of geometric models quantifying multi-body interactions, so as to account for the complexity of atomic environments. As the neighbors of a given atom intersect its sphere along circles, one way to tackle this problem consists of investigating the equivalence classes of arrangements of circles induced by the neighbors of this atom. To do so, the first step consists of computing an arrangement of circles in a sphere so as to decompose the whole atomic surface. This decomposition features regions exposed on the molecular surface —which may be retrieved from α-shapes. But is also contains regions which are not exposed on the boundary, and encode higher order multi-body contacts.

1.2 The Arrangement of circles in a sphere

Arrangement and related problems. Consider a collection of n circles in a reference sphere S_0, and assume each circle corresponds to the intersection between S_0 and another sphere S_i. Recall the *arrangement* induced by the circles is the decomposition of the sphere into spherical polygons (faces), circle arcs (edges) and vertices. Denote B_i the ball associated to sphere S_i. Using these circles, one may pose four different problems. To define them, we

shall use the following terminology: a *singular point* is a point of S_0 crossed by at least two circles; a pair of circles *in contact* is a pair or circles running through a singular point. The problems are:
• P1. Report all singular points.
• P2. Report all pairs of circles in contact.
• P3. Construct the arrangement induced by the circles.
• P4. Assuming each circle is induced by a ball, solve P3, but also report the list of balls covering each face of the arrangement.

Problem P4 is of special interest in structural biology, as intersection circles are induced by neighboring atoms. Problem P3 solves problem P1, and implicitly contain the solution of problem P2. Yet, one may be interested in singular points and not in the arrangement, as the later requires handling topological data structures. However, for a collection of circles through the same singular point, problem P1 can be solved using linear storage while the number of intersecting pairs is quadratic. In conclusion, this paper develops an algorithm to solve problem P4. But by removing the appropriate pieces, one gets an algorithm solving the three other problems.

Previous work and contributions. Previous work on molecular surfaces and related geometric questions is discussed at length in section 2. The contribution closest to our is [HS98], which develops an algorithm to compute the trapezoidal decomposition of a sphere induced by a collection of circles. Degeneracies are removed using an explicit perturbation, which has the drawback of introducing a non-deterministic result.

We develop instead an algorithm computing the exact arrangement of circles in a sphere, based on an explicit handling of degeneracies and certified predicates. By working in the sphere —instead of using a stereographic projection, one avoids numerical troubles and infinite objects —as a circle through the projection pole is mapped to a line in the plane. The algorithm adapts the Bentley-Ottmann (BO) paradigm [dBvKOS97] to the spherical setting, the sweep line being replaced by a meridian M_θ. Apart from providing the exact solution, our strategy has two major advantages. First, assuming the circles are induced by balls, the algorithm can easily be adapted to report balls covering each face of the arrangement. Second, exactness guarantees the arrangements of intersecting atoms are coherent. For large collection of balls, arrangements can be computed in parallel and merged using a map overlay algorithm.

Our construction is motivated by the calculation of parameters describing inter-atomic contacts in structural biology, so as to go beyond the classical exposed and buried surface areas. As an illustration, statistics featuring spectacular changes wrt traditional observations on protein - protein complexes are provided.

1.3 Notations and paper overview

Notations. We consider a collection of n balls $B_{i,i=1,\ldots,n}$ intersecting a given ball B_0. Ball $B_i(a_i, r_i)$ is represented by its center $a_i = (x_i, y_i, z_i)$ and radius r_i. The sphere associated

to ball B_i is denoted S_i, and the intersection circle, if any, between S_0 and S_i is denoted C_i. The power of point p wrt sphere S_i is defined by $\pi(p, S_i) = pa_i^2 - r_i^2$. The radical flat $RF_{i,j}$ of any two spheres S_i, S_j is the plane containing the points whose power wrt S_i and S_j are equal. Whenever the spheres intersect, the intersection circle is also defined as the intersection between either sphere or the radical flat. Without loss of generality, we assume S_0 is centered at the origin. Sphere S_0 is endowed with cylindrical coordinates (θ, z), and by poles we refer to its North and South poles. Consider a meridian M_θ of S_0 parametrized by $\theta \in [0, 2\pi^-]$.

When developing algorithms, we shall use the following conventions : *variables* are written in italic, while `functions` are written in typewriter style. Functions prefixed `topo_` are dedicated to handling the topology of the arrangement.

Paper overview. This paper develops the combinatorial operations required by the spherical BO algorithm. All the predicates involved in the manipulation of intersection circles are presented in [CL], referred to as the *companion paper* in the sequel, where the following is proved:

Theorem. 1 *Predicates involved in the computation of the exact arrangement of intersection circles in a sphere rely on the comparison of algebraic numbers of degree at most two.*

The paper is organized as follows. Previous work on molecular surfaces is discussed in section 2. Fundamentals on BO algorithms are recalled in section 3, while terminology on circles is introduced in section 4. The algorithm is presented in sections 5 and 6, while its correctness is discussed in section 7. Questions concerned with the construction of the arrangement and the calculation of the multiplicity of its faces are discussed in sections 8 and 9. Finally, experiments are reported in section 10.

2 Molecular surfaces and related topics

2.1 Molecular surfaces

Starting with the pioneering work of Lee and Richards [Ric74, Ric77], later popularized by Connolly [Con83, Con85], molecular surfaces and volumes are ubiquitous in molecular modeling. These constructions are related to packing properties of atoms, solvation and electro-statics properties, identification of cavities, whence their importance in modeling docking and folding. To meet these needs, a variety of surface models have been proposed, the three prominent ones being the *Van der Waals / molecular - Connolly - Solvent Excluded / Solvent Accessible* surfaces, denoted as VdW/MS/SAS in the sequel. Not surprisingly, a wealth of algorithms computing them have been developed, a review of which up to 1996 may be found in [Con96]. These methods may be split into *exact* and *approximate*, the later using a discretization of the ambient space (a grid) or of the spheres to provide estimates. In the following, we describe selected exact methods with an emphasis on the underlying

geometry. Readers not familiar with the Delaunay/regular and Voronoi/power diagrams are referred to [BY98]. Programs accompanying the methods are cited in parenthesis.

To begin with, two general comments are in order. First, the key problem in computing any of the afore-mentioned surfaces is to compute the boundary of a union of balls: depending on the atomic radii used, this boundary is either the VdW or SAS surface, while the combinatorial structure of the MS matches that of the SAS [BDD92, TRS02]. Therefore, from now on, by *surface* we refer to the boundary of a union of balls. Second, observe that computing a surface subsumes at least two stages, namely finding the neighbors of an atom, and using these neighbors to compute the atom contribution to the surface. (Two atoms are neighbors if their balls intersect along a circle.) Spatial partitions can be used to efficiently report neighbors, the two standard ones being grids and Delaunay/regular triangulations. Interestingly, this later option offers the possibility to directly read the surface from a subset of the triangulation.

A first class of algorithms are based on properties of power diagrams. An elementary observation states that the contribution of any ball to the boundary of the union lies in its Voronoi cell —points outside this region are covered by other balls. A natural strategy therefore consists of computing the intersection between the the ball and its Voronoi region, or equivalently with the planes bounding this region. The later step requires a list of neighbors. These neighbors can be retrieved from a grid [VB93]. Following the non-naive way to construct a Voronoi region using polarity, selected redundant neighbors can be filtered out [FB98] (GETAREA) [1]. A more elaborate strategy consists of resorting to the α-complex with $\alpha = 0$ [Ede92]. This complex is a subset of the regular triangulation, and the structure of the boundary of the union directly reads from the 0-complex [AE96]. Alternatively, using space partitioning data structures, one can construct the contribution of the 0-complex to the boundary of the union by a greedy exploration of contacts between pairs and triples of balls [SOS96] (MSMS).

A second class of algorithms focuses on the intersection circles found in a given atom sphere, so as to construct the loops made of circle arcs bounding the convex spherical patches contributed by the atom to the surface. Example algorithms incrementally maintaining the relevant loops are [TA96], [TRS02] (SurfRace, FastSurf). On the other hand, one may compute the *arrangement* of intersection circles, that is the decomposition of the sphere into spherical patches, circle arcs, and intersection points. Given this arrangement (or its trapezoidal decomposition), the caps which are not covered by any neighbor contribute to the boundary of the union [HS98]. In passing, it should be noticed that to the best of our knowledge, this algorithm has been the first one to certify the resulting surface is free from degeneracy. This statement is achieved at the expense of explicit perturbations of the balls' centers.

To conclude with molecular surfaces, one may cite alternate representations based on density maps [Las95] (SURFNET), NURBS [BPS$^+$03], skin surfaces [CDES01].

[1] Instead of computing Voronoi as an upper envelope on the standard paraboloid, this paper uses the polarity wrt the unit sphere followed up by a convex hull calculation. Care is also taken to compute the quantities involved in the Gauss-Bonnet theorem.

2.2 Related topics

In three dimensions, the α-complex defines a simplicial complex of dimension three –as a k-simplex ($k \leq 3$) in the regular triangulation corresponds to the intersection between $k+1$ balls. Phrased differently, intersections featuring $n > 4$ balls cannot be inferred from the α-complex. On one hand, such intersections are useless if one focuses on standard molecular surfaces and volumes. On the other hand, such informations are relevant to investigate correlations involving $n > 4$ atoms. High order overlaps can be computed using inclusion-exclusion formulae [Ede95]. As shown in [Pet94], contacts up to order seven occur in small chemical compounds.

One may also cite related problems such as meshing molecular surfaces [LB01], maintaining surfaces dynamically [EH05], computing derivatives of areas and volumes [EK03].

3 Bentley-Ottmann algorithms

Line-segments in the plane. To report line-segments intersections in the plane [dBvKOS97], the BO algorithm requires two data structures which are an event point queue $\mathcal{E}$ featuring the line segments endpoints and intersection points, and another sorted data structure $\mathcal{V}$ providing the ordering along the vertical sweep line. The data structure $\mathcal{V}$ features segments intersected by the sweep-line, which are pairwise checked for intersection when they become consecutive in $\mathcal{V}$. The algorithm requires predicates to (i) test whether two segments intersect (ii) maintain $\mathcal{E}$ and (iii) maintain $\mathcal{V}$.

Circles in a sphere. For circles in a sphere, the extension of the BO algorithm consists of sweeping the sphere by a meridian M_θ. The tangency point(s) between M_θ and a circle, if any, induce a decomposition of the latter into one or two θ-monotonic circle arcs. This decomposition allows one to maintain a vertically sorted data structure $\mathcal{V}$ listing the θ-monotonic circle arcs intersected by M_θ during the sweep —such circle arcs are called *active*. Since each θ-monotonic circle arc is intersected in at most one point by M_θ, notice that $\mathcal{V}$ implicitly orders their intersection points. Therefore, at a given θ, *below, above* and *successor, predecessor* should be understood wrt this ordering along $\mathcal{V}$. In addition and to ease the search operations, $\mathcal{V}$ is also equipped with two sentinels set to the poles. The data structure used for $\mathcal{V}$ (and for $\mathcal{E}$) is a dictionary supporting the usual `contain, insert, delete` operations in logarithmic time. Practically, we use a red-black tree. The sweep process consists of having θ sweep the interval $[0, 2\pi^-]$. Similarly to the classical case, we consider $\mathcal{E}$ as an event queue. However, a major difference between line-segments and circles is that while endpoints are explicitly given in the former case, they are not in the latter. Developing predicates for that task is a crucial part of the algorithm.

4 Circles and event points

4.1 Circle classification

We shall use the following terminology, illustrated on Fig. 1:

Definition. 1 *A circle C_i is called* polar *if it goes through a single pole of S_0;* bipolar *if it goes through the two poles of S_0;* threaded *if the intersection point between $RF_{0,i}$ and the z-axis belongs to the open disk bounded by C_i;* normal *otherwise.*

With this definition, a normal circle defines two θ-monotonic circle arcs homeomorphic to the closed line-segment $[0, 1]$; a polar circle defines such an arc, the second one reducing to a point —the pole itself. During the sweep process, *active* arcs are those in $\mathcal{V}$. For homogeneity, we consider that a threaded circle defines an arc always active. From now on, these θ-monotonic circle arcs are just called arcs. Considering the two arcs of a normal or polar circle, we call them the *upper* and the *lower* arcs wrt the z axis. For clarity in the sequel, an arc shall never be reduced to a point (pole).
Note that defining arcs for a bipolar circle is useless since such a circle is not transversally intersected by M_θ, so that no insertion into $\mathcal{V}$ is planed —the circle contains the meridian at special θ values.

As a *great circle* in S_0 is a circle whose center is the center of S_0, observe that a non great circle is normal, polar or threaded, while a great circle is either threaded or bipolar.

Figure 1 The four types of intersection circles.

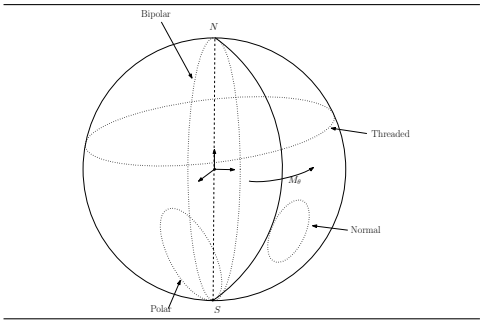
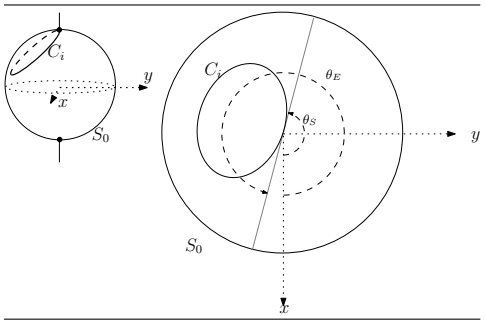


Figure 2 Start (θ_S) and end (θ_E) values of start and end points of a polar circle C_i.



4.2 Points vs events

We proceed with the geometric and the algorithmic representation of the objects manipulated, summarized on Fig. 3. As evoked above, a point refers to a point of S_0, expressed in cylindrical coordinates.

Normal critical points. Consider a normal circle C_i. The *normal critical points* associated to C_i are the two points where the meridian intersects C_i in a single point. The start point (θ_S, z_S) (end point (θ_E, z_E)) is the point where the intersection between the circle and the meridian starts (ends).

Polar and bipolar critical points. When a normal circle is shifted towards a pole, its critical points coalesce at the pole. Denoting z_p the z coordinate of the pole, we define the *polar critical points* as the pairs (θ_S, z_p) and (θ_E, z_p), with θ_S and θ_E the values such that the meridian is tangent to the circle. The start point is distinguished from the end point as for normal circles. See Fig. 2. This extension carries to bipolar circles, yielding *bipolar critical points*, the θ values being those where the circle is included in the plane containing the meridian —the z coordinate being irrelevant. For such a circle, the start (end) critical point is the one corresponding to the smallest (largest) value of θ.

Remark 1 *Critical points of a polar or a normal circle decompose its circle, say C_i into its lower (upper) arc denoted $\underline{A_i}$ ($\overline{A_i}$). When the upper or lower status does not matter, an arc is just denoted A_i.*

Crossing, tangency and intersection points. Whenever two circles intersect generically, there are two *crossing points*. If the two crossing points of two circles coincide, we speak of *tangency point*. A crossing or tangency point is also called an *intersection point*. A tangency point between two circles may coincide with their critical point —Fig. 3(d), and similarly, a crossing point may coincide with one critical point of the circles —Fig. 3(e). If so, for reasons explained after Def. 2, the status of critical dominates, and from now on, we qualify such points as degenerate tangency and degenerate crossing points.

The previous classification qualifies the local geometry at special points of S_0. But as *critical point* subsumes one circle while *intersection point* subsumes two circles, in order to distinguish the geometry (the *point*) from the arcs involved in algorithms, we define the notion of event:

Definition. 2 *The* critical event *associated to a normal (polar) critical point refers to the pair of arcs (the arc) defining the circle, together with a Start/End tag stating whether the point is a start or an end point. A bipolar (critical) event refers to the relevant circle arc bounded by poles, together with a Start/End tag. The event associated to a normal (polar,bipolar) critical point is called a normal (polar,bipolar) critical event. An* intersection event *associated to an intersection point refers to the pair of intersecting arcs.*

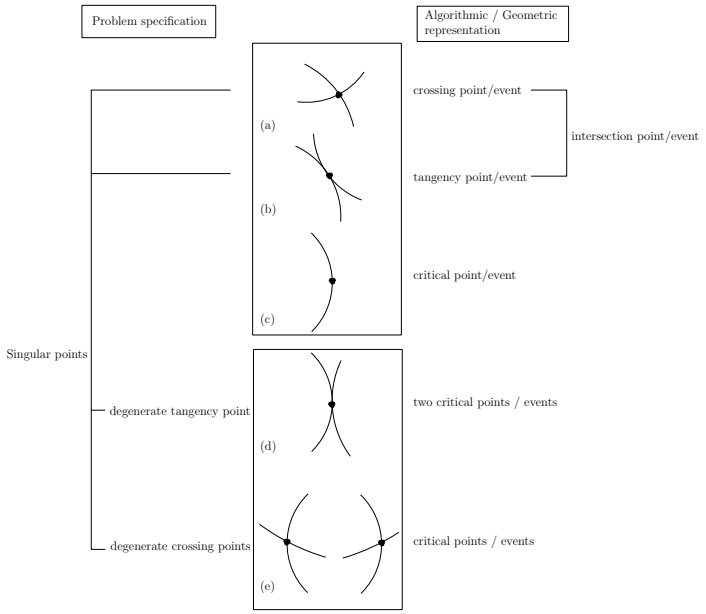
The point associated to an event is called the event point.

This said, we comment the afore-mentioned degenerate tangency and degenerate crossing points. In the classical BO algorithm, an intersection point acts like a *pivot* about which line-segments are permutated in the vertical ordering. In our setting, no permutation is required at a degenerate tangency and degenerate crossing point. In the same way, we do

not mention intersection points at the poles, because an event associated to such a point does not trigger any permutation of arcs. However, the intersections at these singular points are detected when processing start/end (bi)polar events. Conversely, as an intersection point does not coincide with a critical point of the circles involved, an intersection event is well specified in terms of arcs, and these arcs have to be permutated in the vertical ordering apart from the intersection point.

As a singular point is crossed by at least two different circles, such points allow one to specify the problems addressed.

Figure 3 Terminology away from poles. Singular points are used to specify the problem(s), while the algorithm uses intersection (crossing, tangency) and critical (start,end) points and events. As circles are decomposed into θ-monotonic arcs, intersection points are the only singular points triggering a permutation of arcs in the vertical ordering $\mathcal{V}$.



4.3 Filling the event queue $\mathcal{E}$ with event sites

To run the BO algorithm, we schedule *event sites* into $\mathcal{E}$. In this section, we introduce event sites and a total order on them. To present this order, which is guided by the local arrangement of circles in a neighborhood of the point of interest, we first define:

Definition. 3 *Two normal events are in conflict if their event points coincide. A (bi)polar critical event and an event are in conflict if their event point have the same θ value.*

▷ **1. Normal and (bi)polar event sites.** To handle conflicts between normal event points, we define the event site data structure, which uniquely associates a collection of normal events to a point of S_0. Because this association is one-to-one, we may also speak of the event site associated to a point. More precisely:

Definition. 4 *A* normal event site *is the partition of the normal events with same event point into the following three data structures:*
—One list $\mathcal{F}$ for end (finish!) events, sorted by increasing circle radius.
—One list $\mathcal{S}$ for start events, sorted by decreasing circle radius.
—One list $\mathcal{CT}$ which contains the crossing events and tangency events. The restriction of $\mathcal{CT}$ to crossing (tangency) events is denoted $\mathcal{C}(\mathcal{T})$.

Notice that elements of $\mathcal{F}$ ($\mathcal{S}$) correspond to the arcs of a circle to be removed from (inserted into) $\mathcal{V}$, while elements of $\mathcal{CT}$ refer to pairs of arcs intersecting. We shall get back on the sortedness of lists $\mathcal{F}$ and $\mathcal{S}$ in section 5. To handle circles through poles, we define similarly:

Definition. 5 *A* (bi)polar event site *associated to one (bi)-polar event is the event itself.*

Since all events stored in a normal event site have the same event point, we can say that an event site also defines an event point. Using the point of an event site, the notion of conflict can be extended among (bi)polar event sites, and between (bi)polar and normal event sites.

▷ **2. Ordering normal event sites.** Using the event points of two event sites, we define:

Definition. 6 *Consider two normal event sites whose event points are $p = (\theta_p, z_p)$ and $q = (\theta_q, z_q)$. The event site associated to p is said to* occur *before the one associated to q iff*

$$\theta_p < \theta_q, \quad or \quad \theta_p = \theta_q \quad and \quad z_p > z_q \tag{1}$$

▷ **3. Ordering polar event sites.** For conflicts between polar events, we need to distinguish between conflicts at the same pole —circles having the same tangent at the pole, and conflicts between circles through the two poles. Since (i) arcs associated to end points are removed from $\mathcal{V}$ before the insertion of arcs from start critical points (ii) $\mathcal{V}$ is maintained top-down (iii) the relative position of circles at a pole is a function of their radii, we get:

Definition. 7 *For two polar critical event sites in conflict, one has:*
—A polar end event site occurs before a polar start event site.
—A polar event site at the north pole occurs before one at the south pole.
—Among polar start (end) event sites at the same pole, the one whose associated circle is of largest (smallest) radius occurs first.

▷ **4. Ordering event sites: the general case.** Finally we must resolve conflicts between two different types of event site (normal, bipolar, polar) [2]. First, as a natural extension of the def. 7 and since a bipolar circle features the largest radius possible, a bipolar event site in conflict with polar event sites must be enclosed between those representing the start and the end events. Second, as a normal event point —whose θ value matches that of a bipolar event site— is located on the associated bipolar circle, the bipolar event site must be handled first. These two constraints do not impose an order between normal and polar start events. However, as normal event points are on the bipolar circle, we process them sequentially. Summarizing, we get the total order used to schedule event sites in $\mathcal{E}$:

Definition. 8 *Event sites of different types, if in conflict, are ordered as follows:*

$$e_{P_{ep}} < \left\{ \begin{array}{c} e_{B_{sp}} \\ e_{B_{ep}} \end{array} \right\} < e_N < e_{P_{sp}}$$

where $a < b$ *means* a occurs before b*;* e_{P_*}, e_{B_*}, e_N *for* polar, bipolar, normal event sites*;* $e_{*_{sp}}$, $e_{*_{ep}}$ *for* start, end.

4.4 Circles in a sphere : degenerate cases

Generically, two spheres intersect into a circle and three spheres intersect together into two points if they intersect at all. To enumerate all degenerate cases, we proceed as follows: starting from a generic configuration, we add the least number of spheres to end up with a non-generic one. In the following, the spheres we start with and those which are added are separated by a semi-colon.

▷ **1.** PC_1, $\{S_0; S_i\}$: sphere S_0 and S_i are tangent i.e. intersect in one point.
▷ **2.** PC_2, $\{S_0, S_i; S_j\}$: S_0 and S_i intersect along a circle, and S_j is another sphere in the pencil of these two spheres, i.e. intersects along the same circle.
▷ **3.** PC_3, $\{S_0, S_i; S_j\}$: S_i intersects S_0 along a circle. S_j intersects S_0 so that the commum intersection of S_i, S_j and S_0 is one point. Equivalently, C_i and C_j are tangent circles in S_0.
▷ **4.** PC_4, $\{S_0, S_i, S_j; S_k\}$: the two circles $C_i = S_0 \cap S_i$ and $C_j = S_0 \cap S_j$ intersect at two points, and C_k goes through one of these points. Equivalently, C_i, C_j, C_k intersect in a single point in S_0.
▷ **5.** PC_5, $\{S_0, S_i, S_j; S_k\}$: Circle C_k goes through the 2 intersection points of S_0, S_i, S_j.

Notice that case PC_3 is not inferred from PC_1 although S_i and S_j are tangent, since PC_1 is concerned by the tangency between S_0 and another sphere. Notice also relevant cases to consider in our BO adaptations are PC_3, PC_4 and PC_5. As we shall see later, cases PC_1 and PC_2 are concerned with the covering of faces of the arrangement by balls.

[2] A conflict between two bipolar event sites subsumes identical bipolar circles, which we prevent: see section 4.4.

Before proceeding, let us just mention how degenerate cases are handled. Case PC_1 is detected from the radii of the intersection circles, which is null in this case [3]. Case PC_2 is detected from the centers and radii of the intersection circles. Case PC_3, which features a single event, is handled in Algorithm 2. So are cases PC_4, PC_5, for which the ordering introduced in Def. 6 is essential in handling the several (at least two) events colliding at the event site.

5 Bentley-Ottmann in a sphere

5.1 Algorithm overview

The algorithm, whose pseudo-code is presented on Algorithm. 1, is similar to the standard BO algorithm in the plane [dBvKOS97]: we iterate over event sites and maintain the queue together with the ordering along the meridian. The ordering in $\mathcal{V}$ is initialized with arcs intersected by M_θ at $\theta = 0$. Since we aim at reporting the arrangement as a half-edge data structure (HDS), we also create the corresponding half-edges —to be completed at $\theta = 2\pi^-$. Queue $\mathcal{E}$ is initialized using critical events of all circles —but threaded ones which do not have any, together with the intersection events between arcs adjacent along $\mathcal{V}$ at $\theta = 0$.

Recall that functions prefixed by `topo_` are dedicated to the construction of the arrangement. The algorithm involves the following main functions:
—Function `classify_circles` classifies circles according to Def. 1.
—Function `topo_init_arrangement` initializes the HDS used to store the arrangement.
—Function `break_adjacencies` removes from $\mathcal{E}$ intersection events which do not correspond to two adjacent arcs in $\mathcal{V}$ upon processing of an event site.
—Function `handle_event_site` maintains $\mathcal{E}$ and $\mathcal{V}$, but also calls topological routines to manage half-edges at the event point : `topo_handle_left`, `topo_handle_right` and `topo_handle_above_below`.
—Function `handle_polar_bipolar_event_site` updates $\mathcal{E}$ and $\mathcal{V}$ for circles passing by a pole and has a topological part consisting in managing half-edges anchored at a pole.
—Function `topo_merge_virtual_faces` completes the construction of the arrangement once the event queue has been exhausted.

5.2 Blocks within a normal event site

To run the algorithm, we shall maintain the following:

Invariant. 1 *Upon processing of a normal event site, the following holds:*
—The list $\mathcal{CT}$ contains one tangency event for each pair of tangent arcs which are adjacent along $\mathcal{V}$.

[3] In constructing the arrangement, we shall actually skip this point, a design choice. We could equally report it as a point, located in a 2-face, 1-face or 0-face of the arrangement. As this point may be seen as the critical point of a circle of null radius, it can be located during the sweep process

Algorithm 1 Bentley-Ottmann: pseudo-code

```
1: classify_circles {find circle types }
2: Initialize event queue E
3: Initialize vertical order V
4: topo_init_arrangement
5: while (E ≠ ∅) do
6:    e=E.pop
7:    break_adjacencies(e)
8:    if (e is a normal event) then
9:       handle_event_site(e)
10:   else
11:      handle_polar_bipolar_event_site(e)
12:   end if
13: end while
14: topo_merge_virtual_faces
```

—The list $\mathcal{CT}$ contains one crossing event for each pair of arcs intersecting transversally which are adjacent along $\mathcal{V}$.

Notice this invariant subsumes an event site does not provide all pairwise intersecting arcs at a given point. To maintain it with the algorithm `break_adjacencies` presented in the next section, we shall need to infer the following structure on the lists of a normal event site :

Definition. 9 *A collection of normal events defines a* block *of arcs if the collection of the arcs associated to these events is a connected component of adjacent arcs in $\mathcal{V}$. The* top *and* bottom *arcs of a block are defined with respect to the arc positions in $\mathcal{V}$. The block is termed a* crossing (tangency) *block if all its events are crossing (tangency) events.*

Moreover, the upper and lower bounding *arcs of a block are the arcs of $\mathcal{V}$ located above and below the top and bottom arcs of the block.*

As illustrated on Fig. 4 (a), a set of events can define many blocks by considering its partition into subsets of events of same type. Notice the bounding arcs of a block always exist since the poles are used as sentinels in $\mathcal{V}$.

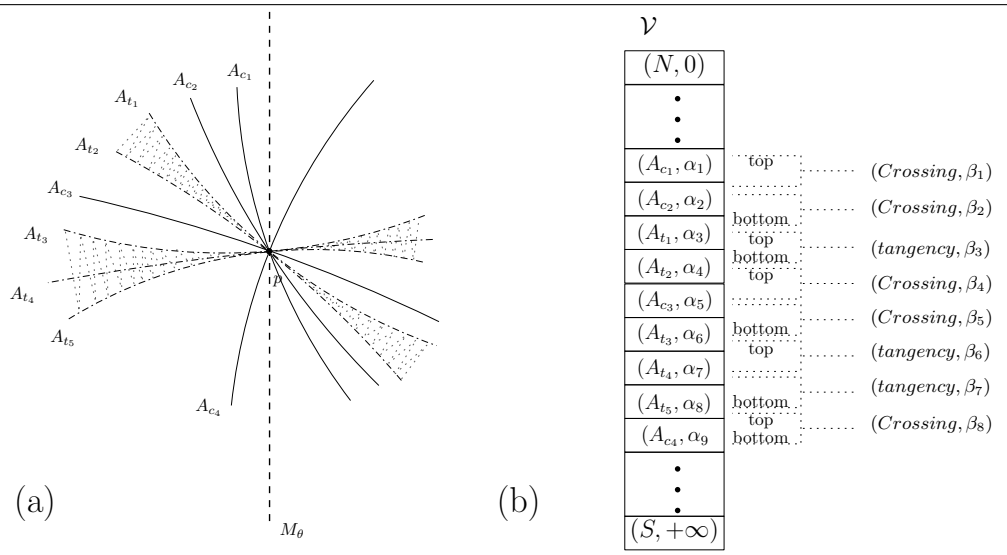
Given the list $\mathcal{CT}$ of a normal event site, an important operation, required by algorithm `break_adjacencies` to maintain invariant 1, but also algorithm `handle_event_site` to revert arcs along $\mathcal{V}$, consists of finding its tangency and crossing blocks. To present the algorithm `find_IT_block_bounds`, we shall assume for the sake of exposure that each arc in $\mathcal{V}$ is endowed with a key which is a rational number [4]. Upon insertion in $\mathcal{V}$, this key is set to the average of its neighboring arcs' keys. Moreover, any operation on $\mathcal{V}$ which

[4]The key can be skipped at the expense of predicates aiming at locally comparing Taylor expansions of the circles at the first or second order.

modifies the ordering of arcs alters the keys accordingly: the vertical ordering along $\mathcal{V}$ and that associated to the keys always match.

These keys serve two purposes. First, by scanning the arcs associated to events in the list $\mathcal{CT}$, one easily locates the top and bottom arcs of the $\mathcal{CT}$ block. Second, the keys are used to sort the intersection events of list $\mathcal{CT}$ —by sum of the arcs' keys. This sorted list of intersection events provides blocks by examining consecutive intersection event types (i.e. crossing or tangency). Notice this strategy is correct as long as Invariant 1 holds: before algorithm `find_IT_block_bounds` is launched, the list $\mathcal{CT}$ contains one event for each pair of intersecting arcs which are adjacent along $\mathcal{V}$. This process is illustrated on Fig. 4 (b), where α_i are the rational keys.

Figure 4 (a) In the event site corresponding to point p, there are: five crossing events : $(A_{c_1} \cap A_{c_2})$, $(A_{c_2} \cap A_{t_1})$, $(A_{t_2} \cap A_{c_3})$, $(A_{c_3} \cap A_{t_3})$ and $(A_{t_5} \cap A_{c_4})$; three tangency events : $(A_{t_1} \cap A_{t_2})$, $(A_{t_3} \cap A_{t_4})$ and $(A_{t_4} \cap A_{t_5})$. There are two tangency blocks defined by the tangency events : $[A_{t_1} A_{t_2}]$ and $[A_{t_3} A_{t_4} A_{t_5}]$, and three blocks defined by crossing events, namely $[A_{c_1} A_{c_2} A_{t_1}]$, $[A_{t_2} A_{c_3} A_{t_3}]$ and $[A_{t_5} A_{c_4}]$. $\mathcal{CT}$ defines the block $[A_{c_1} A_{c_2} A_{t_1} A_{t_2} A_{c_3} A_{t_3} A_{t_4} A_{t_5} A_{c_4}]$ (b) Sorting the intersection events yields the tangency and intersection blocks; $\alpha_1 < \alpha_2 < \cdots < \alpha_9$; $\beta_i = \alpha_i + \alpha_{i+1}$.



5.3 Algorithm `break_adjacencies`

Equipped with the notion of block, we present algorithm `break_adjacencies` to maintain invariant 1. We say that an intersection event is valid, if its associated arcs are in $\mathcal{V}$ and are adjacent along $\mathcal{V}$. We wish to keep in each normal event site of $\mathcal{E}$ intersection events corresponding only to arcs consecutive along $\mathcal{V}$. Assuming by induction that all intersection

events stored in normal event sites of $\mathcal{E}$ are valid before processing the top event site e, we need to remove from event sites of $\mathcal{E}$ events which are not valid anymore due to the changes in $\mathcal{V}$ triggered by e. The following two steps are executed.

▷ **1.** The first step, illustrated on Fig. 5, deals with the preservation of adjacencies between the top and bottom arcs of the $\mathcal{CT} \cup \mathcal{F}$ block, and its bounding arcs. This step features three exclusive cases:

(a). $\mathcal{F} \neq \emptyset$. Arcs associated to events of $\mathcal{F}$ are about to be removed from $\mathcal{V}$ by function `handle_event_site`. Note that if a circle C is intersected by M_0, then an intersection event involving arcs of this circle may have been detected but not been processed. Let C^+ and C^- the circle of greatest and smallest radius associated to end events of $\mathcal{F}$, and $\overline{A^+}/\underline{A^+}$ and $\overline{A^-}/\underline{A^-}$) associated upper/lower arcs.
— if $C^+ \cap M_0 \neq \emptyset$ remove from $\mathcal{E}$, if exists, any intersection event between $\overline{A^+}$ ($\underline{A^+}$) and its upper neighbor (lower neighbor) in $\mathcal{V}$
— if $C^- \cap M_0 \neq \emptyset$ remove from $\mathcal{E}$, if exists, any intersection event between $\overline{A^-}$ ($\underline{A^-}$) and its lower neighbor (upper neighbor) in $\mathcal{V}$

(b). $\mathcal{F} = \emptyset$ **and** $\big[(\mathcal{S} \neq \emptyset$ **and** $\mathcal{CT} \neq \emptyset)$ **or** $(\mathcal{S} = \emptyset$ **and** $\mathcal{C} \neq \emptyset)\big]$. As the top (bottom) arc of the $\mathcal{CT}$ block loses adjacency with the upper (lower) bounding arc, such intersection(s), if any, are removed from $\mathcal{E}$.

(c). $\mathcal{F} = \emptyset$ **and** $\mathcal{S} \neq \emptyset$ **and** $\mathcal{CT} = \emptyset$. Arcs associated to the inserted circle of largest radius C_s may break adjacencies as follows:

1. if there exists an arc passing through the start point of C_s, we remove from $\mathcal{E}$, if exists, any intersection event between this arc and the upper (and the lower) neighbor in $\mathcal{V}$ of $\overline{A_s}$ ($\underline{A_s}$).
2. if arcs of C_s are inserted between two arcs, we remove from $\mathcal{E}$, if exists, any intersection event between these two arcs.

▷ **2.** The second step takes care of adjacency loss between arcs reversed in the $\mathcal{CT}$ block, when $\mathcal{T} \neq \emptyset$ and $\mathcal{C} \neq \emptyset$. Indeed, as the bounding arcs of each tangency block of $\mathcal{CT}$ change upon processing the event, any intersection between such an arc and the top or bottom arc of the block must be removed from $\mathcal{E}$.

Notice that considering (bi-)polar event sites, we just have to consider end event whose point is associated to a polar circle intersected by M_0 and remove from $\mathcal{E}$, if exists, any intersection event between the arc associated to the polar event, and its nearest neighbor in $\mathcal{V}$ which is not a pole (if exists).

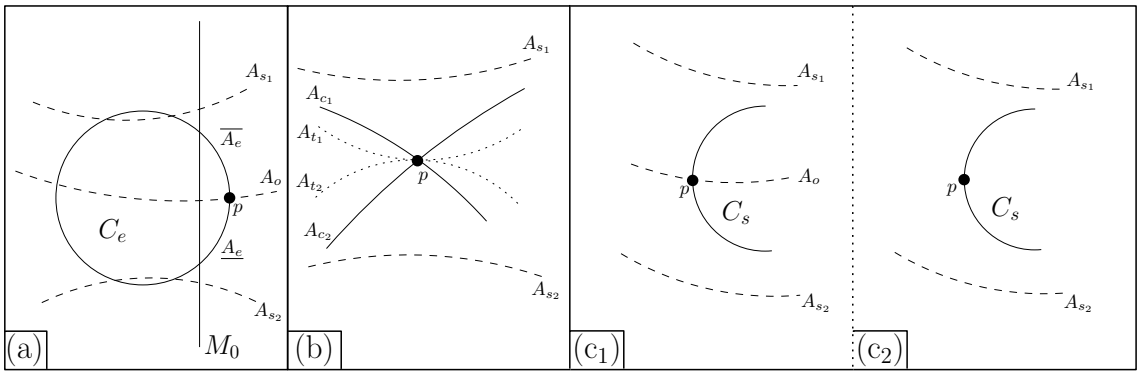
The completeness of algorithm `break_adjacencies` comes from the fact that all cases $\{\mathcal{F} \neq \emptyset, \mathcal{F} = \emptyset\} \times \{\mathcal{S} = \emptyset, \mathcal{S} \neq \emptyset\} \times \{\mathcal{CT} = \emptyset, \mathcal{CT} \neq \emptyset\}$ are covered (case $\mathcal{CT} \cup \mathcal{F} \cup \mathcal{S} = \emptyset$ not relevant). Whence:

Observation. 1 *Algorithm* `break_adjacencies` *maintains invariant 1.*

Remark 2 *Because of algorithm* `break_adjacencies`*, the event site data structure of normal event sites —see section 4.3, is a dynamic container which is created upon the discovery of the first associated event, and which is processed when that event site is popped from the queue* $\mathcal{E}$*.*

Remark 3 *Incidentally, algorithm* `break_adjacency` *guarantees the size of* $\mathcal{E}$ *remain linear, as for the classical BO algorithm [BY98].*

Figure 5 Breaking adjacencies between arcs. Points associated to events are represented by black bullet. (a) When circle C_e intersected by M_0 ends, we remove intersection events of $(A_{s_1}, \overline{A_e})$, $(\underline{A_e}, A_{s_2})$ and $(\overline{A_e}, A_0)$; (b) Processing event site corresponding to p breaks adjacency between pairs (A_{s_1}, A_{c_1}), (A_{c_1}, A_{t_1}), (A_{t_2}, A_{c_2}) and (A_{c_2}, A_{s_2})



(c_1) Inserting circle C_s break adjacency between pairs (A_{s_1}, A_o) and (A_o, A_{s_2}); (c_2) Inserting circle C_s break adjacency between pair (A_{s_1}, A_{s_2}).

5.4 Handling event sites

Finally, we present algorithms `handle_event_site` and `handle_polar_bipolar_event_site`, to handle normal and, polar and bipolar, event sites. These algorithms consist of inserting and removing arcs into/from $\mathcal{V}$, reverting arcs in $\mathcal{V}$, and inserting into $\mathcal{E}$ the new intersections revealed by new adjacencies in $\mathcal{V}$. The pseudo-code of Algorithm 2 uses the following conventions: the node of $\mathcal{V}$ holding arc A_i is denoted $v[A_i]$; reciprocally, the arc associated to a node say x of $\mathcal{V}$ is denoted $*x$; the function `Above` (resp. `Below`) returns the node holding arc above (resp. below) a in $\mathcal{V}$ (i.e. previous and next node).

Algorithm `handle_event_site`**.** This algorithm consists of processing the three lists of events associated to a normal event site. To exploit the relative position of circles at the associated event point, two variables *arc_sup* and *arc_inf* are used to record the nodes of $\mathcal{V}$ holding the arcs bounding the several blocks encountered. More precisely:

▷ **1.** First, arcs ending (list $\mathcal{F}$) are removed, a process from which the variables *arc_sup* and *arc_inf* are initialized;

▷ **2.** Second, arcs starting (list $\mathcal{S}$) are iteratively inserted in-between *arc_sup* and *arc_inf*, which are (initialized if necessary, and) maintained along the process;
▷ **3.** Third, arcs corresponding to intersections and tangencies (list $\mathcal{CT}$) are reversed.

The operations just listed are accompanied by the appropriate intersection tests, so as to update $\mathcal{E}$:
▷ **1.** If we have only circles ending, the two arcs bounding those of circles ending become adjacent. Note that if there exists only one arc passing through the associated end points, it is this arc that is adjacent to the two arcs bounding in $\mathcal{V}$ the corresponding arcs of circles ending.
▷ **2.** If we have circles starting, the upper (lower) arc of the circle of largest radius starting becomes adjacent with the arc above (below) it in $\mathcal{V}$. Idem with the circle of smallest radius for its upper (lower) arcs with the arc below it (above) (handled degenerate crossing point).
▷ **3.** If we have crossing events and tangency events, two steps:

- The top and bottom arcs of the block of arcs defined by $\mathcal{CT}$ after reversal get adjacent to the bounding arcs of the block defined by $\mathcal{CT}$.
- Similarly, the top and bottom arcs of each $\mathcal{CT}$ tangency block, after reversal, get adjacent to the bounding arcs of that block.

Note that we avoid testing a second time pairs of arcs previously handled by just reminding those already tested.

Algorithm `handle_polar_bipolar_event_site.` Consider a polar event site. To handle a such start (end) event, we have to insert in (remove from) $\mathcal{V}$ the associated arc close to the correct pole.
Consequently, the intersection test is performed after insertion of arc close to the corresponding pole, between this arc and its only relevant neighbor in $\mathcal{V}$.

Bipolar event site do not induce any insertion of arcs and so no intersection tests. But even if the action of these events are topological, we discover all singular points induced by such a circle by using active arcs of $\mathcal{V}$ when a bipolar event site is encountered during the sweep process. See section 8.2 for more detail.

Remark 4 (Implementation) *To optimize memory requirements while initializing $\mathcal{E}$, we insert start events only, a given end event being inserted when M_θ encounters its start event or if the associated circle is intersected by M_0.*

Algorithm 2 handle_event_site

1: arc_sup = **NULL**
2: arc_inf = **NULL**
3: `topo_handle_left`
4: {Remove arcs ending, if any, by increasing radius}
5: **if** ($\mathcal{F} \neq \emptyset$) **then**
6: **for** each circle C_{e_i} of the end point associated to event in $\mathcal{F}$ **do**
7: Remove from $\mathcal{V}$ the arcs $\underline{A_{e_i}}$ and $\overline{A_{e_i}}$
8: **end for**
9: Record into arc_sup and arc_inf, the nodes of the two arcs below and above the ending circle of largest radius, if exists
10: **if** ($\mathcal{S} = \emptyset$ and $\mathcal{CT} = \emptyset$) **then**
11: **if** ($arc\_sup \neq$ `Above`(arc_inf)) **then**
12: Test intersections of $*arc\_sup$ and $*$`Below`(arc_sup), and possibly update $\mathcal{E}$
13: **end if**
14: Test intersections of $*arc\_inf$ and $*$`Above`(arc_inf), and possibly update $\mathcal{E}$
15: **end if**
16: **end if**
17: {Insert arcs starting, if any}
18: **if** ($\mathcal{S} \neq \emptyset$) **then**
19: **if** (arc_sup = **NULL**) **then**
20: Insert arcs $\underline{A_{s_0}}$ and $\overline{A_{s_0}}$ into $\mathcal{V}$
21: **else**
22: Insert arc $\overline{A_{s_0}}$ below arc_sup
23: Insert arc $\underline{A_{s_0}}$ above arc_inf
24: **end if**
25: $arc\_sup \leftarrow v[\overline{A_{s_0}}]$
26: $arc\_inf \leftarrow v[\underline{A_{s_0}}]$
27: Test intersections of $*arc\_sup$ and $*$`Above`(arc_sup), and possibly update $\mathcal{E}$
28: Test intersections of $*arc\_inf$ and $*$`Below`(arc_inf), and possibly update $\mathcal{E}$
29: **for** each start event in $\mathcal{S}$ whose point is associated to circle C_{s_i} **do**
30: Insert arc $\overline{A_{s_i}}$ below arc_sup; $arc\_sup \leftarrow v[\overline{A_{s_i}}]$
31: Insert arc $\underline{A_{s_i}}$ above arc_inf; $arc\_inf \leftarrow v[\underline{A_{s_i}}]$
32: **end for**
33: **if** ($\mathcal{CT} = \emptyset$) **then**
34: `topo_handle_right`
35: **if** ($arc\_sup \neq$ `Above`(arc_inf)) **then**
36: Test intersections of $*arc\_sup$ and $*$`Below`(arc_sup), and possibly update $\mathcal{E}$
37: Test intersections of $*arc\_inf$ and $*$`Above`(arc_inf), and possibly update $\mathcal{E}$
38: **end if**
39: **end if**
40: **end if**
41: {Process tangent arcs or arcs intersecting}
42: **if** ($\mathcal{CT} \neq \emptyset$) **then**
43: `find_IT_block_bounds`
44: { if arc_sup and arc_inf are **NULL**, they are updated to respectively top and bottom of the block defined by $\mathcal{CT}$ }
45: Reverse block of all arcs defined by $\mathcal{CT}$
46: **for** (each block **B** defined by tangency events of $\mathcal{CT}$) **do**
47: Reverse block **B**
48: **end for**
49: `topo_handle_right`
50: Test intersections between new adjacent arcs and possibly update $\mathcal{E}$
51: **end if**
52: `topo_handle_above_below`

6 Handling $\mathcal{V}$

This section describes the three operations underwent by the vertical ordering $\mathcal{V}$: its initialization; the insertion/deletion of arcs when start/end events are encountered; the revert operations induced by intersection events.

6.1 Initializing $\mathcal{V}$

To initialize $\mathcal{V}$, we first collect among relevant circles (normal, threaded, polar) those whose intersection with M_0 does not reduce to a pole. See the companion paper for the predicates involved during this step. Then, since M_θ intersects an arc in only one point, the initialization consists of sorting the corresponding intersection points by decreasing z value, so that difficulties are faced in case of z-ties. Given such a singular point, we sort the arcs to its left [5], that is we find the vertical ordering of the arcs at $\theta = 2\pi^-$. Running this sort requires a pairwise ordering of arcs, which is based upon a first order calculation if the tangents to the circles at the singular point are distinct, or a second order calculation in case of tangent circles. See details in the companion paper.

All pairs of adjacent arcs in $\mathcal{V}$ are tested for intersection, and the corresponding events inserted into $\mathcal{E}$. In particular, the events occurring at $\theta = 0$ are scheduled. We also schedule in $\mathcal{E}$ end events of circles intersected by M_0.

6.2 Inserting starting arcs into $\mathcal{V}$

First observe that only normal circles trigger non trivial insertions of arcs into $\mathcal{V}$: the arc of a polar circle comes right after its pole; bipolar circles are processed on the fly at the event sites without any manipulation of $\mathcal{V}$.

As explained in algorithm 2, the key step in inserting arcs of normal circles associated to start events of the list $\mathcal{S}$ consists of locating the start point p of the circle of largest radius C_s. (If $\mathcal{F} \neq \emptyset$, this operation is superfluous.) To locate p and since $\mathcal{V}$ is vertically sorted, we wish to run a binary search, which requires stating whether p is located below / above / on a given arc of $\mathcal{V}$.

To present the comparator required by this binary search, we define the following orientation for the radical flats (RF) of all circles drawn in S_0. Denoting c_i the center of circle C_i, we orient the normal vector $\mathbf{n}$ of a RF as follows: for a normal or polar circle, the orientation of $\mathbf{n}$ is that of $\mathbf{a_0c_i}$; for a threaded circle, the orientation is such that $\mathbf{n}$ makes an acute angle with z-axis. Using this orientation, a point is said to be above (below) the RF if it is in the half-space of (opposite to) the normal $\mathbf{n}$. As illustrated on Figs. 6 and 7 and detailed in the companion paper:

Observation. 2 *The position of a start point p wrt to an arc whose circle is C_i can be computed as follows:*

[5] When reporting the arrangement in a half-edge data structure, the ordering of arcs at $\theta = 2\pi^-$ is also used to merge half-edges at the end of the sweep process. Sees section 8.

–If C_i is a normal circle, assume we wish to compare the z coordinate of p with the intersection point $q(\underline{A_i})$ ($q(\overline{A_i})$) between M_θ and an active arc $\underline{A_i}$ ($\overline{A_i}$) in $\mathcal{V}$. One has: if point p is above the radical flat $RF_{0,i}$, then p is below $q(\overline{A_i})$ and above $q(\underline{A_i})$; else, the position of p is given by the sign of the difference of z coordinate of p and c_i.
–If C_i is a threaded or polar circle, the position of p wrt this arc is characterized by its position wrt the oriented radical flat $RF_{0,i}$.

Using this observation, the arcs of circle C_s are inserted as follows. If p is not on any arc, the upper and lower arcs are inserted into $\mathcal{V}$ where indicated by the binary search. Otherwise, if p lies on arc say A_i, it is easily seen that its circle and C_i are transverse. (By assumption $\mathcal{F} = \emptyset$ so that no arc is ending at p; moreover C_s is assumed to be the normal circle of largest radius starting at p). Therefore, the upper and lower arcs are respectively inserted above and below the collection of arcs point p lies on.

Figure 6 Position of start point p w.r.t. normal circles: the position of p wrt circle arcs reads from the position wrt oriented radical flats. See text.

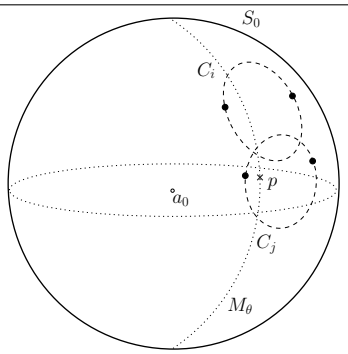
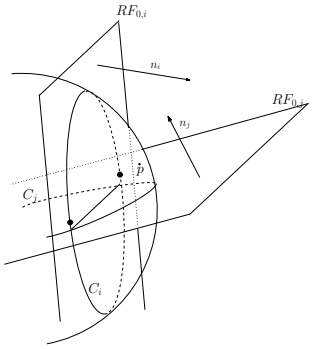


Figure 7 The position of start point p wrt a threaded circle also read from oriented radical flat. See text.



6.3 Intersection events and arcs reversing

In all generality, an event site features crossing and tangency events. The maintenance of $\mathcal{V}$ once an event site has been passed requires exchanging the order of the arcs in blocks defined by crossing events of the event site, wrt blocks defined by tangency events. (Together with the accompanying rational keys if these are used: see section 5.2.) We do so in two steps, by first reverting the arcs involved in the block defined by $\mathcal{CT}$, and second by reverting again the arcs involved in tangency blocks. Since we use a tree for $\mathcal{V}$, reverting a set of consecutive arcs can be done by re-writing the values of the nodes holding the arcs involved —which does requires any insertion of deletion on the tree.

7 Spherical BO algorithm: correctness and complexity analysis

As the algorithm uses critical points, we shall need considerations on the structure of the arrangement. A vertex of this arrangement is either a singular point, or a non-degenerate critical point. Denoting k (v) the number of singular points (vertices), we have $v = O(n+k)$. An edge is a circle arc in-between two vertices. A face is a region of S_0 whose interior is connected. Some difficulties arise if the 1-skeleton of the arrangement is not connected, in which case faces contain holes.

Definition. 10 *Consider a simply connected region R consisting of the union of one or several faces. The* support *of R is the topological hemi-sphere containing R. If the 1-skeleton of R is not connected, the* principal 1-skeleton *is the connected component (cc) connected to the boundary of R*

Consider a non-simply connected face bounded by several cycles: the principal 1-skeleton defines the principal cycle*, while the support of each remaining cycle defines a* hole *of the face.*

Notice a hole is a simply connected region which consorts of the union of a collection of faces. Notice also this Def. is recursive, as a hole is a simply connected region which is the union of faces, each of them possibly containing holes. The arrangement may be seen as tree specified as follows: one leaf corresponds to a simply connected region consisting of one (or several) simply connected face(s); one internal node corresponds to the support of a non simply connected face. See Fig. 8.

Lemma. 1 *Denote v and e the number of vertices and edges of an arrangement of n circles in the sphere, and let l stand for the number of faces bounded by exactly two edges. One has $e \leq 3(v-1)+l$.*

Proof. We apply Euler's relationship to simply connected regions.

Starting from the root of the afore-mentioned tree, consider the decomposition of S_0 into simply connected regions —faces or their supports. Let V_s, E_s the numbers of vertices and edges. Moreover, let $F_s = F_s^{(3)} + L_s$ be the number of faces/supports, with $F_s^{(3)}$ (L_s) the number of faces bounded by at least (exactly two) three edges. As an edge bounds two faces/supports, we have $2E_s \geq 3F_s^{(3)} + 2L_s$. But from the Euler characteristic of the sphere, we get $V_s - E_s + F_s = V_s - E_s + F_s^{(3)} + L_s = 2$. Whence

$$E_s \leq 3(V_s - 2) + L_s. \tag{2}$$

We now apply Euler's relationship to the decomposition of each hole by its principal 1-skeleton. To do so, some care is in order as the unbounded face around the hole has been accounted for —the support of its containing face. For a given hole whose number of vertices, edges and faces are also denoted V_i, E_i, F_i, let T_i be the number of edges bounding

the unbounded face. Counting edges over faces of the hole yields $2E_i - T_i \geq 3F_i^{(3)} + L_i$. As, $V_i - E_i + F_i^{(3)} + L_i = 1$, we get

$$E_i \leq 3(V_i - 1) - T_i + L_i < 3(V_i - 1) + L_i. \tag{3}$$

The 1-skeletons involved in the proof of Equations (2) and (3) are independent and form a partition all vertices and edges of the arrangement, that is, summing over all connected components, we get $v = v_s + \sum_i V_i$, $l = L_s + \sum_i L_i$. Therefore, summing the above two inequalities completes the proof. □

We shall also need the following lemma:

Lemma. 2 *Invariant 1 is always verified before handling an event site.*

Proof. From section 6.1, we know that after the initialization of $\mathcal{V}$, we look for intersection events between adjacent arcs in $\mathcal{V}$. This guarantees that invariant 1 is verified before handling the first event site. Moreover, in observation 1, we saw that `break_adjacency` maintains this invariant before handling a new event site. is true. □

Finally, we now state the main theorem. Notice its complexity involves the so-called *lenses* and *lunes* determined by a family of circles. For n circles of arbitrary radii in the plane, this number is known to be is $O(n^{\frac{3}{2}+e})$, for any $e > 0$, where the constant of proportionality depends on e [ALPS01].

Theorem. 2 *Algorithm 1 correctly reports all the singular points of a family of n circles in a sphere. Denoting k the number of such points, the algorithm uses $O(n)$ storage and has $O((n + k + l) \log n)$ complexity.*

Proof. **Correctness.** Following the terminology of Fig. 3, we establish that Algorithm 1 reports intersection points, all degenerate tangency points, and all degenerate crossing points.

For the first class, the correctness is similar to that of the classical BO algorithm. More precisely, any intersection points is reported, since we detect an intersection events between two arcs when they get adjacent in $\mathcal{V}$. From lemma 2, such an intersection point is reported a number of times at least equal to the number of consecutive pairs of arcs passing through the same intersection point before the corresponding event site is processed. (Notice *at least* comes from the updates perform by algorithm `break_adjacencies`.) Second, Degenerate tangencies are detected upon insertion of start events and end events into $\mathcal{E}$, as the corresponding event points are identical. Third, for degenerate crossings, we distinguish degeneracies associated to a start and an end critical point. For the former, the crossing is detected when the two circles arcs are inserted into $\mathcal{V}$, as the start point is found to be on a circle arc. for the later, for an event site with $\mathcal{F} \neq \emptyset$ and $\mathcal{CT} = \emptyset$, when removing the two arcs associated to the circle of smallest radius, the fact these arcs are not adjacent in $\mathcal{V}$ witnesses the degenerate crossing.

Finally, note that singular points occurring at $\theta = 0$ are correctly detected as the initialization of $\mathcal{V}$ at $\theta = 2\pi^-$ is followed by an intersection test of adjacent arcs.

Memory requirements. The vertical ordering requires linear storage. The event queue also has linear size since Algorithm `break_adjacencies` makes sure only events corresponding to arcs incident along $\mathcal{V}$ are stored.

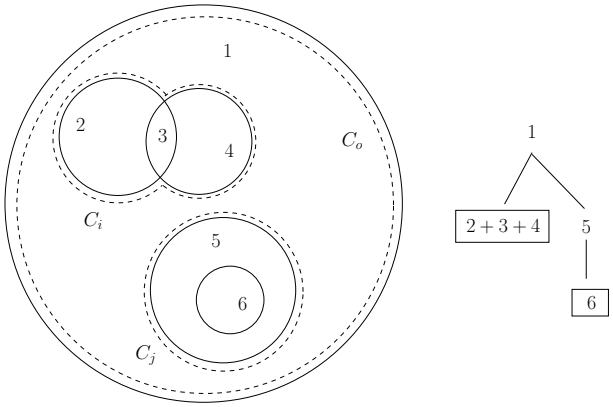
Time complexity. To analyse the complexity, let us consider the initialization, and the overall cost of the **while** loop over the queue $\mathcal{E}$. Before doing so, a comment is in order with respect to the insertion of an event into $\mathcal{E}$. Such insertion indeed requires looking for an event site in $\mathcal{E}$ —and creating one if necessary. But for normal event, in case of conflicts, critical points associated must be inserted into the sorted list $\mathcal{S}$ or $\mathcal{F}$. As $\mathcal{E}, \mathcal{S}, \mathcal{F}$ are dictionaries of size $O(n)$, the insertion/removal costs $O(\log n)$.

To begin with, the algorithm classifies the circles, which incurs a linear cost. As there are $O(n)$ normal circles, inserting these critical events into $\mathcal{E}$ requires $O(n \log n)$. Similarly, inserting at most $2n$ arcs into $\mathcal{V}$ costs $O(n \log n)$.

Consider now the **while** loop over queue $\mathcal{E}$. For each event processed, denote m_p the number of arcs passing through the corresponding point. The following steps need to be accounted for:
— removing from $\mathcal{E}$ the intersection points corresponding to arcs no longer adjacent in $\mathcal{V}$: $O(m_p \log n)$;
— removing/inserting ending/starting arcs from/into $\mathcal{V}$: $O(m_p \log n)$;
— finding bounding arcs of the block $\mathcal{CT}$: $O(m_p \log m_p)$; — detecting and inserting into $\mathcal{E}$ new intersection events corresponding to consecutive arcs along $\mathcal{V}$: $O(m_p \log n)$;
— exchanging the position in $\mathcal{V}$ of arcs intersecting at the event point: $O(m_p)$.

The cost of one iteration is therefore $O(m_p \log n)$, so that the overall costs of iterations is $O(M \log n)$, with $M = \sum_{p \text{ an event site}} m_p$. But $M = 2e$ with e the number of edges. From lemma 1, we have $M = O(v + l)$, and since $v = O(n + k)$, we get $M = (n + k + l)$. Adding up the cost of the initialization and of the iterations yields the overall complexity. □

Figure 8 Arrangement of 6 faces. Cycles bounding face 1 are C_o, C_i, C_j. Cycle C_i bounds a hole with 3 faces, while C_j bounds face 5 which contains a hole.



8 Reporting the arrangement in a half-edge data structure

In this section, we develop the topological operations required to construct the arrangement induced by the circles and store it in a half-edge data structure (HDS).

8.1 Describing the arrangement

Arcs and half-edges. The vertices of the HDS correspond to event points, its edges are circle arcs delimited by such vertices, while the faces are the regions of S_0 bounded by vertices and edges. Recall a face is a two-dimensional region whose interior is connected, and that half-edges are oriented so as to leave the interior of the face to its left. Over the course of the BO algorithm, an half-edge is created when its first vertex is created, and remains active until its second vertex is created. Following classical terminology, notice the first vertex may be the source or the target vertex of the half-edge [6]. To each arc A_i, we associate two active half-edges qualified wrt the intersection between the meridian and the arc —as this intersection is unique: the *upper* (*lower*) half-edge associated to A_i is the half-edge leaving to its left the portion of S_0 lying above (below) A_i. For a threaded circle which is not intersected by any other circle, at the end of the sweep process, the source and the target of the half-edges associated to its arc are fixed to a common null vertex [7]. For bipolar circles, no arc is defined. Half-edges are created on the fly at the bipolar events, so as to handle intersections with active arcs –if any.

As two points on a circle, a source and a target, define two half-edges, we need to qualify both. For a circle which is not a great circle, consider the spherical cap of S_0 of smallest area induced by this circle. If an half-edge on this circle induces this cap, it is called *inner*, and *outer* otherwise. For a great circle, we adopt the following conventions: if the circle is bipolar, an half-edge is *inner* if it induces the spherical cap swept by M_θ between its θ_S and θ_E associated values —*outer* otherwise; if the circle is threaded, an half-edge is *inner* if it induces the spherical cap containing the north pole —*outer* otherwise.

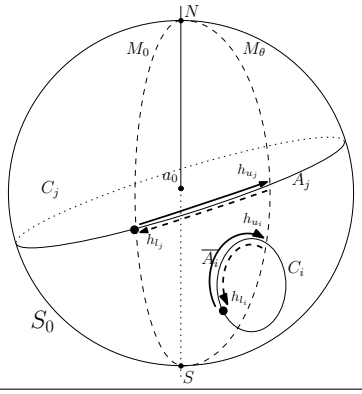
Faces and holes. To describe the arrangement, we use sequences of connected half-edges (SCH), such a sequence being called *closed* if its topology is that of a circle. The atomic operation in incrementally building a SCH consists of merging two half-edges, which amounts to fixing the common vertex and updating the next/previous pointers. Two active half-edges associated to arcs in $\mathcal{V}$, with respect to the ordering along $\mathcal{V}$, are termed *adjacent* if their arcs are adjacent in $\mathcal{V}$, and if they bound the same segment along the meridian M_θ. (Out of the four pairs of half-edges associated to two consecutive arcs along $\mathcal{V}$, a single pair corresponds to adjacent half-edges.). Following Def. 10, a *face* of the arrangement is represented by a collection of closed SCH. Each such sequence is called a *Connected Component of the Boundary* or CCB. A CCB is oriented and always induces a contractible region in the

[6] When a pair of opposite half-edges associated is no longer active, it is stored in the HDS.
[7] Any half-edge with two null vertices is bound to a threaded circle void of intersection.

sphere S_0. The set of all CCB describing a face can be split into two sets: one CCB called *principal*, which defines the spherical cap containing the face; the remaining ones defining *holes* in the face. See Fig. 16 [8].

Remark 5 *The structure of principal CCB and holes is recursive: see Def. 10.*

Figure 9 Half-edges formation : looking for a second vertex. Circle C_j is a great circle and C_i is a normal one. Solid half-edges are *inner*, dashed ones are *outer* for the arcs A_j and $\overline{A_i}$. A large black dot represents a point fixed as vertex for a pair of half-edges



8.2 Handling half-edges

Initialization and termination. As explained in section 6.1, the initialization of $\mathcal{V}$ consists of finding the ordering of arcs along M_θ for $\theta = 2\pi^-$. To be able to define the faces cut by M_0, to each arc in $\mathcal{V}$ after initialization, we attribute a pair of opposite half-edges with one *null* vertex implicitly representing the intersection between M_θ — with $\theta = 2\pi^-$, and the corresponding arc. This collection of half-edges is stored in a list H_0. At the end of the sweep process, we have a one-to-one correspondence between half-edges of arcs in $\mathcal{V}$ and the sequence of half-edges in H_0, so that a merge can be performed. These tasks correspond to functions `topo_init_arrangement` and `topo_merge_virtual_faces` in algorithm 1.

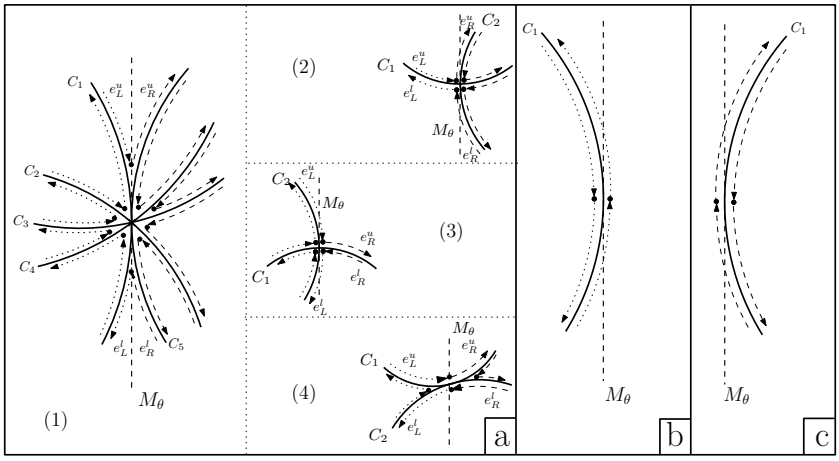
Handling half-edges for a normal event. To describe the operations underwent by half-edges at a normal event site, consider the arcs involved in the three lists of a normal event site in the neighborhood of the corresponding event point, say p. First, as arcs associated to events in $\mathcal{F}$ are removed from $\mathcal{V}$, the corresponding half-edges are stopped i.e. their second vertex is fixed at p. Second, as arcs associated to event from the list $\mathcal{CT}$ go through p, their active half-edges are stopped at p, and new half-edges are created with p

[8] A face is created and stored in the HDS at the end of the sweep process when its holes and its principal CCB have been defined

as first vertex. Third, while inserting into $\mathcal{V}$ new arcs associated to events in the list $\mathcal{S}$, new half-edges are created with p as first vertex. These operations are performed as follows.

To reflect the position of half-edges associated to arcs passing through p wrt the meridian, a half-edge stopped is said to be to the left of p, while one created is said to be to the right of p. Using the bounding arcs of the block defined by the union of the three lists of the event site —see algorithm 2, we proceed as follows:
▷ **1.** Functions `topo_handle_left` and `topo_handle_right`. We make two series of merges between adjacent half-edges, to the left and to the right of p. Each such merge features half-edges of arcs from $\mathcal{V}$, if any, enclosed in-between the bounding arcs.
▷ **2.** Function `topo_handle_above_below`. To the left of p, let e_L^u, e_L^l be the half-edges respectively adjacent to the lower half-edge of the upper bounding arc, and to the upper half-edge of the lower bounding arc. To the right of p, define (e_R^u,e_R^l) similarly. We face three cases: (i) if theses two pairs are defined , we merge e_L^u with e_R^u and e_L^l with e_R^l. See Fig. 10(a). (ii) if e_R^l and e_R^u are not defined, we merge e_L^l with e_L^u. See Fig. 10(b). (iii) if e_L^l and e_L^u are not defined, we merge e_R^l with e_R^u. See Fig. 10(c).

Figure 10 Operations on half-edges at a normal event. Dashed half-edges are new active ones; dotted ones represent half-edges associated to active arcs, before handling the event site. A black dot corresponds to a merge of two half-edges. (a) different cases with merge to the left and to the right from M_θ. Sub-case (1) is a general example while sub-case (2) and (3) degenerate crossing point. Sub-case (4) shows that a vertex is added when only $\mathcal{T}$ list is not empty, in order to have the opposite half-edge relation ; (b) merge at end event; (c) merge at start event;



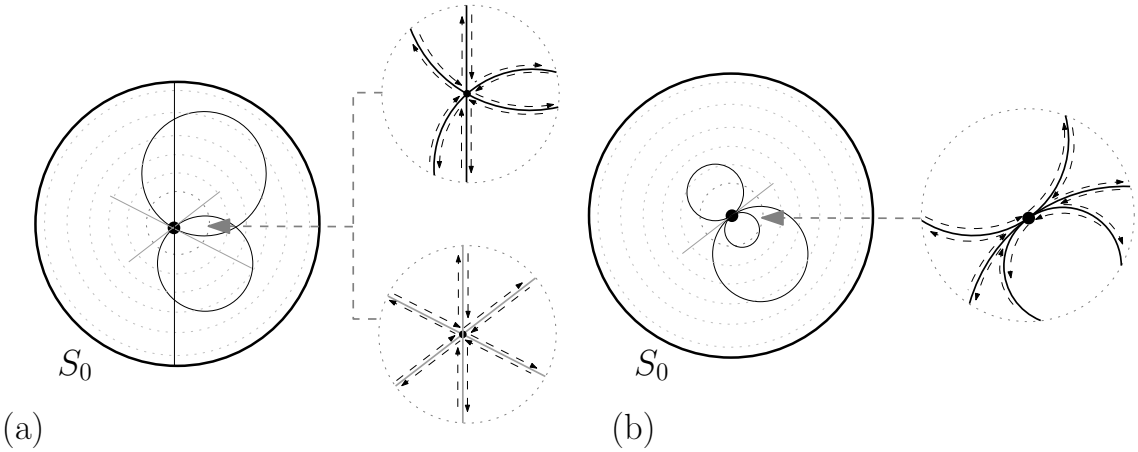
Handling half-edges at poles for a (bi)polar event. To handle half-edges at poles, as illustrated on Fig. 11, when turning around poles with M_θ, we merge consecutively encountered half-edges created at (bi)polar event sites. The algorithm handles cases when

circles are intersected at $\theta = 0$ and when there are tangencies between polar or bipolar circles. The algorithm is straightforward yet tedious, and the reader is referred to section 12. Note that this handling is designed for the ordering of (bi)polar event sites in conflicts according to circle radius, given in def. 7 and def. 8.

Handling half-edges for a bipolar event. Considering $\mathcal{V}$ at a bipolar event site, we first handle half-edges associated to the bipolar circle at the north pole as indicated in the previous paragraph. Next, we take care of bipolar cuts, i.e. intersections between the bipolar circle and active arcs. First suppose that the bipolar event site is not in conflict with any normal event. As depicted in Fig. 12 (a), four half-edges get created and four merges occur. To finish up, half-edges associated to the bipolar circle at the south pole are handled as indicated in the previous paragraph.

For the sake of completeness, a comment is in order in case of conflict between a normal and a bipolar event sites. In that case, the bipolar circle provides half-edges to be connected with e_L^u,e_R^u, e_L^l and e_R^l. See Fig. 12 (b) and algorithm 4 for the details.

Figure 11 Operations on half-edges at polar event. Tangents are in grey:(a) classical case with two polar and one bipolar circles (b) tangency case of three polar circles

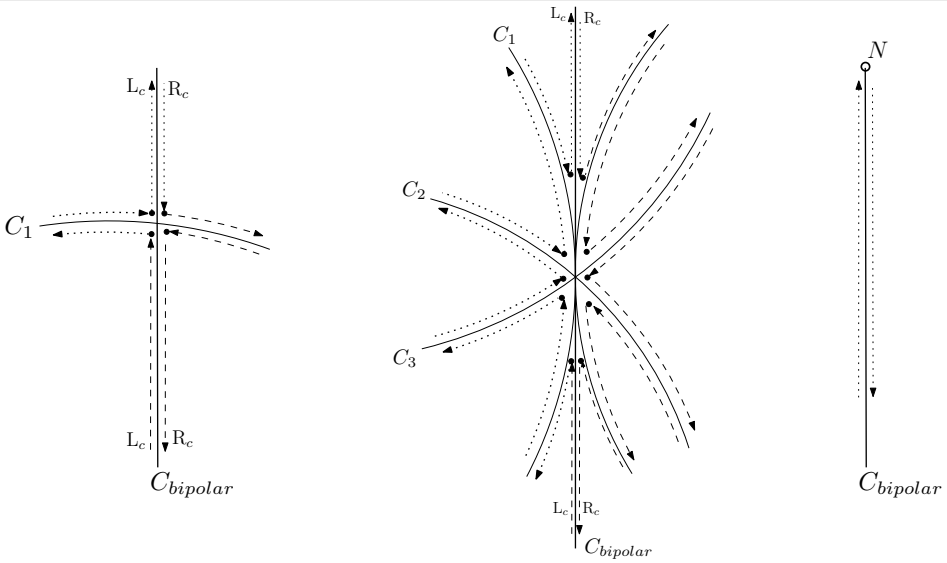


8.3 Building the faces

Building the faces of the arrangement subsumes two steps, namely creating CCB and faces. To do so, we resort to two independent union-find algorithms.

Creating a CCB. A SCH becomes a CCB whenever the merge step of two half-edges creates a topological circle. To detect this, we use a union-find algorithm, which requires

Figure 12 Operations on half-edges at bipolar event. Dotted half-edges are the old active one, dashed half-edges are the new ones created. A black dot represents a merge. (a) Example of arc crossed by a bipolar circle ; (b) Example of normal event site with same event point on a bipolar circle; (c) Two half-edges created at a bipolar critical event



endowing each half-edge with a pointer to a master half-edge. A merge of half-edges makes their master become the same, and a CCB appears when the two half-edges being merged already have the same master half-edge.

Creating a face. A face consists of a principal CCB and of CCB defining holes. We construct faces using union-find on SCH. The pointer used along the process is the master SCH pointer. While closing a SCH, the resulting CCB is principal if it is its own master; if not, the CCB becomes a hole of the master. Whenever two arcs are adjacent in $\mathcal{V}$, a pair of active adjacent half-edges bounds the same face. But as the intersection between a face and the meridian may feature several connected components, we merge these components using union-find. Phrased differently, the same face is started from different points, and the corresponding components are eventually merged —see Fig. 13 for an illustration. (Notice a merge step is also performed upon completion of the sweep at $\theta = 2\pi^-$.) The only way for two components to become adjacent is to be merged at an end point. So at every non-degenerate end point only involving ending circles, we must test if the relevant half-edges contribute to the same face, and if so union their master SCH.

Having explained the union-find process, we complete the description by the initialization of master SCH. To set the pointer of a newly created SCH, we face to options: the SCH either contributes to a face in progress or starts a new face. To describe both options, observe a new SCH s_{ch} is created with one or two half-edges: $s_{ch} = \{e_1\}$ for a (bi)polar circle critical event and during the initialization at $\theta = 2\pi^-$; $s_{ch} = \{e_1, e_2\}$ for a normal event site, in which case we assume the arc of e_1 is above the arc of e_2 in $\mathcal{V}$, and for a bipolar cut, in which case we assume e_2 in on the bipolar circle. The master assignment works as follows.

▷ **1.** If e_1 is not associated to a bipolar circle :
— If e_1 is the upper half-edge of its arc [9], let A_i be the arc above the arc of e_1 in $\mathcal{V}$. The pointer of s_{ch} is set to the master of the SCH of the lower half-edge associated to A_i. A comment is in order if A_i is the north pole: if a face containing the north pole exists, the master sought is that corresponding to this face; if not, S_{ch} starts the face containing the north pole. See table 1 for a description of events creating/changing the face containing the north pole.
— If e_1 is the lower half-edge of its arc, s_{ch} becomes its own master.

▷ **2.** If e_1 is associated to a bipolar circle, e_1 is created at a bipolar critical event and is anchored at the north pole (case (c) of fig. 12). As can be seen from table 1, the associated SCH starts a new face containing the north pole.

[9] This is due to the fact at initialization, we first create half-edge (upper and then lower) for arc from top to bottom in $\mathcal{V}$ —excluding poles.

Figure 13 Building faces using union-find. $SCH_1, SCH_2, SCH_3, SCH_4$ define the same face. At i_1 and i_2, we start two new faces with SCH_1 and SCH_2. Before the meridian reaches ep, two faces are defined, both with two SCH. After ep, one face with three SCH remains. This merge is achieved using union-find.

Table 1 Modifying current face containing the north pole

When	Half-edge starting the current face containing the north pole
In `topo_init_arrangement`	upper half-edge of the highest arc active at $\theta = 0$
At bipolar and north polar start event	Current inner half-edge
At bipolar and north polar end event	Current outer half-edge
At any other start event	Current outer half-edge if none of the three events above occurred

Figure 14 Two normal circles inducing the creation of a face with a hole. At θ_0 : e_1 and e_2 define a new SCH and start a new face, say F; at θ_1 : e_3 and e_4 define a new SCH, which is associated to face F; at θ_2 : the SCH defined by e_3 and e_4 becomes a CCB which becomes a hole of F; at θ_3 : the SCH defined by e_1 and e_2 becomes the principal CCB of F.

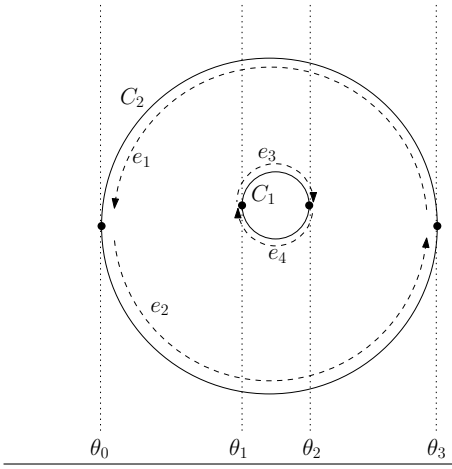


Figure 15 Illustration of the use of Union-find to define a face; Four normal circles in S_0; When creating a SCH , we look for a master pointer : represented by dotted oriented arrows.

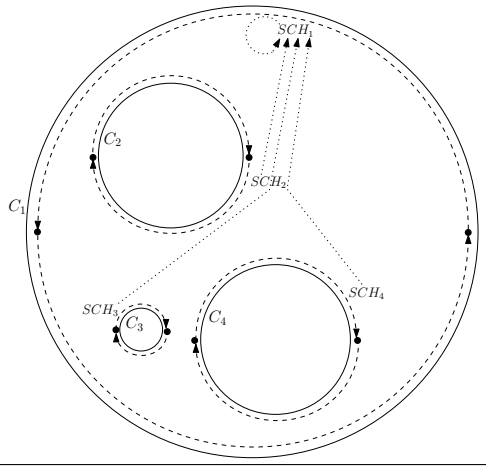
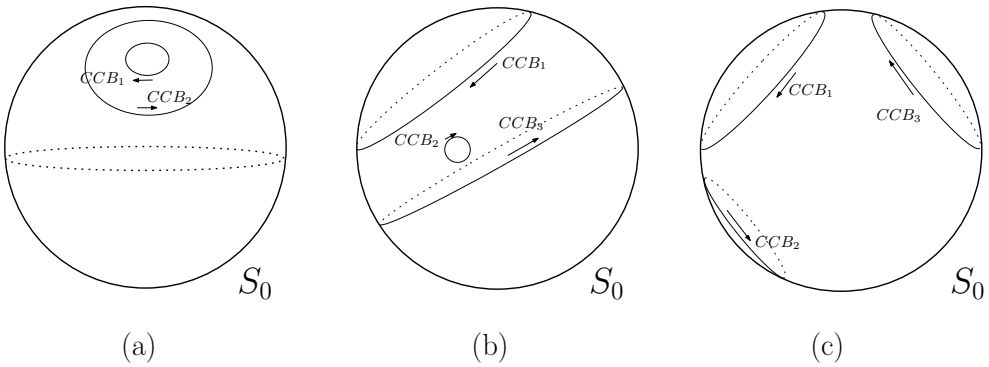


Figure 16 Definition of a face with a set of CCB. Each arrow represents a CCB. In the four cases, only one face is defined by the CCB represented



8.4 Complexity analysis

To conclude, we analyse the cost of constructing the HDS storing the arrangement. The analysis consists of counting the number of find and union operations used to maintain the topological data structures i.e. SCH/CCB and faces. Denoting α the inverse of Ackermann's function, recall that the complexity of performing M union-find operations on a N elements set, with $M \geq N$, is $\Theta(M\alpha(M, N))$ [Tar83].

Theorem. 3 *Constructing CCB has complexity* $\Theta((e+n)\alpha(6e+12n, 2e+4n))$.

Proof. The e edges of the arrangement yield $2e$ half-edges. The n_0 arcs intersected at initialization yield $2n_0 \leq 4n$ half-edges. The number of half-edges manipulated in the algorithm is thus at most $2e+4n$. To count the number of union and find operations, notice whenever two half-edges are merged to bound a face, we have three elementary operations: two find and one union. A vertex of the arrangement adjacent to m_p edges requires $3m_p$ operations, whence a total number of $6e$ operations. The merge steps at $\theta = 2\pi^-$ require 3 operations per pair of half-edges, so $6n_0 \leq 12n$ operations. The total number of operations is thus $6(e+2n)$. Whence an overall complexity of $\Theta((e+n)\alpha(6e+12n, 2e+4n))$. □

Theorem. 4 *Grouping CCB into faces has complexity* $\Theta(n\alpha(10n+3, 6n+2))$.

Proof. Since there are at most $2n$ arcs in $\mathcal{V}$, at any time, at most $2n+1$ faces crossed by the meridian can be started and not closed. If we now add faces started at $\theta = 0$, which are not already closed, we never manipulate more than $2(n+n_0+1) \leq 2(n+2n+1)$ master SCH for operations of union and find. Counting the number of operations, first consider merging at $\theta = 2\pi^-$. Since adjacent half-edges are always associated to the same master SCH, we just have to do two find and one union operation per face defined, i.e. $3(2n+1)$ operations. Then relevent event points modifying the connectivity of a face along the meridian are start and end point. At a start point, we make one find to attach a new SCH to a master SCH (notice the corresponding find may be non trivial). At an end point, we must make at most 2 find and 1 union. This leads to $6n+3+3n+n = 10n+3$ operations. The overall complexity is $\Theta(n\alpha(10n+3, 6n+2))$. □

9 Reporting inclusion into spheres

Assume the n intersection circles are generated by m balls, that is, for each intersection circle $C_i = S_0 \cap S_i$, there exists a ball B_i bounded by sphere S_i. In this section, we describe an algorithm reporting the balls covering each face of the arrangement in S_0. Notice that (i) if a sphere is tangent to S_0 the intersection reduces to a point, and if S_0 is covered by the associated ball, so are all the faces of the arrangement (ii) if two spheres intersect S_0 along the same circle, and their balls cover (do not cover) the same part of S_0, a face covered by one is covered by the other (is not covered by the other).

9.1 Filtering spheres before the sweep process

To describe the BO algorithm, we assumed the circles were pairwise different. To meet this requirement, we sort the n intersecting spheres using a total ordering returning equality when two spheres intersect S_0 along the same circle. We also equip each unique intersection circle two special balls: the *primary* ball, which is associated to the sphere which first defines a given circle; and the *opposite* ball, such that its sphere is the first which intersects S_0 along the same circle, but is opposite to the primary wrt to the plane containing the circle. Notice the opposite ball may not exist. Moreover, we maintain two types of pointers: one for each circle referring to the primary and opposite balls, and one for each primary/opposite ball referring to the collection of balls covering the same region of S_0. Geometric predicates to order two spheres, and in case of equality, to state whether their balls cover the same region of S_0 are developed in the companion paper. Using these predicates, associating the previous pointers and determining the set of intersection circles requires $O(m \log m)$ time.

9.2 Refinement notions for threaded and bipolar circle

To report balls covering faces of the arrangement, we need to refine several notions which were not used for constructing the arrangement : the notions of upper and lower arcs for threaded circles —section 4.1; the notion of spherical cap with smallest area induced by a great circle (threaded or bipolar) —section 8.1.

We call a *north* threaded circle a threaded circle whose center has a z coordinate larger than or equal to that of the center of S_0. Remaining threaded circles are *south* threaded circles. Recall that a north (south) polar circle has a single non trivial arc, which is lower (upper) —the other arc being represented as the pole itself. Similarly, we consider that a north (south) threaded circle has a single arc which is lower (upper).

The spherical cap of smallest area is not defined if bounded by a great circle. A ball producing a north (south) threaded great circle is considered to cover the smallest (biggest) part of S_0. For a bipolar circle, the caps swept between θ_S and θ_E by M_θ is the one of smallest area. With these extensions, one has:

Observation. 3 *The spherical cap of smallest (biggest) area bounded by a circle is described by inner (outer) half-edges of the circle. The lower/upper half-edge of an upper (lower) arc is always inner/outer (outer/inner).*

Using observation 3, it is clear that when an inner (outer) half-edge of a circle C describes a face F, any ball whose sphere intersects S_0 along C, which covers the smallest (biggest) part of S_0 bounded by C, also covers F. Moreover, using the upper or lower status or an arc A in $\mathcal{V}$, one can locate which side of S_0 w.r.t. A is described by its current inner/outer half-edges.

9.3 Inclusion into spheres

A collection of covering balls is associated to each master SCH, as such a master describes a face. The algorithm setting these collections relies upon two ingredients. First, a collection LNB containing selected balls, initialized before the sweep process and updated when processing events. Second, an incremental update of the collections and of LNB.

The LNB collection. The top-down processing of $\mathcal{V}$ governs the updates of faces of the arrangement. Consequently, the update of collections is an iterative process relying on this top-down processing. This iterative process relies on initial informations, namely the collection of balls covering the north pole in the half-space orthogonal to M_θ and containing M_θ. This collection of balls, denoted LNB, is initialized when filtering spheres to spot intersection circles, and when classifying circles. This collection is further updated when a critical point of polar or bipolar circle is encountered –as such circles locally start covering the neighborhood of the north pole in the half-space orthogonal to M_θ and containing M_θ. Details of these operations are provided in table 2.

Lists of balls covering: explicit encoding. In this paragraph, we describe a strategy to explicitly construct the collection of balls covering a face associated to a new SCH —see the next paragraph for an alternative. To present the construction, we follow the notations of section 8.3 regarding the half-edges involved in the creation of a SCH.
▷ **1.** Half-edge e_1 is not associated to a bipolar circle. Denoting A the arc associated to e_1, we consider two cases:
—**e_1 is the upper half-edge of arc A.** If A is not below the north pole in $\mathcal{V}$, e_1 describes the same face than the lower half-edge of the arc above A : as arcs in $\mathcal{V}$ are processed top-down, the collection of balls already exists. If A is below the north pole , we deal with the face containing the north pole. If this face has not already been started, i.e. its master does not exist, the collection of relevant balls are those in LNB.
—**e_1 is the lower half-edge of arc A.** Let L_B, be the collection of balls associated to the face of the SCH of the upper half-edge of A. If A is an upper (a lower) arc, e_1 is inner (outer). Therefore, using balls of spheres intersecting S_0 along the circle of A, the collection of balls covering the new face is obtained by (i) adding to (removing from) L_B those covering the smallest part of S_0 bounded by the circle of A and (ii) removing (adding to) from L_B those covering the biggest part. Note that balls covering biggest part removed have been inserted while initializing LNB (see table 2).
▷ **2.** Half-edge e_1 is associated to a bipolar circle. Recall that a new SCH defined by only one half-edge of a bipolar circle (always anchored at the north pole) contributes to define the face containing the north pole. So the collection of balls associated is filled with the collection of balls covering the north pole before/after the update shown in table 2 if the half-edge is outer/inner (inner/outer) at a bipolar start event (bipolar end event). See Fig. 12.

A comment is in order when a merge between two SCH is performed by the union-find algorithm. In this case, the collection of balls of the node pointing to the master is deleted.

Table 2 Balls in LNB. The ball/sphere/circle processed is denoted $B/S/C$. Function $\texttt{EBP}(C)$ ($\texttt{ESP}(C)$) returns, if any, the balls covering the biggest (smallest) part of S_0 bounded by C. Operator + (-) means the balls are added to (removed from) LNB

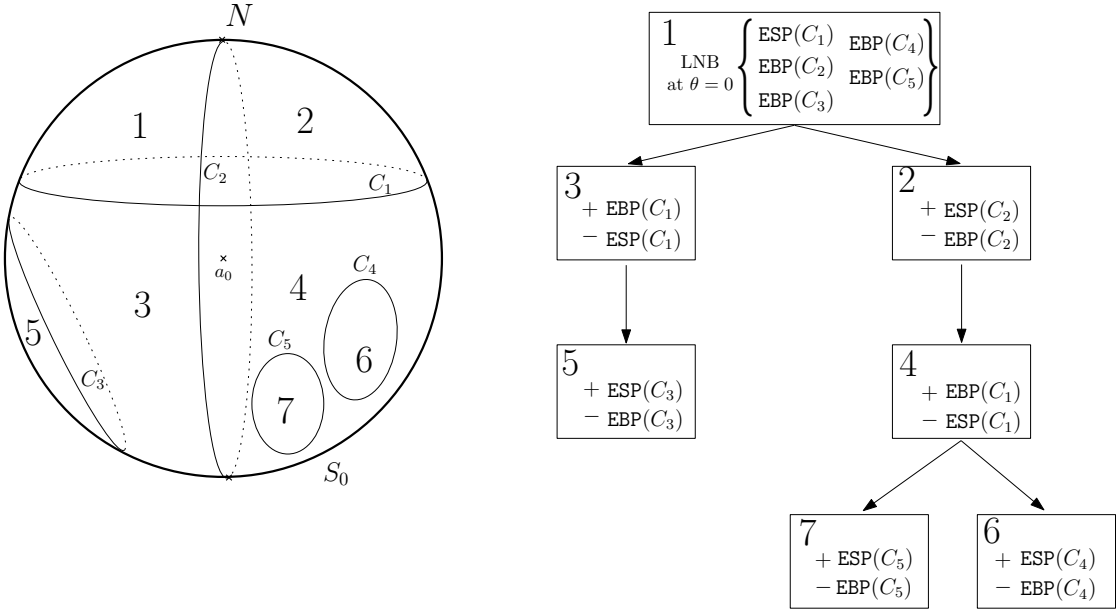
Steps	type of event (exclusive cases)	Actions on LNB
(1) Filtering spheres	$S \cap S_0 =$ a point and $S_0 \subset B$	$+ B$
	B covers the smallest part of S_0 and B covers the N pole	$+ B$
	B covers the biggest part of S_0	$+ B$
(2) Classifying circles	North polar circle with $\theta_E < \theta_S$	+ $\texttt{ESP}(C)$ and - $\texttt{EBP}(C)$
(3) Handling events	North polar or bipolar circle Start event	+ $\texttt{ESP}(C)$ and - $\texttt{EBP}(C)$
	North polar or bipolar circle End event	+ $\texttt{EBP}(C)$ and - $\texttt{ESP}(C)$

Collections of covering balls: implicit encoding. The previous strategy can be improved to factor out common collections of balls shared by faces, resulting in a hierarchical non redundant encoding of the collections.

To begin with, observe primary and opposite balls associated to a circle C are sufficient to recover the balls covering a region of S_0 bounded by C, as the other balls can be retrieved from the data structures introduced in section 9.1. Therefore, the construction of the collections can focus on primary and opposite balls. This said, while walking down $\mathcal{V}$, we have seen a collection of balls associated to a face is obtained using a previously defined collection by adding/removing the primary/opposite or the opposite/primary ball of one circle. So the collections of balls covering the faces can be represented as a tree whose root is LNB at $\theta = 0$, with one node per face of the arrangement. Walking down the tree, a son node, always associated to an arc. For such a node, one just needs to specify whether the primary/opposite or the opposite/primary ball must be added to/removed from the collection of balls of its father node. Note that a change in LNB follows this pattern, as such a change features one circle and is associated to a new face. At the end of the sweep process, the number of nodes is exactly the number of faces in the arrangement. Moreover, if n stands for the number of intersection circles, the maximum distance between the root and a node is $2n$ —as $\mathcal{V}$ never contains more than $2n$ arcs. An example such tree is presented on Fig. 17.

As for the previous strategy, a comment is in order for the merge between two SCH with different master SCH. In this case, sons of the *dominated* node become sons of the node associated to the remaining master.

Figure 17 Implicit encoding of covering balls. Circle C_1 is a north threaded circle and C_2 a bipolar one, while C_3, C_4 and C_5 are normal ones. Faces of the arrangement depicted on the left are identified using numbers from 1 to 7. Notice that node associated to face 2 reflects the modification of LNB induced by bipolar circle C_2.



9.4 Complexity analysis

To analyze the complexities of the implicit encoding, n_0 denotes the number of arcs intersected by M_0, m the number of input balls, m_{N_0} number of input balls –either primary or opposite— covering the face including the north pole at $\theta = 0$.

Theorem. 5 *Denote $s_T(F)$ total number of balls covering face F. Implicitly encoding the covering of all faces of the arrangement requires $O(f + m)$ time and $O(f + m)$ space. Constructing individual lists for all the faces can be done in $O(\sum_{F \in faces} s_T(F) + f)$ time.*

Proof. Let us first analyze the time complexity. Independently of the number of faces, considering $\mathcal{V}$ at $\theta = 0$, each pair of adjacent half-edges (also considering poles) at $\theta = 0$ defines a face and we compute a collection of balls for each of them. Then, a collection of balls is computed every time we think a SCH creates a new face. Since two such SCH defining the same face are always merged at an end point, we never compute more than $2n$ time superflous collections. So the maximum number of master SCH manipulated is never more than $f + 2n + n_0 + 1$. Constructing the collection of balls covering the face including the north pole at $\theta = 0$ requires m_{N_0} insertions. Constructing the remaining collections involves a constant number of operations. The overall cost is thus $O(f + n + n_0 + m_{N_0})$.

Consider now the space requirements, which stem from the size of the tree and the storage of balls. The former is $(f + 2n + n_0) + m_{N_0}$, one node represents a face started, and there are m_{N_0} nodes corresponding to faces intersected by M_0. The later is m as each sphere is counted once —a ball is either primary/opposite or is listed in a collection accounted for by a primary/opposite. Overall storage size is thus $O(f + m)$.

Finally, consider the cost of constructing the individual collections —one per face. Let $s(F)$ stand for the number of primary or opposite balls covering face F, and $s_T(F)$ be the total number of balls covering face F.

First, we deal with primary/opposite balls. Each collection is stored in a set encoded in a dictionary providing logarithmic time updates. Constructing the set of the tree root requires $\mathrm{O}(m_{N_0} \log m_{N_0})$ time. Next, a breadth-first over the tree is used to construct the set of primary/opposite balls of each node from that of its father [10], which requires adding/removing at most one ball. The copy of the set of balls of its father is linear, but addition/removal are logarithmic. Summing up over all the faces of the arrangement yields an overall cost of

$$O(\sum_{F \in faces} (s(F) + 1 + 2\log(s(F) + 1))) \quad \text{which is} \quad O(\sum_{F \in faces} s(F) + f).$$

Second, we complete the sets from balls covering the same region of S_0 as primary/opposite from the data structures introduced in section 9.1, a linear time operation. We finally obtain that building the collections of balls covering the faces of the arrangement is $O(\sum_{F \in faces} s_T(F) + f)$ time. □

10 Experimental results

10.1 Setup

Implementation sketch. The code is written in CGAL style —see `www.cgal.org`, with three main classes. Given a collection of balls, the first one provides a function reporting the balls intersecting a given ball. A grid templated by a traits class specific to balls is used: the grid spacing is taken as twice the maximum radius of balls, so that only balls whose center are in same cube or in one adjacent cube can intersect. Given a specific ball/sphere, the second class computes the arrangement of circles, and reports the balls covering each face of the arrangement. This class is templated by a geometric kernel and by the HDS used to store the arrangement. The third class, templated by the HDS, implements the Gauss-Bonnet formula to compute the surface area of a spherical cap.

Predicates and number types. As reported in the companion paper, all predicates involved in our algorithm rely on algebraic numbers of degree two. As these predicates are not

[10] The root of the tree contains primary/opposite balls covering the face including the north pole at $\theta = 0$

already in a CGAL kernel, we implemented them using the number types provided by CGAL kernels. The exact algorithm uses the `Exact_predicates_exact_constructions_kernel`, which provides a filtering strategy resorting to `Lazy_Exact_NT<Gmpq>` number type, which is a filtered version of `Gmpq` using intervals : exact computation is done using `Gmpq` when intervals are not sufficient to conclude. The algorithm can also be instantiated with the `Exact_predicates_inexact_constructions_kernel`, whose number type is a plain `double`. In the following, *exact* and *double* refer to these two kernels.

Datasets and statistics. We first report results on random arrangements, a random circle in S_0 being specified by picking uniformly at random its center in B_0. We also report results on molecular models (single proteins, protein - protein complexes) retrieved from the Protein Data Bank at `www.rcsb.org`. Given a PDB file, hydrogen atoms are discarded. Thus, in the VdW model, radii used for the heavy atoms are the so-called group radii (Notice these radii influence the number of neighbors.). In the *solvent accessible model*, these VdW radii are expanded by the radius r_w of a water probe, usually taken as $r_w = 1.4\mathring{A}$. For complexes, water molecules having neighbors on the two partners are also kept provided their temperature factor is small [CP+06]. Atoms' centers and radii are rational numbers. In order to avoid coordinate changes, we set the north and south poles of S_0 as the intersection of S_0 with the z-axis.

When reporting running times, we distinguish between the pre-processing time t_p used to report neighbors, the time t_a used to compute the arrangement, the time t_s used to compute surface areas. Tests have been conducted on a bi-proc. Pentium(R) IV CPU at 3.06GHz with 2.5 GB of memory. Notice though, that the code is not written to use several processors.

10.2 Results and illustrations

Numerics. As shown on Tables 3 and 4, the ratio between *double* and *exact* arithmetics is of 10-12. But calculations in double fail on (nearly-)degenerate inputs. One such highly degenerate example is displayed on Fig. 18. We compute the exact arrangement, a result beyond reach of explicit perturbations. Failures also happen for molecular models, since 6 atoms had to be removed to get Table 3.

Numerics may be improved in two ways. First, we may use the new CGAL `Lazy_kernel`. This kernel consists of filtering at the predicate level, which may provide a gain factor of two wrt filtering at the number type level as reported in [FP06]. Second, we may design static filters. For $3D$ Delaunay triangulations, such filters yield a modest 30% overhead wrt a double arithmetic [MP05]. Notice these two lines are independent, as the constructions allowed by lazy kernel may be inter-twined with static filters to re-use intermediate calculations.

Algorithm complexity. To assess the value of the constant in the algorithm overall complexity, we use random arrangements, see Fig. 19. With either double or exact arithmetic, the theoretical curve $c(n+k)\log n$ is in excellent agreement with the experimental curves,

the value of c being set to match the experimental time obtained for $n = 300$ and the corresponding k.

Comparison with existent work. Table 4 compares our algorithm with the method based on explicit perturbations of spheres [HS98, EH03]. For their tests, a bi-proc Pentium III(R) 1GHz with 2GB of RAM computer has been used —we use a bi-proc Pentium VI(R) 3.06Ghz with 2.5GB of RAM computer. As the runtime does not scale linearly with the CPU clock frequency, our code seems faster (is slower) with the double (exact) arithmetic. But as mentioned when discussing numerics, a significant progression margin remains in the exact realm.

10.3 Bio-chemistry: mining multi-body interactions

Context. As recalled in introduction, our developments come from the need to model multi-body interactions in structural biology. This modeling, usually undertaken in the solvent accessible model, has essentially focused so far on pairwise contacts —between atoms or residues. But as mentioned in introduction, pairs of contacts fall short from accounting for the complexity of atomic environments [CP$^+$06]. In the following, we give a taste of a new model dissecting molecular surfaces as *Exposed, Self, Buried, Interaction* (ESBI). Given an atom of a particular type (e.g. the carbon atom of the peptide bond), this model is meant to investigate equivalence classes of atomic environments accounting for multi-body contacts. A complete report on this topic is being written [BCL06].

Consider a macro-molecular complex featuring two partners A and B, in the solvent accessible model. Recall the *Solvent Accessible Surface area* (SASA) of a partner is the surface area of the boundary of the union of its atomic balls. The contribution of a given atom to this surface is easily retrieved from the arrangement of intersection circles with its neighbors, provided one maintains the collection of balls covering a face of the arrangement. Also recall the *Buried Surface Area* (BSA) of the complex, which characterizes the extent of the interaction between partners, is defined as $BSA(A \cup B) = SASA(A) + SASA(B) - SASA(A \cup B)$. To extend this notion, define the *Interaction Surface Area* (ISA) of partner A (B) as the surface area of the spherical caps of atoms of A (B), which are covered by atoms of A (B) when molecule A (B) is alone, yet get also covered by atoms of B (A) in the complex. Define similarly the ISA of the complex as $ISA(A \cup B) = ISA(A) + ISA(B)$.

Highlights. We focus on the 96 protein - protein complexes from [CP$^+$06], split into fives classes : Protease - Inhibitor (PI), Enzyme - Substrate (ES), Antibody - Antigen (AA), Signal Transduction (ST), Misc (M). Out of these complexes, the *high resolution* set features 30 complexes at a resolution of $2\mathring{A}$ or better. For the high resolution (whole) set, crystallographic water molecules are used (discarded) to compute the arrangements, as at such resolution, these molecules are spotted reliably enough (un-reliably) in the crystal. A detailed discussion of results is beyond the scope of this paper, and we just report two striking observations.

First, a major parameter of interest in previous studies of interfaces has been the BSA [CJ02, BCRJ04]. As reported in Table 5, the ISA is about 4 times larger. This dramatic change encodes the complex interactions between atoms of the partners, which are non covalent contacts across the interface, as well as covalent and non-covalent contacts within a partner. Second, we noticed above that calculations in the solvent accessible model were carried out with a water probe of $r_w = 1.4\mathring{A}$ —a rather arbitrary value. As reported on Fig. 20, varying r_w in the range $0 \dots 2\mathring{A}$ yields an important variation on the number of neighbors. Most interestingly, the curves obtained for 3 complexes from different classes 1acb (PI, 2433 atoms), 1vfb (AA, 2779), 1got(ST, 3449) are almost identical. In the same vein, the curves featuring the runtime per atom are almost identical Fig. 21. Mining these variations to understand which informations are of *universal* bio-chemical value, and which are family specific, should provide important insights on the complexity of atomic environments, and on the formation of interfaces in docking.

10.4 Illustrations

Table 3 Double versus exact arithmetics for complex 1acb (2433 atoms): run-times in the VdW model

NT	**neighbor**	**argt**	**area**	**total**
double	0.05s	2.76s	0.88s	3.69s
exact	2.43s	35.78s	11.85s	50.06s

Figure 18 Left : degenerate arrangement of 47 circles. Right: zoom about the point $(r_0, 0, 0)$: 6 different directions of tangency for 13 circles. This point is a start, end, intersection point for 1,1,10 circles, and is crossed by a bipolar circle. Euler characteristic reads as $674 - 1384 + 712 = 2$.

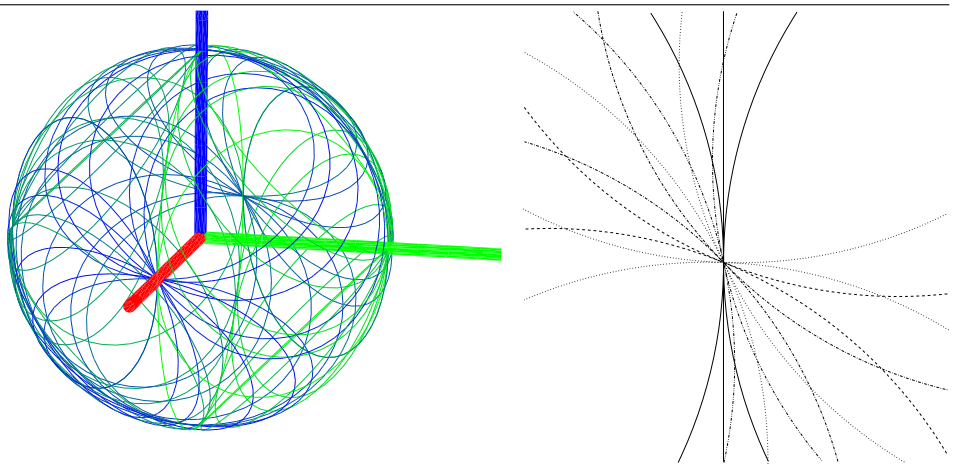


Figure 19 Observed vs theoretical complexities for random arrangements

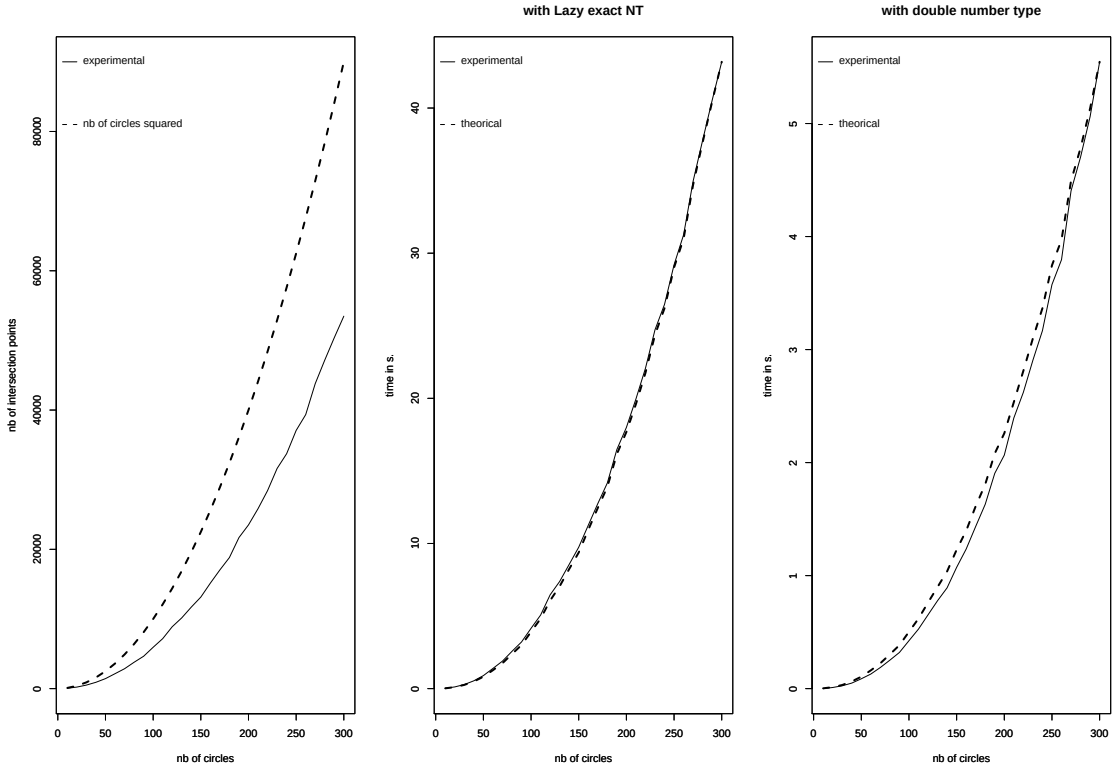


Table 4 Total time (in seconds) of computing the surface (including the perturbation) vs. total time of computing arrangement of circles in each atomic spheres.

Input file	#atoms	[EH05, EH03]	Double	Exact
1bzm	2049	8.31	2.62	33.87
1emt	3233	13.42	3.77	54.87
1jky	5734	26.94	7.78	108.62
7at1	7169	28.40	8.19	119.3
1l7x	12912	54.50	16.44	231.22

Table 5 Surface areas ratios: ISA/BSA for the whole set and the high resolution set

Water	#complexes	min	max	mean	median	std dev
Without	96	2.74	5.66	4.59	4.53	0.57
With	30	3.32	4.39	3.77	3.70	0.26

Figure 20 Variation of the number of neighbors for the 3 complexes 1acb, 1vfb, 1got

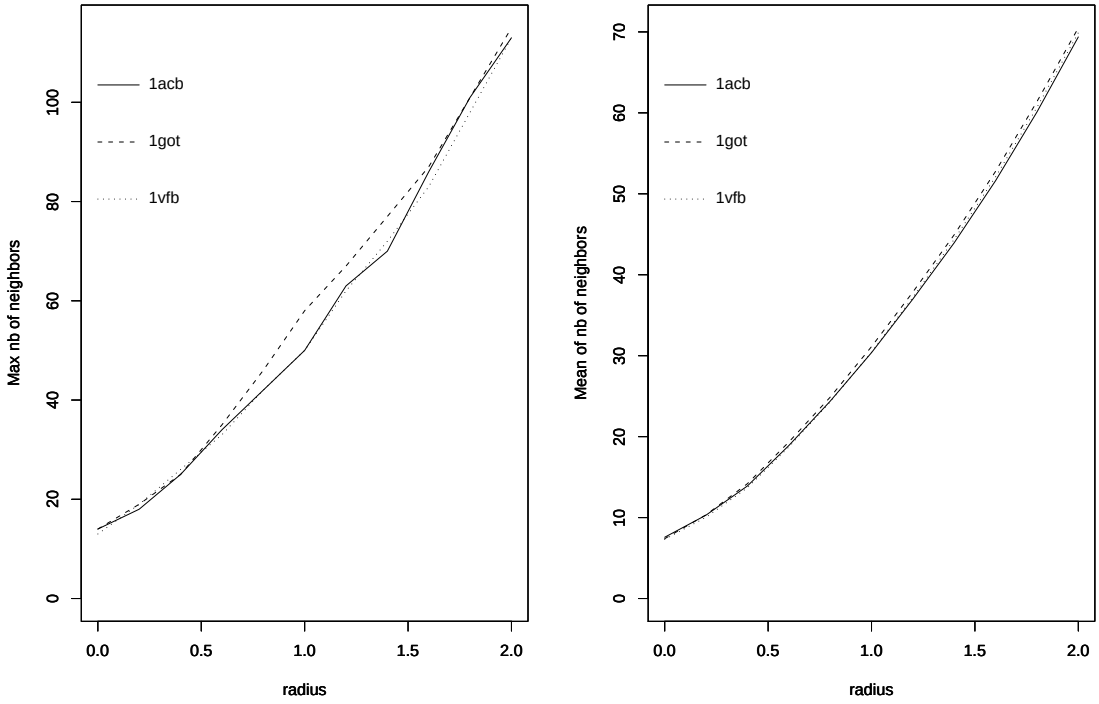
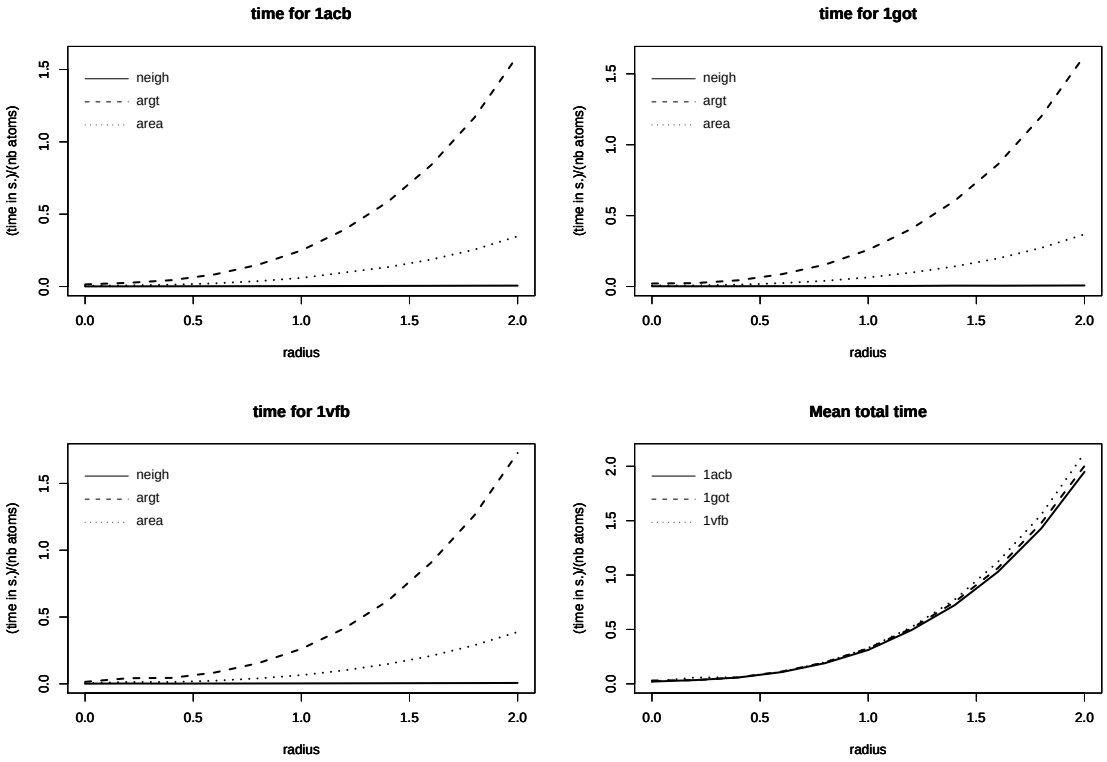


Figure 21 Running times per atom for the 3 complexes 1acb, 1vfb, 1got



11 Conclusion

This paper presents a generalization of the Bentley-Ottmann algorithm to compute the exact arrangement of circles in a sphere. The algorithm is non trivial due to the degeneracies and the algebraic specification of events, and departs from previous work whose robustness stems from explicit perturbations.

On the numerical side, arithmetic profiling shows one order of magnitude may be gained in speed by fine-tuning the predicates, constructions and filters used. On the algorithmic side, arrangements of spheres being isotropic, an important topic in the future will be to compute arrangements of ellipsoids, so as to account for anisotropic interactions between atoms. Finally, on the application side, the algorithm is being used in structural biology to investigate molecular surface models going beyond the traditional exposed and buried surface areas. As an illustration, statistics featuring spectacular changes wrt traditional

observations on protein - protein complexes are provided. Insights gained in modeling such multi-body contacts may prove essential for a number of issues.

Acknowledgments. I. Emiris, S. Pion and E.Tsigaridas are acknowledged for discussions on the numerical issues.

References

[AE96] N. Akkiraju and H. Edelsbrunner. Triangulating the surface of a molecule. *Discrete Appl. Math.*, 71:5–22, 1996.

[ALPS01] N. Alon, H. Last, R. Pinchasi, and M. Sharir. On the complexity of arrangements of circles in the plane. In *Discrete and Comput. Geometry 26*, pages 465–492., 2001.

[AT97] R. Abagyan and M. Totrov. Contact area difference (cad): A robust measure to evaluate accuracy of protein models. *J. Mol. Biol.*, 268, 1997.

[BCL06] J. Bernauer, F Cazals, and S. Loriot. Exposed, self, buried, and interaction surfaces: towards a dissection of multi-body inter-atomic interactions. 2006.

[BCRJ04] R.P. Bahadur, P. Chakrabarti, F. Rodier, and J. Janin. A dissection of specific and non-specific protein-protein interfaces. *J. Mol. Bio.*, 336, 2004.

[BDD92] J-D. Boissonnat, O. Devillers, and J. Duquesne. Computing connolly surfaces. In *IFIP Conf. on algorithms and efficient computation*, 1992.

[BPS+03] C. Bajaj, V. Pascucci, A. Shamir, R. Holt, and A. Netravali. Dynamic maintenance and visualization of molecular surfaces. *Discrete Applied Mathematics*, 127:23–51, 2003.

[BY98] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic Geometry*. Cambridge University Press, UK, 1998. Translated by Hervé Brönnimann.

[CDES01] H-L. Cheng, T. K. Dey, H. Edelsbrunner, and J. Sullivan. Dynamic skin triangulation. *Discrete Comput. Geom.*, 25:525–568, 2001.

[Cha05] D. Chandler. Interfaces and the driving force of hydrophobic assembly. *Nature*, 437:640–647, 2005.

[CJ02] P. Chakrabarti and J. Janin. Dissecting protein-protein recognition sites. *Proteins*, 47, 2002.

[CL] F. Cazals and S. Loriot. Circles on a sphere: algebra and precicates. In preparation.

[Con83] M. L. Connolly. Analytical molecular surface calculation. *J. Appl. Crystallogr.*, 16, 1983.

[Con85] M. L. Connolly. Molecular surface triangulation. *J. Appl. Crystallogr.*, 18, 1985.

[Con96] M. Connolly. Molecular surfaces: A review. *Network Science*, 14, 1996.

[CP+06] F. Cazals, F. Proust, , R. Bahadur, and J. Janin. Revisiting the voronoi description of protein-protein interfaces. *Protein Science*, 15(9):2082–2092, 2006.

[dBvKOS97] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[Ede92] H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1760, Dept. Comput. Sci., Univ. Illinois, Urbana, IL, 1992.

[Ede95] H. Edelsbrunner. The union of balls and its dual shape. *Discrete Comput. Geom.*, 13:415–440, 1995.

[EH03] E. Eyal and D. Halperin. Improved implementation of controlled perturbation for arrangement of spheres. In *ECG techreport*, pages ECG–TR–363208–01, 2003.

[EH05] E. Eyal and D. Halperin. Dynamic maintenance of molecular surfaces under conformational changes. In *Proc. 21st ACM Symposium on Computational Geometry*, pages 45–54, 2005.

[EK03] H. Edelsbrunner and P. Koehl. The weighted-volume derivative of a space-filling diagram. *PNAS*, 100(5), 2003.

[EM86] D. Eisenberg and A.D. McLachlan. Solvation energy in protein folding and binding. *Nature*, 319:199–203, 1986.

[FB98] R. Fraczkiewicz and W. Braun. Exact and efficient analytical calculation of the accessible surface areas and their gradients for macromolecules. *J. Comp. Chem.*, 19(3):319–333, 1998.

[FP06] A. Fabri and S. Pion. A generic lazy evaluation scheme for exact geometric computations. In *Proc. 2nd Library-Centric Software Design*, 2006.

[GK01] Holger Gohlke and Gerhard Klebe. Statistical potentials and scoring functions applied to protein-ligand binding. *Curr. Op. Struct. Biol.*, 11:231–235, 2001.

[HS98] Dan Halperin and Christian R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Comput. Geom. Theory Appl.*, 10:273–287, 1998.

[Las95] R. Laskowski. Surfnet: a program for visualizing molecular surfaces, cavities, and intermolecular interactions. *J. Mol. Graph.*, 13(5), 1995.

[LB01] P. Laug and H. Borouchaki. Molecular surface modeling and meshing. In *International Meshing Roundtable*, pages 31–41, Newport Beach, California, USA, 2001.

[LFSB03] M.S. Lee, M. Feig, F.R. Salsbury, and C.L. Brooks. New analytic approximation to the standard molecular volume definition and its application to generalized born calculations. *J Comput Chem.*, 24(11):1348–56, 2003.

[MJ85] S. Miyazawa and R.L. Jernigan. Estimation of effective interresidue contact energies from protein crystal structures: Quasi-chemical approximation. *Macromolecules*, 18, 1985.

[MP05] G. Melquiond and S. Pion. Formal certification of arithmetic filters for geometric predicates. In *Proceedings of the 17th IMACS World Congress on Computational and Applied Mathematics*, Paris, France, 2005.

[Pet94] M. Petitjean. On the analytical calculation of van der waals surfaces and volumes: Some numerical aspects. *J. Comput. Chem.*, 15(5):507–523, 1994.

[Ric74] F. M. Richards. The interpretation of protein structures: Total volume, group volume distributions and packing density. *Journal of Molecular Biology*, 82:1–14, 1974.

[Ric77] F. M. Richards. Areas, volumes, packing and protein struture. *Ann. Rev. Biophys. Bioeng.*, 6:151–176, 1977.

[RTVC04] D. Rajamani, S. Thiel, S. Vajda, and C.J. Camacho. Anchor residues in protein-protein interactions. *PNAS*, 101:11287–11292, 2004.

[SOS96] M.F. Sanner, A. J. Olson, and J.C. Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38:305–320, 1996.

[TA96] M. Totrov and R. Abagyan. The contour-buildup algorithm to calculate the analytical molecular surface. *J Struct Biol.*, 116(1):138–43, 1996.

[Tar83] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[TRS02] O. Tsodikov, M. Record, and Y. Sergeev. A novel computer program for fast exact calculation of accessible and molecular surface areas and average surface curvature. *J. Comput. Chem.*, 23:600–609, 2002.

[VB93] A. Varshney and F. P. Brooks, Jr. Fast analytical computation of Richards's smooth molecular surface. In G. M. Nielson and D. Bergeron, editors, *IEEE Visualization '93 Proceedings*, pages 300–307, October 1993.

12 Appendix: Handling the topology at (bi)polar events

This section complements the algorithms presented in section 8.2 by providing the pseudo-code for algorithms `topo_handle_polar_event` and `topo_handle_bipolar_event`. In doing so, we assume functions `upper_halfedge`(A_i) and `lower_halfedge`(A_i) return the named half-edges for a given arc A_i. For a polar or a bipolar circle C, *in(C)* (resp. *out(C)*) refers to the current half-edge associated to the inner (outer) half-edge of the arc.

We describe how to proceed for one pole. For each pole, we consider a pair of global variables L_h, P_h pointing on half-edges —Last and Previous half-edges. These two pairs are initialized to `NULL`. When the sweep process is over, i.e. $\mathcal{E}$ is empty, and before launching `topo_merge_virtual_faces`, we merge L_h P_h if they are not still `NULL`.

Polar circle. To handle topological operations when a polar circle starts or ends, we just have to manage the half-edges passing by a pole. This can be done using the function `topo_handle_polar_event` when starting and ending a polar circle.

Bipolar circle. Let us consider an event site of a bipolar circle C. Topological operations to correctly handle this event can be decomposed in three steps.
▷ **1.** We launch the routine `topo_handle_polar_event` for the north pole, in order to anchor at the north pole half-edges of the bipolar circle, and to connect with previously encountered half-edges of other (bi)polar circles.
▷ **2.** We manage intersection of arcs in $\mathcal{V}$ by C (see Fig. 12 for illustration).
▷ **3.** We relaunch the routine `topo_handle_polar_event` but for the south pole.

These operations are summarized in algorithm `handle_bipolar_event`. To simplify its presentation, during intersection of arcs of $\mathcal{V}$ by C we consider two functions that create a new half-edge and return a pointer to it :
— `topo_new_left_halfedge`: returns a pointer to the new half-edge which becomes *out(C)* (*in(C)*) at a start event (end event), with the correct intersection point associated to one extremity of the half-edge.
— `topo_new_right_halfedge`: returns a pointer to the half-edge which becomes *in(C)* (*out(C)*) at a start event (end event), with the correct intersection point associated to one extremity of the half-edge.

Algorithm 3 `topo_handle_polar_event`

```
1: if We handle a start point then
2:    if (P_h=NULL) then
3:       L_h=out(C)
4:    else
5:       merge(P_h,out(C))
6:    end if
7:    P_h=in(C)
8: else
9:    if (P_h=NULL) then
10:      L_h=in(C)
11:   else
12:      merge(P_h,in(C))
13:   end if
14:   P_h=out(C)
15: end if
```

Contents

1 Introduction **3**
1.1 Atoms, spheres, and multi-body interactions . . . 3
1.2 The Arrangement of circles in a sphere . . . 3
1.3 Notations and paper overview . . . 4

2 Molecular surfaces and related topics **5**
2.1 Molecular surfaces . . . 5
2.2 Related topics . . . 7

3 Bentley-Ottmann algorithms **7**

4 Circles and event points **8**
4.1 Circle classification . . . 8
4.2 Points vs events . . . 8
4.3 Filling the event queue $\mathcal{E}$ with event sites . . . 10
4.4 Circles in a sphere : degenerate cases . . . 12

5 Bentley-Ottmann in a sphere **13**
5.1 Algorithm overview . . . 13
5.2 Blocks within a normal event site . . . 13
5.3 Algorithm `break_adjacencies` . . . 15
5.4 Handling event sites . . . 17

Algorithm 4 Algorithm `handle_bipolar_event`

1: L_h=`topo_new_left_halfedge`
2: R_h=`topo_new_right_halfedge`
3: `topo_handle_polar_event`(NORTH_POLE)
4: event_site *evt_pol*=$\mathcal{E}$.`pop`{Polar event site}
5: *current*=iterator pointing on successor of north pole in $\mathcal{V}$
6: *stop*=iterator pointing on south pole
7: **while** (*current* is not pointing on south pole) **do**
8: **if** (*stop* is not pointing on south pole) **then**
9: `handle_event_site`(L_h,R_h)
10: *current*=iterator pointing on lower bounding arc of block defined by lists of $\mathcal{E}$.`top`
11: **end if**
12: **if** (($\mathcal{E} \neq \emptyset$) **AND** ($\mathcal{E}$.`top` and *evt_pol* have same θ value) **AND** ($\mathcal{E}$.`top` is not a polar event site)) **then**
13: *stop*=iterator pointing on upper bounding arc of block defined by lists of $\mathcal{E}$.`top`
14: **else**
15: *stop*=iterator pointing on south pole
16: **end if**
17: **while** (*current* $\neq$*stop*) **do**
18: A=arc pointed by *current*
19: *current*=successor of *current* in $\mathcal{V}$
20: merge(`upper_halfedge`(A),L_h)
21: L_h=`topo_new_left_halfedge`
22: merge(`lower_halfedge`(A),L_h)
23: `upper_halfedge`(A)=new halfedge
24: merge(R_h,`upper_halfedge`(A))
25: `lower_halfedge`(A)=new halfedge
26: R_h=`topo_new_right_halfedge`
27: merge(R_h,`lower_halfedge`(A))
28: **end while**
29: **end while**
30: **while** (($\mathcal{E} \neq \emptyset$) **AND** ($\mathcal{E}$.`top` and *evt_pol* have same θ value) **AND** ($\mathcal{E}$.`top` is not a polar event site)) **do**
31: `handle_event_site`(L_h,R_h){Only start points in the event site}
32: **end while**
33: `topo_handle_polar_event`(SOUTH_POLE)

6 Handling $\mathcal{V}$ **20**
6.1 Initializing $\mathcal{V}$ . . . 20
6.2 Inserting starting arcs into $\mathcal{V}$ . . . 20
6.3 Intersection events and arcs reversing . . . 21

7 Spherical BO algorithm: correctness and complexity analysis **22**

8 Reporting the arrangement in a half-edge data structure **25**
8.1 Describing the arrangement . . . 25
8.2 Handling half-edges . . . 26
8.3 Building the faces . . . 28
8.4 Complexity analysis . . . 33

9 Reporting inclusion into spheres **33**
9.1 Filtering spheres before the sweep process . . . 34
9.2 Refinement notions for threaded and bipolar circle . . . 34
9.3 Inclusion into spheres . . . 35
9.4 Complexity analysis . . . 37

10 Experimental results **38**
10.1 Setup . . . 38
10.2 Results and illustrations . . . 39
10.3 Bio-chemistry: mining multi-body interactions . . . 40
10.4 Illustrations . . . 41

11 Conclusion **44**

12 Appendix: Handling the topology at (bi)polar events **49**

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399