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

Frédéric Cazals, Sebastien Loriot

INRIA

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 – version 2

version initiale Décembre 2006 – version révisée Septembre 2007

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° 6049 – version 2 †— version initiale Décembre 2006 — version révisée Septembre 2007 56 pages

Abstract: Given a collection of circles on a sphere, we adapt the Bentley-Ottmann algorithm to the spherical setting to compute the *exact* arrangement of the circles. The algorithm consists of sweeping the sphere with a meridian, which is non trivial because of the degenerate cases and the algebraic specification of event points.

From an algorithmic perspective, and with respect to general sweep-line algorithms, we investigate a strategy maintaining a linear size event queue. (The algebraic aspects involved in the development of the predicates involved in our algorithm are reported in a companion paper.)

From an implementation perspective, we present the first effective arrangement calculation dealing with general circles on a sphere in an exact fashion, as exactness incurs a mere factor of two with respect to calculations performed using *double* floating point numbers on generic examples. In particular, we stress the importance of maintaining a linear size queue, in conjunction with arithmetic filter failures.

From an application perspective, we present an application in structural biology. Given a collection of atomic balls, we adapt the sweep-line algorithm to report all balls covering a given face of the spherical arrangement on a given atom. This calculation is used to define molecular surface related quantities going beyond the classical exposed and buried solvent accessible surface areas. Spectacular differences w.r.t. traditional observations on protein - protein and protein - drug complexes are also reported.

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

* `firstname.name@sophia.inria.fr`

† Revision of report 6049, triggered by the publication of the companion report *Design of the CGAL Spherical Kernel and application to arrangements of circles on a sphere*

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 donné un ensemble de cercles sur une sphère, nous présentons une adaptation de l'algorithme de Bentley-Ottmann sur la sphère pour calculer l'arrangement *exact* induit par les cercles. L'algorithme consiste à balayer la sphère avec un méridien, ce qui est non trivial en raison des cas dégénérés et de la spécification algébrique des évènements.

D'un point de vue algorithmique dans le registre des algorithmes de balayage, nous développons une stratégie garantissant une taille linéaire de la queue de priorité. (Les aspects algébriques relatifs aux prédicats utilisés par l'algorithme sont présentés dans un travail associé.)

Du point de vue de l'implémentation, cet algorithme est le premier algorithme effectif traitant de façon exacte tous les types de cercles sur une sphère, le surcoût requis pour garantir l'exactitude du résultat étant un facteur deux par rapport à une implémentation avec arithmétique flottante —sur des exemples génériques. En particulier, nous mettons en évidence l'importance de la linéarité de la taille de la queue de priorité afin de n'être pas pénalisé par les échecs de filtres arithmétiques.

Du point de vue applicatif, nous présentons une utilisation de l'arrangement en biologie structurale. Étant données un ensemble de boules atomiques, l'algorithme est adapté de façon a obtenir pour chaque face de l'arrangement sur un atome donné, la liste des boules qui la contiennent. Cette information est utilisée pour définir des quantités allant au-delà des surfaces moléculaires exposées et enfouies. Des différences spectaculaires vis-à-vis des observations classiques sont ainsi obtenues sur des complexes protéine-protéine et protéine-médicament.

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.

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 consists 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 [CPBJ06]. 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, it 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 on a sphere so as to decompose the whole atomic surface, and the second one consists of reporting the atomic balls covering a given face of this arrangement. This decomposition features regions exposed on the molecular surface —which may be retrieved from α-shapes. But it also contains regions which are not exposed on the boundary, and encode higher order multi-body contacts.

Illustrations of insights gained from arrangement based models on protein - protein and protein - drug complexes are provided on Figs. 1 and 2 –refer to section 10 for the details.

Figure 1 (a)A protein - protein complex —PDB code 1vfb: protein chains and the two layers of interface atoms from the ESBI model –see section 10. Atoms with light colors feature buried regions, while atoms with dark colors do not –and are missed by previous models. Structural water molecules are displayed by grey-colored spheres. (b)Top view of interface atoms.

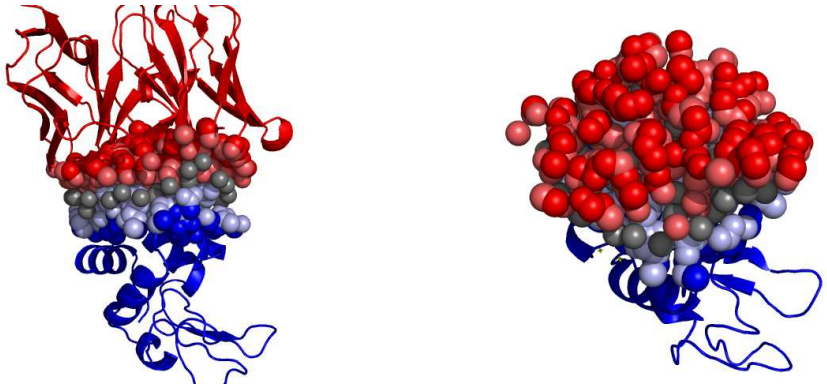
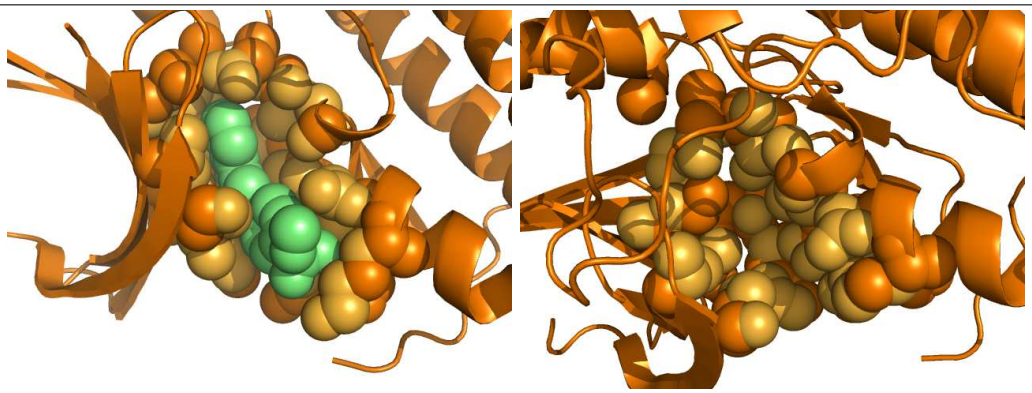


Figure 2 Left: Protein - drug complex —`PDB code 1ke5`: protein chain, drug atoms (in green), and the two layers of interface atoms on the protein (light and dark orange); Right: The two layers in the unbound form of the protein —`PDB code 1PW2`;



1.2 The arrangement of circles on a sphere

Arrangement and related problems. Consider a collection of n circles on 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 S_0 into

faces, circular arcs and vertices. A *singular point* is a point of S_0 crossed by at least two circles; a pair of circles *in contact* is a pair of circles running through a singular point.

One may pose four different problems:

- 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 latter 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, we present a solution for problem P3, and an extension to handle problem P4.

Paper positioning in Computational Geometry. Our construction of the exact arrangement of circles on a sphere is based on an explicit handling of degeneracies and certified predicates. By working on 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.

The following works are directly related. An algorithm to compute the trapezoidal decomposition of a sphere induced by a collection of circles is developed in [HS98]. Degeneracies are removed using an explicit perturbation, while we compute the exact arrangement. Exact sweep-line algorithms have motivated a great deal of work [FHK$^+$06], in particular for line-segment in the plane [MN00], and for conic arcs in the plane [BEHH02]. For sweep-line algorithms on surfaces, a generic algorithm to compute the exact arrangement of curves of surfaces is presented in [BFHW]. With respect to this latter work, our algorithm is dedicated to circles on a sphere, but has three advantages (i) it comes with a complexity analysis (ii) all kinds of circles on a sphere are handled (as opposed to great circles only) (iii) the algorithm comes with an efficient implementation, as exactness incurs a performance penalty of two with respect to calculations performed using doubles on generic examples. To finish up this review, one should mention papers dealing with arrangements of quadrics. The first complete and exact implementation computing the planar map induced by intersection curves of a set of quadrics running on the surface of one of them is described in [BHK$^+$05]. In theory, this algorithm could be adapted to the special case of spheres. In [MTT05], an algorithm to compute exact 3D arrangements of quadrics is developed, but no implementation is provided.

With respect to sweep-line algorithms, we investigate a strategy maintaining a linear size event queue in section 4.3 [1], and present a combinatorial strategy to reverse geometric primitives at intersection points in section 4.2. Implementation-wise, we provide the first solution to compute the exact arrangement of circles of any type on a sphere. In particular, we stress the importance of maintaining a linear size queue, in conjunction with arithmetic filter failures.

Paper positioning in Molecular Modeling. Traditionally, collection of balls have been used in molecular modeling to define molecular surfaces, see [Ric74, Ede92, Con96] for selected pointers. But as mentioned in introduction, our interest stems from the question of modeling atomic environments using multi-body contacts, a central question to design structural alphabets [CGT04], to develop hydrophobic force fields [Hum99] and statistical potentials [SLD05]. To the best of our knowledge, our arrangement based model of interactions is the first one encoding multi-body contacts based on atomic balls.

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 $B_i(c_i, r_i)$ is represented by its center $c_i = (x_i, y_i, z_i)$ 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 w.r.t. sphere S_i is defined by $\pi(p, S_i) = pc_i^2 - r_i^2$. The radical plane $RP_{i,j}$ of any two spheres S_i, S_j is the plane containing the points whose power w.r.t. 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 plane. Sphere S_0 is endowed with cylindrical coordinates (θ, z), and by poles we refer to its North and South poles. We consider the meridian M_θ of S_0 parametrized by $\theta \in [0, 2\pi)$.

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 [CLdCTon], 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 on a sphere rely on the comparison of algebraic numbers of degree at most two.*

The paper is organized as follows. Fundamentals on BO algorithms are recalled in section 2, while terminology and the data structures used are introduced in section 3. The algorithm and the optimization are presented in sections 4 and 5, while its correctness is established in section 6. Topological operations are presented in section 7. The strategy used to report all balls covering a face on a given sphere is discussed in section 8. Implementation and experiments are reported in sections 9 and 10.

[1] Interestingly, this strategy had not been fully investigated in previous work, as testified by the footnote [MN00, page 741]

Finally, appendices 12 and 13 provide complementary pseudo-code.

2 Bentley-Ottmann algorithms

2.1 The planar and spherical settings

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 segment 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 adjacent in $\mathcal{V}$. The algorithm requires predicates to (i) test whether two segments intersect (ii) maintain $\mathcal{E}$ and (iii) maintain $\mathcal{V}$.

Circles on a sphere. For circles on 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 circular arcs. This decomposition allows one to maintain a vertically sorted data structure $\mathcal{V}$ listing the θ-monotonic circular arcs intersected by M_θ during the sweep —such circular arcs are called *active*. Since each θ-monotonic circular 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* and *above* should be understood w.r.t. 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 span 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.

2.2 Circle classification

We shall use the following terminology, illustrated on Fig. 3:

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 $RP_{0,i}$ 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 circular 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, $\mathcal{V}$ contains active arcs. For homogeneity, we consider that a threaded circle defines an arc which is always active. Note that defining θ-monotonic circular arcs for a bipolar circle is useless since such a circle is not transversely intersected by M_θ, so that no insertion into $\mathcal{V}$ is planed —the circle contains the meridian

at special θ values. From now on, these θ-monotonic circular arcs are just called arcs. Considering the two arcs of a normal or polar circle, we call them the *upper* and the *lower* arcs w.r.t. the z axis. For clarity in the sequel, an arc shall never be reduced to a point —a pole.
As a *great circle* on 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 3 The four types of intersection circles.

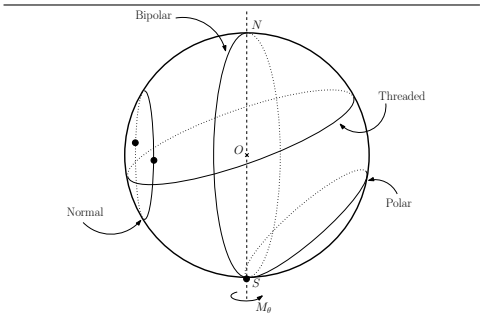
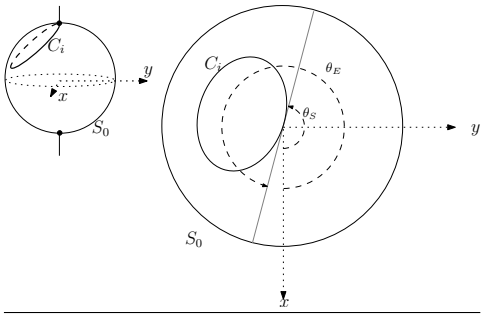


Figure 4 Start (θ_S) and end (θ_E) values of start and end points of a polar circle C_i.



2.3 Circles on 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, $\{S_0; S_i\}$: spheres 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 common intersection of S_i, S_j and S_0 is one point. Equivalently, C_i and C_j are tangent circles on 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 on 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. Cases PC_1 and PC_2 are concerned with the covering of faces of the arrangement by balls —cf problem P4.

Before proceeding, let us just mention how degenerate cases are managed. Case PC_1 is detected from the radii of the intersection circles, which is null in this case [2]. Case PC_2 is detected using the order introduced in section 8.1. Case PC_3, which features a single event, is handled in Algorithm 3. So are cases PC_4, PC_5, for which the ordering introduced in Def. 6 is essential in handling the several (at least two) colliding events.

3 Points, events, and events sites

This section presents the formal definitions of the geometric and algorithmic representations involved in our algorithm. We start by a discussion of the difficulties faced.

3.1 Perspectives and difficulties

Representation. From a geometric standpoint, over the course of the algorithm, we shall focus on *event points*, i.e. points of S_0 where something relevant for the sweep process occurs: two arcs intersect transversely in their interior at that point, or the meridian M_θ is tangent to a circle at that point. But since the algorithm manipulates θ-monotonic arcs, we record the arc(s) involved with a particular event point into an *event*. Additionally, all events having the same event point are gathered into a so-called *event site*.

Ordering. As the BO algorithm requires sorting event sites, one naturally resorts to the lexicographic order over cylindrical coordinates away from the poles. But this order is not sufficient to handle event sites involving polar and bipolar circles. The difficulties stemming from such circles motivate definitions 4 to 8.

3.2 Points vs events

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) [3].

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 corresponding pole, the *polar critical points* are defined 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

[2] 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.

[3] Note that this definition is independent from where the sweep starts.

the end point as for normal circles (see Fig. 4). 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 arcs intersect in their interior, the point is termed *crossing point*. If such arcs are tangent in their interior, the point is termed *tangency point*. If the tangency point coincides with the critical points of the supporting circles, we speak of degenerate tangency. Similarly, if the crossing point coincides with either critical point, we speak of degenerate crossing. See Fig. 5.

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, we define the notion of event:

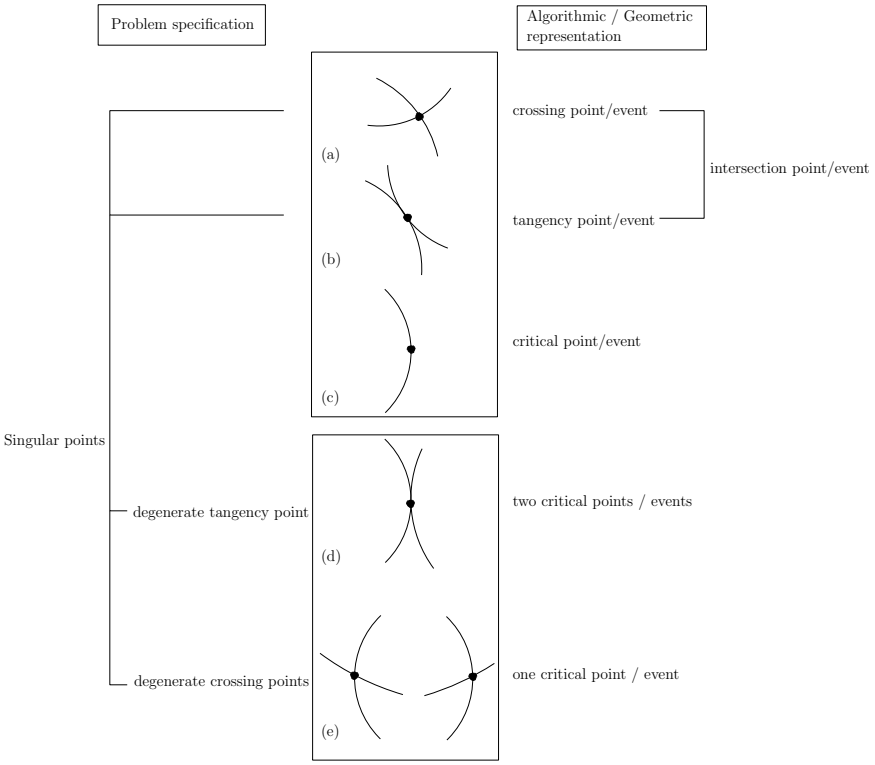
Definition. 2 *The* normal (polar) critical event *associated to a normal (polar) critical point refers to the pair of arcs (the arc) defining the circle, together with a tag* {Start, End} *stating whether the point is a start or an end point. A* bipolar critical event *refers to the relevant part of its circle bounded by the poles, together with a* {Start, End} *tag. An* intersection event *associated to an intersection point refers to the pair of intersecting arcs, together with a tag* {Intersection$_1$, Intersection$_2$, Tangency}*, stating whether the intersection corresponds to the smallest or largest point for a crossing* [4]*, or is a tangency.*

An event which is neither polar nor bipolar is termed normal. *The point of S_0 associated to an event is called the event point.*

Notice that no intersection event is associated to a degenerate crossing or tangency point: a degenerate crossing does not induce a permutation of arcs in $\mathcal{V}$; a degenerate tangency is not specified by a pair of active arcs.

[4] Recall we use the lexicographic order away from poles.

Figure 5 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. The terminology is driven by the decomposition of circles into θ-monotonic arcs. Intersection points are the only singular points triggering a permutation of arcs in the vertical ordering $\mathcal{V}$.



3.3 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 arcs 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 points have the same θ value.*

Normal and (bi)polar event sites. To handle conflicts between normal event points, we define the (normal) event site data structure, which uniquely associates a collection of normal events to a point of S_0. More precisely:

Definition. 4 *A* normal event site *is a collection of normal events with the same event point, partitioned 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 will get back on the representation of intersection points in section 4.2. 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 a normal event site also defines an event point. Using this point, the notion of conflict can be extended among (bi)polar event sites, and between (bi)polar and normal event sites.

Ordering normal event sites. This case is easily handled using the lexicographic order:

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*

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

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 events are removed from $\mathcal{V}$ before the insertion of arcs of start events (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.

Ordering event sites: the general case. Finally we must resolve conflicts between two different types of event site (normal, bipolar, polar). First, as a natural extension of 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 into $\mathcal{E}$:

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

$$e_{P_{ep}} < e_B < e_N < e_{P_{sp}}$$

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

Note that a conflict between two bipolar event sites cannot arise, as we excluded the case of identical circles in section 2.3.

4 Bentley-Ottmann on a sphere

In this section, we recall basics about the classical BO —section 4.1, proceed with the maintenance of the linear size event queue –section 4.2 and 4.3, and the handling of event sites –section 4.4.

4.1 Algorithm and perspective

The algorithm, whose pseudo-code is presented on Algorithm 2 in Appendix 12, 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 $\theta = 2\pi^-$. 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 $\theta = 2\pi^-$.

Of particular interest will be the problem of maintaining a linear size event queue. For BO dealing with line-segments, this strategy may be termed of classical in terms of algorithmic design [BY98]. But the picture is different implementation-wise as testified by the following footnote, from [MN00]:

Our X-structure may contain intersection points between segments that are no longer adjacent in the Y-structure. These events could be removed from the X-structure. Removing these events would guarantee an X-structure of linear size, however, at the cost of complicating the code. Since the size of the X-structure is always bounded by the size of the output graph we do not remove these events.

For BO dealing with curved objects, we are not aware of any report on this strategy. As we shall see in section 10.2, this optimization is actually not just concerned with memory requirements, but rather efficiency concerns related to arithmetic filter failures. To present the strategy, we first introduce the notion of block, which is used to encode the loss of adjacency upon updates of the event queue.

4.2 Blocks within a normal event site and reversal of arcs

As reported in [BY98], maintaining a linear size queue requires keeping in $\mathcal{E}$ events (intersection events in our case, see Def. 2) corresponding to arcs adjacent along $\mathcal{V}$ only. To do so, we need to detect pairs of arcs which are not adjacent anymore along $\mathcal{V}$ when processing an event site from $\mathcal{E}$, so as to update $\mathcal{E}$ accordingly. That is, to maintain the linear size of $\mathcal{E}$, we wish to maintain the following:

Invariant. 1 *Upon processing of an event site, the pair of arcs associated to each intersection event in the list $\mathcal{CT}$ of an event site of $\mathcal{E}$ are adjacent along $\mathcal{V}$.*

This invariant will be maintained by algorithm `break_adjacencies` –see section 4.3, which relies upon the notion of *blocks* partitioning the lists of a normal event site.

Definition. 9 *A* block *is a sorted collection of events such that the arcs of these events match a connected component of arcs adjacent along $\mathcal{V}$. The* top *and* bottom *arcs of a block are defined with respect to the position of the arcs in $\mathcal{V}$. The block is termed a* crossing (tangency) *block if all its events are crossing (tangency) events.*

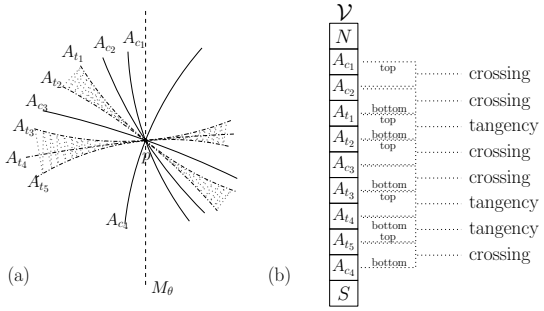
Moreover, the upper and lower bounding *arcs of a block are the arcs of $\mathcal{V}$ located above and below its top and bottom arcs.*

Notice the bounding arcs of a block always exist since the poles are used as sentinels in $\mathcal{V}$. This Def. is illustrated on Fig. 6. 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 reverse arcs along $\mathcal{V}$ and to detect new adjacency between arcs), consists of finding its tangency and crossing blocks. The corresponding algorithm, `find_CT_block_bounds`, consists of two steps. First, thanks to Invariant 1, the block $\mathcal{CT}$ is retrieved by chaining the events, as each arc but the bounding ones appears in exactly two events. Second, from the $\mathcal{CT}$ block, the crossing and tangency blocks are defined by the maximal sequences of events of a given type –recall a crossing and tangency tag is stored in each intersection event, see Def. 2.

Remark 2 *The calculation of blocks is a combinatorial problem, i.e. no predicate is involved, as each intersection event stores a type stating whether the intersection is a crossing or a tangency.*

Remark 3 *We assumed each intersection event effectively stores the two arcs involved. In degenerate cases i.e. when several arcs go through the same event point, this naive implementation duplicates arcs common to two events. But since adjacent arcs are sorted along $\mathcal{V}$, one can reduce an intersection event to the highest arc of the pair, the second one being retrieved, if necessary, from $\mathcal{V}$ –since both arcs are consecutive in $\mathcal{V}$.*

Figure 6 (a) A set of events can define many blocks by considering its partition into subsets of events of same type. 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: the connected components whose arcs are $[A_{t_1} A_{t_2}]$ and $[A_{t_3} A_{t_4} A_{t_5}]$, and three blocks defined by crossing events whose connected components of arcs are $[A_{c_1} A_{c_2} A_{t_1}]$, $[A_{t_2} A_{c_3} A_{t_3}]$ 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) Retrieving the blocks using the arcs stored in events, and the crossing/tangency tag.



4.3 Maintaining a linear size event queue

Equipped with the notion of block, we present algorithm `break_adjacencies` to maintain Invariant 1 and therefore the linear size of $\mathcal{E}$. An intersection event is termed *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 $\mathcal{E}$ valid intersection events only. Assuming by induction that all intersection events stored in normal event site of $\mathcal{E}$ are valid before processing the top event site e, we need to remove from event sites of $\mathcal{E}$ events which get non-valid upon processing e.

Normal event sites. To describe algorithm `break_adjacencies`, we consider and enumerate all possible cases where adjacency is lost, which includes the following two steps.
▷ **1.** The first step, illustrated on Fig. 7, 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). $\boldsymbol{\mathcal{F} \neq \emptyset}$. Function `handle_event_site` removes from $\mathcal{V}$ arcs associated to events of $\mathcal{F}$. 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 processed. Let C^+ and C^- the circle of largest 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). $\boldsymbol{\mathcal{F} = \emptyset}$ **and** $\big[(\boldsymbol{\mathcal{S} \neq \emptyset}$ **and** $\boldsymbol{\mathcal{CT} \neq \emptyset})$ **or** $(\boldsymbol{\mathcal{S} = \emptyset}$ **and** $\boldsymbol{\mathcal{CT} \neq \emptyset})\big]$. 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). $\boldsymbol{\mathcal{F} = \emptyset}$ **and** $\boldsymbol{\mathcal{S} \neq \emptyset}$ **and** $\boldsymbol{\mathcal{CT} = \emptyset}$. 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 (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 $\mathcal{E}$. One can refer to Fig. 6 for an illustration.

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).

(Bi-)polar event sites. The only relevant case is that of a end event site of a polar circle intersected by M_0: we remove from $\mathcal{E}$, if any, the intersection event between the arc corresponding to the polar circle and its neighbor in $\mathcal{V}$ which is not a pole.

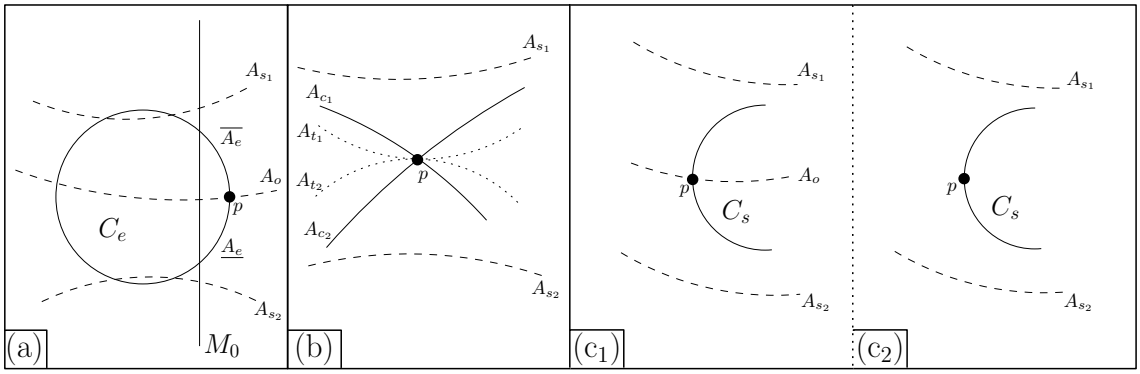
We conclude with the following:

Lemma. 1 *Algorithm* `break_adjacencies` *removes all non valid crossing and tangency events, and thus maintains Invariant 1.*

Remark 4 *During the course of the algorithm, two arcs from two intersecting circles yield one or two events: two if they intersect transversely twice and no event has been processed, and one otherwise –the arcs are either tangent, intersect once, or one of the two crossing*

events has been processed. Upon adjacency loss, to avoid calling the numerical predicates required to locate the event site of the intersection event to be removed, we use a map associating to each pair of arcs adjacent along $\mathcal{V}$ the event site containing the corresponding event(s).

Figure 7 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 the 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 breaks adjacency between pairs (A_{s_1}, A_o) and (A_o, A_{s_2}); (c$_2$) Inserting circle C_s breaks adjacency between of (A_{s_1}, A_{s_2}).

4.4 Handling event sites

Finally, we present algorithms `handle_event_site` and `handle_polar_bipolar_event_site`, to handle normal, as well as polar and bipolar, event sites. These algorithms consist of inserting and removing arcs into and from $\mathcal{V}$, reversing arcs in $\mathcal{V}$, and inserting into $\mathcal{E}$ the new intersections revealed by new adjacencies in $\mathcal{V}$. The pseudo-code of `handle_event_site` is given in Algorithm 3 in Appendix 12 and that of `handle_polar_bipolar_event_site` is given in Algorithm 5 in Appendix 13.

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 so as to bound all the arcs from events of the block;
▷ **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 crossings and tangencies are reversed.

The operations just listed are accompanied by the appropriate intersection tests, so as to update $\mathcal{E}$. We proceed by analyzing cases where two arcs get adjacent along $\mathcal{V}$. We describe intersection tests to perform, each time a configuration matches one of the following cases:
– **If $\mathcal{F} \neq \emptyset$ and $\mathcal{S} \cup \mathcal{CT} = \emptyset$**: If there exists only one arc passing through the end points, this arc becomes adjacent to the two arcs bounding the circles ending, after removing its arcs from $\mathcal{V}$. Else the two arcs bounding the circles ending become adjacent, after removing its arcs from $\mathcal{V}$.
– **If $\mathcal{S} \neq \emptyset$**: After insertion, the upper (lower) arc of the circle of largest radius starting becomes adjacent with the arc above (below) it in $\mathcal{V}$. Having inserting the upper (lower) arc of the circle of smallest radius, it becomes adjacent to the arc below (above) it in $\mathcal{V}$.
– **If $\mathcal{C} \neq \emptyset$ and $\mathcal{S} = \emptyset$**: 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 $\mathcal{CT}$.
– **If $\mathcal{C} \neq \emptyset$**: The top and bottom arcs of each tangency block of $\mathcal{CT}$ after reversal get adjacent to the bounding arcs of that block. Note that if the top (bottom) arc of $\mathcal{CT}$ block is also a top (bottom) arc of one $\mathcal{T}$ block, the intersection test involving this arc is not performed again.

The cases presented cover all possible situations, and no intersection test is performed twice. We conclude with the following:

Lemma. 2 *Algorithm* `handle_event_site` *detects all intersection points corresponding to arcs getting adjacent along $\mathcal{V}$ after handling an event.*

Algorithm `handle_polar_bipolar_event_site.` Consider a polar event site. To handle such a 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 sites do not induce any insertion of arc in $\mathcal{V}$, and thus do not trigger any intersection test. The way singular points on such circles are handled is described in section 7.2.

5 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 reversal operations induced by intersection events.

5.1 Initializing $\mathcal{V}$

To initialize $\mathcal{V}$, we first collect among relevant circles (normal, threaded, polar) those whose intersection with $M_{2\pi^-}$ does not reduce to a pole, that is circles intersected transversely by

M_0, and circles whose end point is located on M_0. The corresponding arcs are sorted [5], that is we find the vertical ordering of the arcs at $\theta = 2\pi^-$. For two arcs going through distinct intersection points with M_0, the ordering is that of the intersection points along M_0; for two arcs going through the same point of M_0, we resort to 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 arcs.

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.

5.2 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 north (south) polar circle comes right after (before) its pole; bipolar circles are processed on the fly at the event sites without any update of $\mathcal{V}$.

As explained in Algorithm 3, 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 among arcs in $\mathcal{V}$. (If $\mathcal{F} \neq \emptyset$, this operation is superfluous.) Then, using the sortedness of $\mathcal{S}$, arcs of circles with smaller radii are consecutively inserted closed to arcs of the last circle processed.
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 planes (RP) of all but bipolar circles drawn on S_0. Denoting c_{0i} the center of circle C_i, we orient the normal vector n of a RP as follows: for a normal, the orientation of n is that of c_0c_{0i}; for a threaded or polar circle, the orientation is such that n makes an acute angle with z-axis —the z coordinate of n is positive. Using this orientation, a point is said to be above (below) the RP if it is in the half-space pointed by (opposite to) the normal n. As illustrated on Figs. 8 and 9:

Observation. 1 *The position of a start point p w.r.t. to an arc whose circle is C_i can be computed as follows:*
– 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 plane $RP_{0,i}$, 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 coordinates of p and C_i start point.
– If C_i is a threaded or polar circle, the position of p w.r.t. the associated arc is characterized by its position w.r.t. the oriented radical plane $RP_{0,i}$.

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.

[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.

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 8 Position of start point p w.r.t. normal circles: the position of p w.r.t. circular arcs reads from the position w.r.t. oriented radical planes. See text.

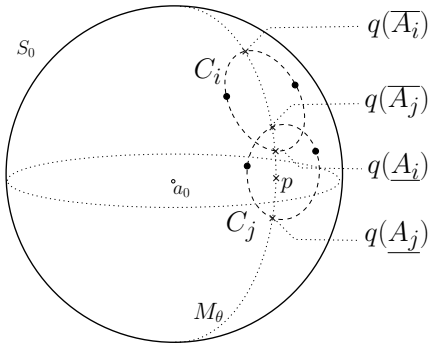
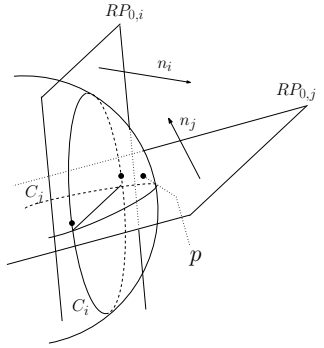


Figure 9 The position of start point p w.r.t. a threaded circle also read from oriented radical plane. See text.



5.3 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, w.r.t. blocks defined by tangency events. We do so in two steps, by first reversing the arcs involved in the block defined by $\mathcal{CT}$, and second by reversing again the arcs involved in tangency blocks —to see how blocks are defined refer to section 4.2. Since we use a tree for $\mathcal{V}$, reversing a set of consecutive arcs can be done by re-writing the values of the nodes holding the arcs involved —which avoids any insertion or deletion in the tree.

6 Correctness and complexity analysis

To analyze the algorithm complexity, 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 n the number of circles, and k (v) the number of singular points (vertices), we have $v = O(n + k)$. An edge is a circular 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. We formally define holes with respect to faces as follows:

Definition. 10 *Consider a simply connected region R consisting of the union of one or several faces of the arrangement. If the 1-skeleton of R is not connected, the* principal 1-skeleton *is the connected component containing the boundary of R.*

Consider a non-simply connected face f bounded by several cycles, and let c_0 be one of these cycles. We define the support *R of face f with respect to c_0 as the simply connected region of S_0 bounded by cycle c_0. Moreover, let $dom(f)$ stand for the geometric domain defined by face f. The connected components of $R \backslash dom(f)$ are* holes *in the support of face f.*

These notions are illustrated on Fig. 10. Notice a hole is a simply connected region consisting of the union of a collection of faces.

Lemma. 3 *Denote v and e the number of vertices and edges of an arrangement of n circles on 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 first introduce a hierarchical representation of the arrangement, based on the concepts of Def. 10. Given an arbitrary edge of the arrangement, let K_0 be the connected component of the 1-skeleton of the arrangement containing this edge. The graph K_0 defines a decomposition D_0 of the sphere into topological disks –simply connected faces of the arrangement or their supports induced by K_0. For each such disk, we can construct a tree 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 of the arrangement. See Fig. 10 for an illustration.

The proof consists of applying Euler's relationship to simply connected regions (s.c.r. for the sake of conciseness in this proof), namely disks of D_0, and holes recursively found in traveling down the tree decomposing each such disk.

Let V_s, E_s, F_s the numbers of vertices, edges and s.c.r. of the decomposition D_0. One has $F_s = F_s^{(3)} + L_s$, with $F_s^{(3)}$ (L_s) the number of s.c.r. bounded by at least (exactly two) three edges. As an edge bounds two s.c.r., 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

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

Consider now a s.c.r. of D_0 featuring holes. We apply Euler's relationship to the decomposition of each such hole by its principal 1-skeleton. To do so, some care is in order as the s.c.r. corresponding to the support of the face containing the hole has been accounted for. For a given hole whose number of vertices, edges and s.c.r. are also denoted V_i, E_i, F_i, let T_i be the number of edges bounding the hole. Counting edges over faces of the decomposition of the hole yields $2E_i - T_i \geq 3F_i^{(3)} + 2L_i$. 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 analysis just carried out for a hole is valid at every level of the tree decomposing one s.c.r. of D_0. Moreover, 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 can finally 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 $O(n^{3/2+e})$, for any $e > 0$, where the constant of proportionality depends on e [ALPS01].

Theorem. 2 *Algorithm 2 correctly reports all the singular points of a family of n circles on 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. 5, we establish that Algorithm 2 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 $\mathcal{V}$ (lemma 2). Second, a degenerate tangency is detected upon insertion of a start event or an end event into an event site of $\mathcal{E}$, 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 circulars arcs are inserted into $\mathcal{V}$, as the start point is found to be on a circular arc. For the latter, 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 analyze 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 an insertion indeed requires looking for an event site in $\mathcal{E}$ —and creating one if necessary. But for a normal critical event, the insertion into the sorted list $\mathcal{S}$ or $\mathcal{F}$, whose size is $O(n)$, has complexity $O(\log n)$.

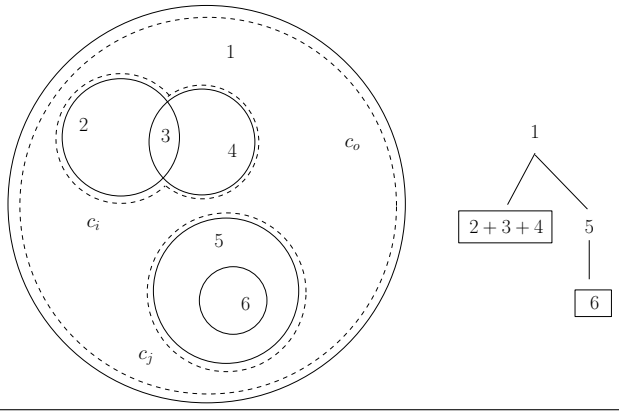
To begin with, the algorithm classifies the circles, which incurs a linear cost. As there are $O(n)$ normal circles, inserting these critical events into $\mathcal{E}$ requires $O(n \log n)$. Similarly, inserting at most $2n$ arcs into $\mathcal{V}$ costs $O(n \log n)$.

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 3, 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 10 Six faces decomposing the disk bounded by c_0. The cycles bounding face 1 are c_o, c_i, c_j. The support of face 1 with respect to cycle c_0 is the disk itself, and this support contains two holes bounded by cycles c_i and c_j. Cycle c_i bounds a hole with 3 faces, while c_j bounds face 5 which contains a hole.



7 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 into a half-edge data structure (HDS).

7.1 Describing the arrangement

Arcs and half-edges. The vertices of the HDS correspond to event points, its edges are circular 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.

For all but bipolar circles, to each arc A_i, we associate two active half-edges qualified w.r.t. 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. We now address particular cases: (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 [6] (ii)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.

In addition to the notions of upper and lower half-edges, we shall need the following qualifiers. 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, and *outer* otherwise; if the circle is threaded, an half-edge is *inner* if it induces the spherical cap containing the north pole, and *outer* otherwise. With these conventions, the spherical caps of smallest area induced by a great circle is set to be the one induced by inner half-edges.

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

[6] Any half-edge with two null vertices is associated to a threaded circle void of intersection with other circles.

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 on the sphere S_0. The set of all CCB describing a face can be split into two sets: one CCB called *principal*, defines the spherical cap containing the face; the remaining ones define *holes* in the face. See Fig. 12 [7].

Figure 11 Completing a half-edge –seeking 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 $\overline{A_i}$. The first point of the active half-edges displayed is represented by a large black.

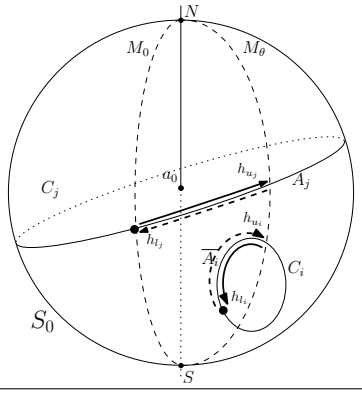
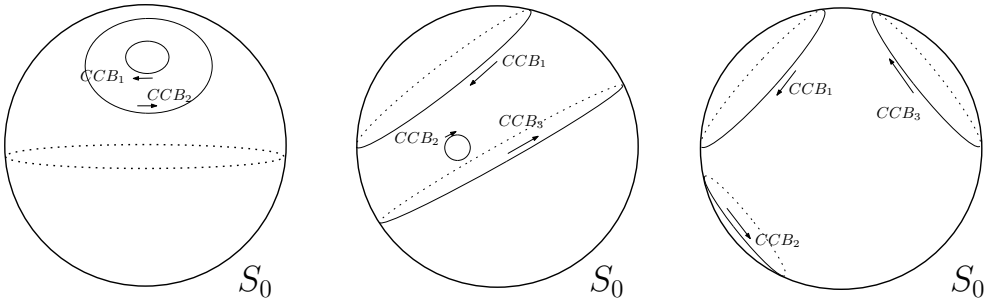


Figure 12 Definition of a face with a set of CCB. Each arrow represents a CCB. In the four cases, only one face is defined by the CCB represented.



[7] 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.

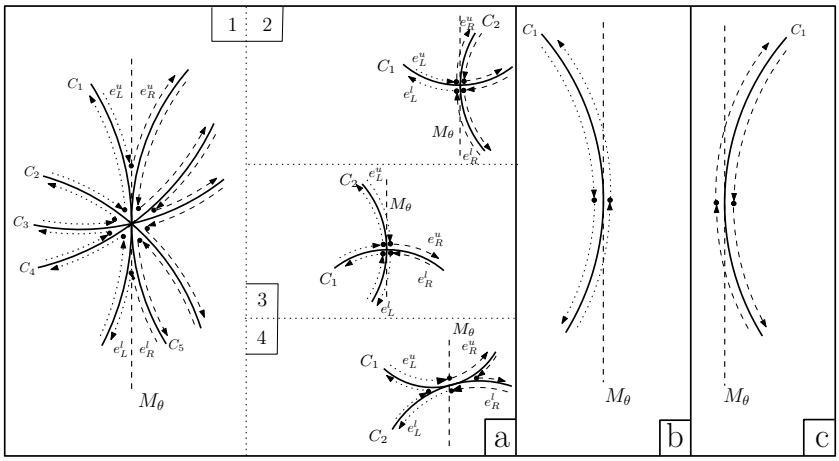
7.2 Handling half-edges

Initialization and termination. As explained in section 5.1, 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 $M_{2\pi^-}$, 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 $M_{2\pi^-}$ 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 2.

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 events from the list $\mathcal{CT}$ go through p, their active half-edges are stopped at p, and new half-edges are created with p 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. The merge operations are performed as follows.

To reflect the position of half-edges associated to arcs passing through p w.r.t. 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 3, 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 $\mathcal{V}$ enclosed in-between the bounding arcs —if any.
▷ **2.** Function `topo_handle_above_below`. Considering the block defined by events of an event site, to the left of p, let e^u_L, 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^u_R,e^l_R) similarly. We face three cases: (i) theses two pairs are defined, we merge e^u_L with e^u_R and e^l_L with e^l_R —see Fig. 13(a). (ii) e^l_R and e^u_R are not defined, we merge e^l_L with e^u_L —see Fig. 13(b). (iii) e^l_L and e^u_L are not defined, we merge e^l_R with e^u_R —see Fig. 13(c).

Figure 13 Operations on half-edges at a normal event. Dashed half-edges get created, while dotted ones get terminated. A black dot corresponds to a merge of two half-edges. (a) different cases with merge to the left and to the right of 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 list $\mathcal{T}$ is not empty, in order to have the opposite half-edge relation ; (b) merge at end event; (c) merge at start event;

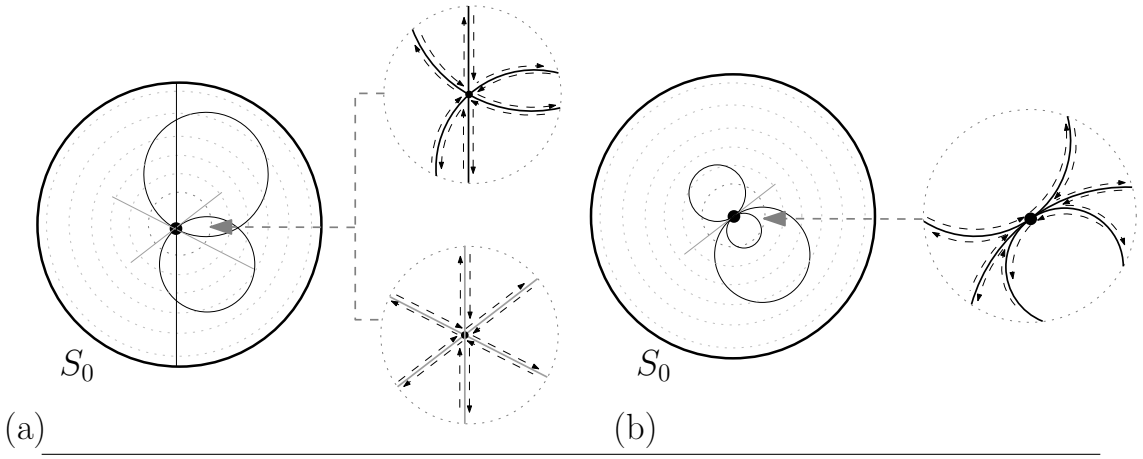


Half-edges for a (bi)polar event at a pole. To handle half-edges at poles, as illustrated on Fig. 14, when turning around poles according to M_θ, we merge consecutively encountered half-edges created at (bi)polar event sites. At a such start (end) event, we first handle the outer (inner) half-edge and then the inner (outer) half-edge. The algorithm handles cases when circles are intersected at $\theta = 2\pi^-$ and when there are tangencies between polar or bipolar circles. The algorithm is straightforward yet tedious, and the reader is referred to section 13 for the details. Note that this handling is designed for the ordering of (bi)polar event sites in conflicts according to circle radii, 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. 15 (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. 15 (b) and Algorithm 5 for the details.

Figure 14 Operations on half-edges at polar event. (a)Transverse intersection between two polar and one bipolar circles (b)Tangency between three polar circles.



7.3 Building the faces

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

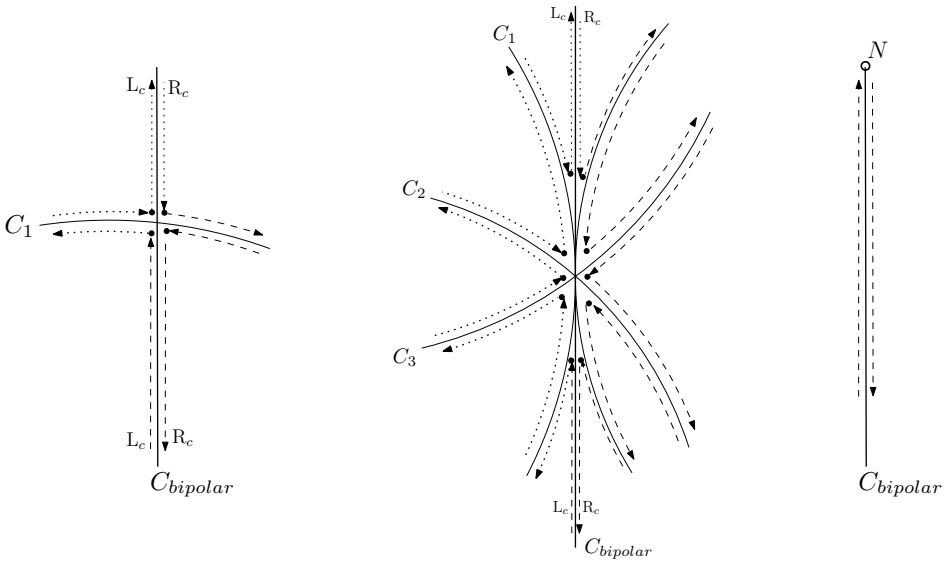
7.3.1 Creating a CCB

A SCH becomes a CCB whenever the merge step of two half-edges creates a topological circle. As the intersection between a CCB and the meridian may feature several connected components —see Fig. 16, we merge SCH using union-find, which requires endowing each half-edge with a pointer to a master half-edge —called the *CCB master pointer* or *CCB master* for short. A merge of half-edges makes their CCB masters become the same, and a CCB appears when the two half-edges being merged already have the same CCB master.

7.3.2 Creating a face

Running Union-Find. A face consists of a principal CCB and of CCB defining holes. We construct faces using union-find on SCH, which requires a union-find pointer called the *face master pointer*, or *face master* for short. While closing a SCH, the resulting CCB is principal if it is its own face master; if not, the CCB becomes a hole of the face the face master refers to. 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. 16 for an illustration. (Notice a merge step is also performed upon completion

Figure 15 Operations on half-edges at bipolar event. Dotted half-edges get terminated, while dashed ones get created. A black dot represents a merge (a) Arc crossed by a bipolar circle (b) Conflict between a normal and a polar events (c) Two half-edges created at a bipolar critical event.



of the sweep at $\theta = 2\pi^-$.) The only way for two components to become adjacent is to be merged at an end point of a normal circle. Therefore, at each event site featuring only end events —no degenerate crossing, we test if the outer half-edges contribute to the same face, and if not make the union of their face masters [8].

Initializing the master pointer. Having explained the union-find process, we complete the description by the initialization of the face master. To set this pointer for a newly created SCH, we face two 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: $s_{ch} = \{e_1\}$ either for a (bi)polar circle critical event, or during the initialization of $\mathcal{V}$ at $\theta = 2\pi^-$; $s_{ch} = \{e_1, e_2\}$ either for a normal event site —in which case we assume the arc of e_1 is above that of e_2 along $\mathcal{V}$, or for a bipolar cut [9] —in which case we assume e_2 is on the bipolar circle.

Notice the meridian M_θ goes through the first point of this (these) half-edge(s).

For a fixed meridian M_θ, we denote $F_N(M_\theta)$ the face containing the north pole. If the arrangement only features normal, threaded and south polar circles, the north pole belongs to the interior of a face, which is precisely $F_N(M_\theta)$. If the arrangement features north polar and/or bipolar circles, the face $F_N(M_\theta)$ evolves over the course of the sweep. If the meridian M_θ is not tangent to any circle at the pole, $F_N(M_\theta)$ is defined as the face having the north pole on its boundary and containing an infinitesimal portion of M_θ anchored at the north pole. If meridian M_θ is tangent to a (bi)polar circle, $F_N(M_\theta)$ is undefined. But we denote $F_N(M_{\theta^-})$ ($F_N(M_{\theta^+})$) the face containing the north pole for an infinitesimally smaller (larger) value of θ.

In describing the face master assignment, we use the following conventions: when a given SCH starts a face, a new face master is created; when a SCH contributes to an already existing face (under construction), its face master is set to that of this face. The face master assignment works as follows:

▷ **1.** If e_1 is not associated to a bipolar circle:

– **If $\mathbf{e_1}$ is the upper half-edge of its arc.** Let A_i be the arc above the arc of e_1 in $\mathcal{V}$. s_{ch} contributes to the face described by the lower half-edge associated to A_i. A comment is in order if A_i is the north pole. Let F be $F_N(M_{\theta^+})$ if the first vertex of e_1 is the north pole [10], and $F_N(M_\theta)$ otherwise. If the face F is already started, s_{ch} contributes to F; if not, s_{ch} starts F. See Table 1 for a description of events creating/setting the face containing the north pole.

– **If $\mathbf{e_1}$ is the lower half-edge of its arc.** If the event processed is a north polar start event, s_{ch} starts or contributes to $F_N(M_{\theta^-})$. If not, s_{ch} starts a face.

[8] If the event point is shared by several normal circles ending, the check is concerned with the outer half-edges of the circle of largest radius —outer half-edges of a normal circle are indeed concerned with the *outer part* of the circle on the sphere.

[9] See the half-edge R_c in Fig. 15(a,b).

[10] e_1 is the inner half-edge of a north polar circle starting.

▷ **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. See Fig. 15(c). As can be seen in Table 1, the associated SCH starts or contributes to a new face containing the north pole —either $F_N(M_{\theta^+})$ or $F_N(M_{\theta^-})$.

7.3.3 Relationship between the union-find processes for CCB and faces

For the sake of clarity, we presented the above two union-find processes independently. But merging two SCH triggers operations on both pointers at an end event or at $\theta = 2\pi^-$, so as to connect the two components of a face that got created independently. The pseudo-code is presented on Algorithm 1, the functions `Union` and `Find` being distinguished by their subscript to indicate which master is handled. An example is provided on Fig. 16, when merging SCH_1 SCH_2 at *ep*.

Algorithm 1 Merging two SCH

```
1: if (Find_CCB(s_ch1) ≠ Find_CCB(s_ch2)) then
2:    Union_CCB( Find_CCB(s_ch1),Find_CCB(s_ch2))
3:    if (Find_face(s_ch1) ≠ Find_face(s_ch2)) then
4:       Union_face( Find_face(s_ch1),Find_face(s_ch2))
5:    end if
6: end if
```

Figure 16 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 —$SCH_1 + SCH_3$ and $SCH_2 + SCH_4$. After *ep*, one face with three SCH remains. These merges are achieved using union-find for the CCB masters and using union-find of the face masters.

Table 1 Updates of $F_N(M_\theta)$, $F_N(M_\theta^-)$, $F_N(M_\theta^+)$

Position of M_θ	Operation(s) at north pole.
In `topo_init_arrangement`	Upper half-edge of top arc of $\mathcal{V}$ at $\theta = 2\pi^-$ creates and sets $F_N(M_\theta)$
At bipolar and north polar start event	Inner half-edge creates and sets $F_N(M_{\theta^+})$ Outer half-edge creates $F_N(M_{\theta^-})$ if it is not already done
At north polar end event	Outer half-edge sets $F_N(M_{\theta^+})$ Inner half-edge describes the already defined face $F_N(M_\theta^-)$
At bipolar end event	Outer half-edge creates and sets $F_N(M_{\theta^+})$ Inner half-edge describes the already defined face $F_N(M_\theta^-)$
At any other normal start event	Outer half-edge creates $F_N(M_\theta)$ if it is not already done

7.4 Complexity analysis

To conclude, we analyze 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]. In bounding the complexity of the topological operations, we shall use the following observation: if $M \leq U$, then $\alpha(M,N) \leq \alpha(U,U)$. To check this, recall that

$$\alpha(M,N) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor > \log n\}.$$

The inequality follows from two facts: (i) Ackermann's function A is strictly increasing with each of its two arguments, so that $A(i,1) \leq A(i, \lfloor \frac{M}{N} \rfloor)$ (ii) $\log N \leq \log U$.

Upon completion of the sweep, the union-find data structure features f connected components corresponding to the f faces of the arrangement, and h holes. Let n_0 be the number of arcs found in $\mathcal{V}$ at $\theta = 2\pi^-$, and n the number of circles.

Theorem. 3 *Constructing CCB has complexity* $O((e+n_0)\alpha(6e+8n_0, 6e+8n_0))$.

Proof. The e edges of the arrangement yield $2e$ half-edges, while the n_0 arcs intersected at initialization yield $2n_0$ half-edges. The number of half-edges manipulated is thus $2e + 2n_0$.

To count the number of operations, we proceed as follows: we count operations required by the initialization at $\theta = 2\pi^-$; next, we enumerate operations on a vertex (of the arrangement) basis; finally, we count operations involved at the merging step at $\theta = 2\pi^-$.

First, the initialization step requires $2n_0$ *make_set* operations. Second, at a vertex of the arrangement, whenever two half-edges are merged to bound a face, three situations arises, as 2, 1 or 0 half-edges get created. The first case requires two *make_set* and one *union*. The second case requires one *make_set*, one *find*, and at most one *union*. The second case requires two *find* and at most one *union*. In any case, we have at most three operations. A vertex of the arrangement adjacent to m_p edges requires at most $3m_p$ operations, whence a

total number of operations bounded by $6e$. Third, the merge steps at $\theta = 2\pi^-$ requires at most 3 operations for a pair of half-edges, whence $6n_0$ operations at most. Adding up these contributions yields an upper bound of $6e + 8n_0$. □

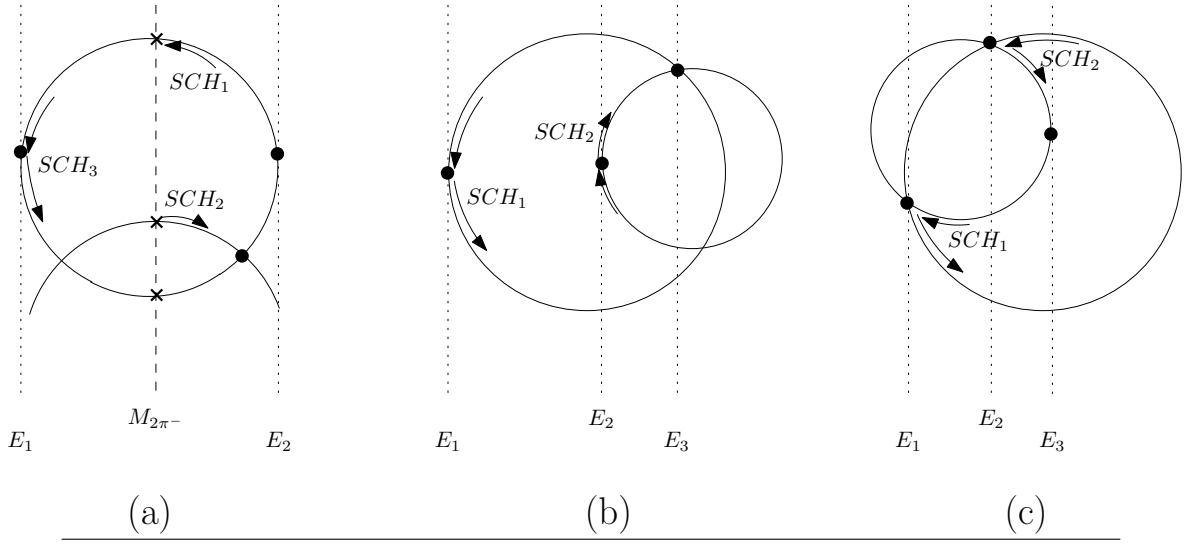
Theorem. 4 *Grouping CCB into faces has complexity* $O((f+h)\alpha(f+h+20n, f+h+20n))$.

Proof. Objects manipulated in this union-find algorithm are SCH. The total number of SCH created is bounded by $f+h+2n_0+n+n$. Let us examine the three overheads w.r.t. $f+h$: the first one, $2n_0$, corresponds to the SCH created upon initialization and merged upon completion of the sweep —Fig. 17(a); the second one, n, corresponds to the SCH started by the outer half-edges of the circle of largest radius involved in a normal event site featuring only start events —Fig. 17(b); the third one, n, corresponds to the SCH that get merged at the end point of the circle of largest radius involved in a normal event site featuring only end events —Fig. 17(c).

Apart from the N *make_set* operations, *union* and *find* operations are performed in three cases:
– first, when merging at $\theta = 2\pi^-$ —Fig. 17(a). Since adjacent half-edges are always associated to the same face master, we have to do at most two *find* and one *union* operations per face defined, i.e. at most $3(n_0+1)$ operations.
– second, at a start point —Fig. 17(b). The SCH corresponding to the outer half-edge of the circle of largest radius is attached to a face in progress, which requires one *find*, and one *union*. Two extra find operations may be required in case the SCH created at the start point of a normal circle is merged to another SCH.
– third, at an end point of some normal circles —Fig. 17(c). Merging the different faces of two different SCH requires two *find* and one *union*.

Adding up these contributions results in $M \leq N + 3(n_0+1) + 7n$, which we rewrite as $M \leq f+h+20n$. □

Figure 17 Counting superfluous SCH. (a) SCH_1 and SCH_2 are created at initialization and merged at E_2; SCH_3 is created at E_1; a single CCB remains at $\theta = 2\pi^-$ (b, c) SCH_1 (SCH_2) is created at E_1 (E_2); both are merged at E_3



8 Reporting inclusion into balls

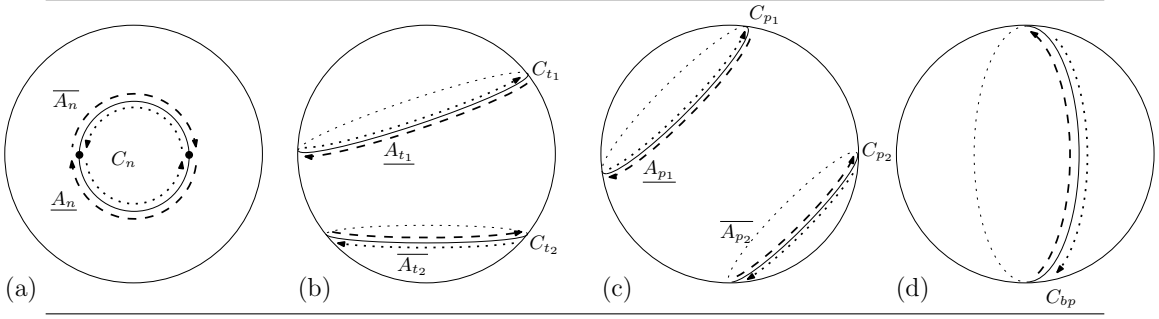
Assume the n intersection circles are generated by m balls, that is, for each intersection circle $C_i = S_0 \cap S_i$, there exists a ball B_i bounded by sphere S_i. In this section, we describe an algorithm reporting the balls covering each face of the arrangement on S_0. Notice that (i) if a sphere 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).

8.1 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. While sorting the circles, each unique intersection circle is endowed with two special balls: the *primary* ball, which is associated to the sphere which first defines such a circle; and the *opposite* ball, which is the first to intersect S_0 along the same circle, but is opposite to the primary —w.r.t. to the plane of the circle. Notice the opposite ball may not exist. Each primary and opposite ball is attached a list of balls yielding the same intersection circle and covering the same part of S_0. We call these the lists of *friends* in the sequel.

Two non-great circles are identical iff they have the same center. Two great circles are identical iff the center of spheres defining the circles are aligned with the center of S_0 —the spheres generating these circles belong to the same pencil. To distinguish great circles, we use the lexicographic order on a canonical vector describing the pencil of spheres yielding a given circle. One candidate vector is the normalized vector of the plane containing the circle. To avoid normalization, we use the following slope vector, whose x (y, z) coordinate is the projection onto the plane $x = 0$ ($y = 0$,$z = 0$) of the slope of the line supporting the normal vector of the plane. Note that the coordinates belongs to the closure of real numbers —i.e. $\mathbb{R} \cup \{\infty\}$. Determining the unique circles and setting the lists of friends has complexity $O(m \log n)$.

Figure 18 Circle arcs, upper/lower and inner/outer half-edges. Dashed half-edges are outer half-edges, while dotted one are inner. (a) normal circle C_n and its upper and lower arcs —$\overline{A_n}$ and $\underline{A_n}$; (b) C_{t_1} is a north threaded circle defining a lower arc, while C_{t_2} is a south threaded circle defining an upper arc; (c) C_{p_1} is a north polar circle defining a lower arc, while C_{p_2} is a south polar circle defining an upper arc; (d) C_{bp} is a bipolar circle inducing half-edges (upper and lower arcs irrelevant).



8.2 Inclusion into balls

8.2.1 Algorithm

Outline. While running the sweep, we construct an implicit encoding of the lists of balls covering the faces of the arrangement —each such list being called a *covering list*. More precisely, the covering lists are represented by a tree, the *covering tree*. Each node in this tree corresponds to one face of the arrangement, and the edge connecting two nodes corresponds to an arc found in $\mathcal{V}$, stating which primary/opposite ball is added to/removed from the covering list of the father node. Notice that working with the primary and opposite balls is sufficient, as these balls are associated lists of friends.

Following the notations of section 7.3, let LNB be the list of balls covering the face $F_N(M_\theta)$. This list is initialized at $\theta = 2\pi^-$, and evolves over the course of the sweep, as indicated on Table 2. The root of the covering tree is precisely LNB. Since a face consists

of one or several CCB and a CCB is incrementally built from SCH, the principle used to set the covering lists consists of setting one such list for each face master upon creation of the corresponding SCH. (For a face featuring several CCB, the only one endowed with a covering list is the face master.)

This strategy subsumes two things: first, upon extension at a crossing point, even if another half-edge contributes to the SCH identified as the face master, the covering list does not change as the contribution (covering ball) of the corresponding circle has already been taken into account in a father node; second, whenever two SCH both identified as face masters are merged —two faces get merged, one of the two covering lists can be discarded as the information is redundant. In the covering tree, the sons of the node suppressed are attached to the node that remains —this node corresponds to the face master that stays after the union-find.

A technical observation. To report balls covering faces of the arrangement, we first define the notions of upper and lower arcs for threaded circles —section 2.2.

A threaded circle is called *north threaded* if its center has a z coordinate larger than or equal to that of the center of S_0, and *south* threaded otherwise. 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 relationship between circles arcs, upper/lower and inner/outer half-edges is illustrated on Fig. 18, and we have:

Observation. 2 *The spherical cap of smallest (largest) 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 of a all but bipolar circle is always inner/outer (outer/inner).*

Using Observation 2, 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, and that covers the smallest (largest) part of S_0 bounded by C, also covers F. Moreover, using the upper or lower status of an arc A in $\mathcal{V}$, one can locate which side —smallest or largest part, of S_0 w.r.t. A is described by its current inner/outer half-edges.

Updating the covering lists. To describe the addition or removal of balls upon creating a new SCH, we elaborate upon the strategy already described for the face creation —refer to paragraph *Creating a face* in section 7.3 for the notations:
▷ **1.** Half-edge e_1 is not associated to a bipolar circle. Denoting A the arc associated to e_1, we consider the following 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 a face containing the north pole. If this face has not already been started, the collection of relevant balls are those in LNB.

– **e_1 is the lower half-edge of arc A.** If the event processed is a north polar start event, as the SCH starts or contributes to $F_N(M_{\theta^-})$, the covering list either exists or is LNB before the update mentioned in Table 2.

If not, let L_B, be the collection of balls associated to the face of the face master 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 the primary/opposite balls of the circle of A, the covering list of the new face is obtained by (i) adding to (removing from) L_B the primary/opposite ball covering the smallest part of S_0 bounded by the circle of A and (ii) removing (adding to) from L_B the primary/opposite ball covering the largest part. Note that the ball covering the largest part removed —if any, has been inserted while initializing LNB (see Table 2).

▷ **2.** If half-edge e_1 is associated to a bipolar circle, e_1 is created at a bipolar critical event and is anchored at the north pole. Half-edge e_1 either starts or contributes to $F_N(M_{\theta^+})$ or $F_N(M_{\theta^-})$. At a bipolar start event (bipolar end event), the associated covering list 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). See Fig. 15.

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. 19. Consider the case of face 6. This face got started at the start event of circle C_4 using its two inner half-edges. The highest one of the pair is the lower half-edge of the upper arc of C_4. The upper half-edge of that arc describes face 4. Therefore, the node of face 6 is a son of the node of face 4, and since the arc considered is an upper one, the edge between the two nodes indicates that the ball covering the smallest (largest) part of S_0 according to C_4 must be added (removed).

Remark. 1 *As reported on Table 2, the maintenance of LNB consists of three steps. Step 1b handles all spheres in the same way, which imposes corrections in two cases. First, if a north polar circle is intersected by M_0, its ball covering the smallest (largest) part of S_0 should be added (removed) —case 2a. The same holds for a north threaded circle —which is always intersected by the meridian, as seen for case 2b.*

Notice also that during the sweep, as North polar and bipolar events modify the face $F_N(M_{\theta^+})$, one of the two associated balls must be added and the other removed.

Figure 19 Implicit encoding of covering lists. Circle C_1 is north threaded, C_2 bipolar, while C_3, C_4 and C_5 are normal circles. 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.

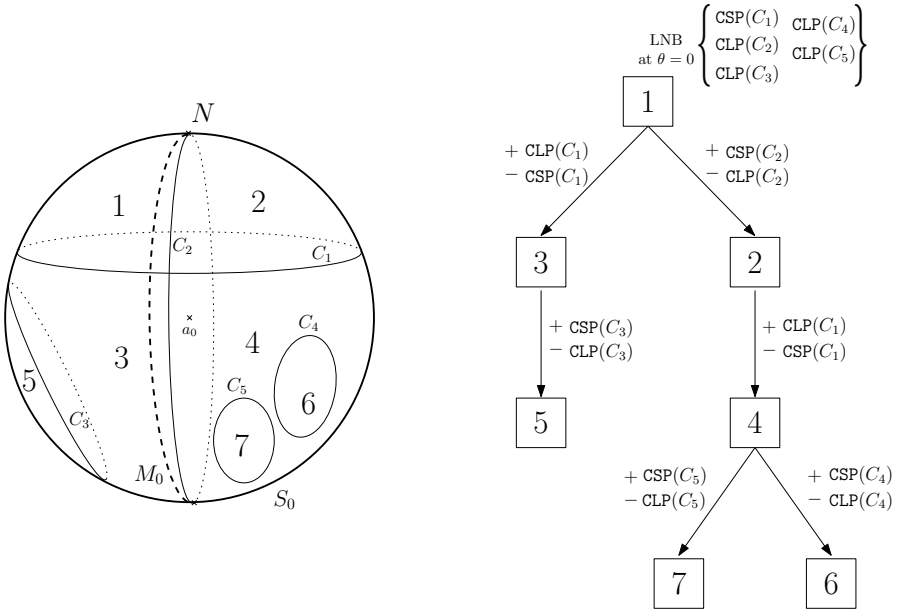


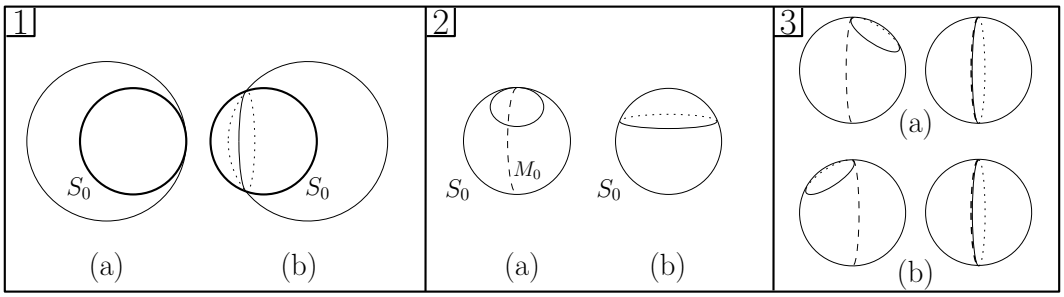
Table 2 Updating balls in LNB. The ball/sphere/circle processed is denoted $B/S/C$. Function CLP(C) (CSP(C)) returns, if any, the balls Covering the Largest (Smallest) Part of S_0 bounded by C. Operator + (-) means the ball is added to (removed from) LNB. See Fig. 20.

Steps		type of event (exclusive cases)	Actions on LNB
Filtering spheres	1a	$S \cap S_0 =$ a point and $S_0 \subset B$	+ B
	1b	B covers the largest part of S_0	+ B
Classifying circles	2a	North polar circle with $\theta_E < \theta_S$	+ CSP(C) and - CLP(C)
	2b	North threaded circle	+ CSP(C) and - CLP(C)
Handling events	3a	North polar or bipolar circle Start event	+ CSP(C) and - CLP(C)
	3b	North polar or bipolar circle End event	+ CLP(C) and - CSP(C)

8.2.2 Complexity analysis

To analyze the complexity of the implicit encoding, n_0 denotes the number of arcs intersected by M_0, m the number of input balls.

Figure 20 Updating the balls of LNB —see Table 2. 1(a) S_0 is covered by a ball whose sphere is tangent to S_0 1(b) A sphere intersects S_0, and the associated ball covers the largest part of S_0 2(a) A north threaded circle. The north pole is always (never) covered by a ball covering the smallest (largest) part of S_0 2(b) Meridian M_0 intersects a polar circle. The ball covering the smallest part of S_0 covers $F_N(M_0)$ 3(a) The dashed meridian M_θ stopped at a (bi)polar start event: $F_N(M_{\theta^+})/F_N(M_{\theta^-})$ is covered by a ball covering the smallest/largest part of S_0 3(b) The dashed meridian M_θ stopped at a (bipolar) end event: $F_N(M_{\theta^+})/F_N(M_{\theta^-})$ is not covered by a ball covering the smallest (largest) part of S_0



Theorem. 5 *Computing the covering tree has complexity $O(f+m)$ and $O(f+m)$ space. Denote $s_T(F)$ total number of balls covering face F. Constructing the covering lists for all faces of the arrangement requires $O(\sum_{F\in faces} s_T(F)+f)$ time.*

Proof. First consider the initialization. Initializing LNB requires $O(m)$ time and space, and $O(n)$ lists are created considering arcs intersected at $\theta = 2\pi^-$. Next, consider the sweep. A constant time and space update is performed on the tree every time a face master is created. Moreover, one node is deleted every time a merge of two faces is performed. But since at most $O(n)$ merges are performed —the number of normal end events together with the number of faces intersected at initialization, see Fig. 17(a,c)—, the total number of nodes created is $O(f+n)$. Finally, upon completion of the sweep, $O(n)$ merges occur to complete the faces started during the initialization. Adding up these costs and space requirements yields the conclusion.

To compute one covering list per face, we first deal with primary/opposite balls. From a depth-first traversal of the covering tree, the lists of primary/opposite balls are constructed for each node. Traversing an edge of the tree consists of copying the list of the father node, from (to) which one ball may be removed (added). Next, the remaining balls are recovered from the lists of friends. □

9 Implementation

Code architecture. The code is written in CGAL style —see `www.cgal.org`. As it is used for applications in molecular modeling —see below, it features three main classes. Given a collection of balls, the first one provides a method reporting the balls intersecting a given one. 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 centers are in the same cube or in an adjacent one can intersect. Given a specific ball and a list of balls intersecting it, the second class computes the arrangement of intersection circles, and reports the balls covering each face of the arrangement. This class is templated by a geometric kernel (providing predicates, number type,...), 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. The geometric kernel just alluded to must be a model of the $3D$ Spherical Kernel developed in the companion paper —an outline of the main predicates needed is given in Table 3. This kernel is itself templated by a linear kernel and an algebraic kernel. The linear kernel provides the number type and usual arithmetic/comparison operations. The algebraic kernel defines an algebraic number of degree two type, together with a polynomial type. It also provides a method to evaluate the sign of a polynomial at a such algebraic number, as well as a method to compare these algebraic numbers. Using several template parameters yields several variants. We define the following three variants, based on three different linear kernels.

For the *exact* variant, `CGAL::Exact_predicates_exact_constructions_kernel` is used as linear kernel. This latter type provides a filtering strategy resorting to the `Lazy_Exact_NT<Gmpq>` number type, which is a filtered version of `Gmpq` using intervals —exact computations use `Gmpq` when intervals are not sufficient to conclude.

For the *double* variant, `CGAL::Exact_predicates_inexact_constructions_kernel` is used as linear kernel. This variant comes with no guarantee as the associated number type is a plain `double` number type. The main interest of such a kernel is to assess the overhead imposed by the certification of the results.

Finally, for the *exact filtered* variant, we use two linear kernels: the one of the exact variant, and `CGAL::Cartesian<CGAL::Interval_nt>`. The underlying number type of the latter is an interval number type, and we use it to implement the following strategy: if answering predicates is not possible using intervals, the sweep is restarted using the number type of the former kernel [11]. Notice that while the standard filtering strategy consists of re-computing a quantity at the predicate level, we do the same but at the arrangement level! As we shall see, this strategy boosts the execution for large sets of balls.

[11] Practically, the C++ code uses a **try {} catch {}**.

Table 3 Predicates and constructions required. Construction are required to report an embedding of the arrangement on the sphere. In predicates, *point* stands for critical or intersection point.

Constructions	**Predicates**
Compute critical points of a circle	Test intersection between two arcs
Compute intersection points of two circles	Compare the value of θ of points
	Compare the Cartesian coordinates of two points
	Give the position of a point w.r.t. a plane
	Sort arcs intersected by $M_{2\pi}$–

10 Experiments and Applications

10.1 Datasets

Four types of datasets were used to run experiments, depending on the goals pursued:
– to investigate the practical behavior of the algorithm, tests were run on collections of random circles on the unit sphere S_0 centered at the origin, each circle being defined by its center, picked uniformly at random within the ball bounded by S_0. (Notice this strategy does not yield great circles.) Values reported are averages over 10 different runs with the same parameters. The number of circles chosen varies from 10 to 400 by steps of 10;
– to compare with previous work [EH05], we focused on 4 PDB files (PDB codes: `1bzm`, `1jky`, `7at1`, `1l7x`).
– for applications in structural biology, we respectively used the 96 protein-protein complexes from [CCJ99], as well as selected protein - drug complexes from the ASTEX dataset [HVC+07].

Except in the case of comparison with previous work —cf Table 4, all computations were run on a bi-proc Pentium IV(R) 3.06Ghz with 2.5GB of RAM.

10.2 Practical behavior

In investigating the practical behavior of the algorithm, two topics are of interest: first, the running time with respect to the theoretical complexity $c(n+k)\log n$, and second the impact of function `break_adjacency` on the constant c, as this function maintains the linear size event queue.

Complexity and constants. To assess the value of the constant in the algorithm overall complexity, we performed curve fitting on running times. As illustrated on Fig. 22, the theoretical curve $c(n+k)\log n$ is in excellent agreement with the experimental curves, whatever the variant —double, exact or exact not using `break_adjacency`. Setting the constant c to match the experimental time obtained for $n = 400$ and the corresponding k, we observe the

constant factor is $1.4 \times 10^{-5}/9.2 \times 10^{-5}/3.2 \times 10^{-4}$ for the double/exact/exact not using `break_adjacency` variant.

Linear size event queue. To assess the impact of function `break_adjacency`, we compared our algorithm against a variant which does not remove from event sites of $\mathcal{E}$ intersection events that no longer correspond to two adjacent arcs along $\mathcal{V}$ —excepted when they have been handled of course. For arrangements of random circles, the performances of our version are much better. This actually corresponds to a much smaller priority queue —Fig. 23(Left), most of the events found in the non cleaned up queue being actually non valid —Fig. 23(Right).

To shed more light on such performances, let us investigate the connexion between the queue size and filter failures. Without function `break_adjacency`, the size of $\mathcal{E}$ may become quadratic. If insertion into $\mathcal{E}$ had a constant cost, no performance degradation would occur, as $\log m = O(\log n)$ if $m = O(n^2)$. But not cleaning up $\mathcal{E}$ may result in repeated insertions of the same event. Numerically, comparing the cylindrical coordinates of critical and intersection points relies on the comparison of algebraic numbers of degree two. In comparing the coordinates of these points in order to maintain $\mathcal{E}$—and more generally in using the total order introduced in Def. 8, one can use efficient filtering strategies to avoid resorting to exact calculations on these algebraic numbers. These strategies are actually twofold: geometric on one hand, and numeric on the other hand —as `Lazy_exact_nt` is used as number type. (For the geometric filtering, see the comparison of cylindrical coordinates on a reference sphere in the companion paper [CLdCTon].) But in re-inserting an event already present in $\mathcal{E}$, these filters fail since filters are useless against equality tests.

As an illustration, we considered a random collection of 200 circles on a sphere, the arrangement featuring $14,946$ intersection points. During the run of the algorithm, $17,153$ intersection events become non-valid. (Note that we have no indication on the distribution of these $17,153$ events amongst the $14,946$ intersection points.) Using the `CGAL_PROFILE` flag with the `Lazy_Exact_NT<Gmpq>` number type, we observe that when removing non-valid intersection events, $343,247$ calls to predicates comparing two algebraic numbers are done, 4 of them only needing exact values to conclude —filter failures. If non-valid intersection points are not removed, the number of calls increases to $432,572$, and the number of filter failures to $34,310$. Observe that the increase of failures is exactly twice the number of non valid events, as any non valid event yields filter failures for the comparison of the θ and z coordinates [12].

Numerics. Consider Tables 4 and 5. The ratio between the *double* variant and the *exact filtered* variant is of 2. But calculations in double fail on (nearly-)degenerate inputs. One such highly degenerate example is displayed on Fig. 21. Failures also happen for molecular models, since 6 atoms had to be removed to get the *double* row in Table 5. Numerics may be improved in two ways. First, we may use the CGAL `Lazy_kernel`. This kernel consists

[12] To be precise, we get one failure for the comparison of $\tan\theta$ or $\cot\theta$, and one for that of z. See the companion paper for the strategy used to compare the cylindrical coordinates of two event points.

of filtering at the predicate level, which provides a gain factor of two w.r.t. filtering at the number type level as reported in [FP06]. This filtering would be intermediate between filtering the number type and filtering the whole arrangement calculation. Second, we may design static filters. For $3D$ Delaunay triangulations, such filters yield a modest 30% overhead w.r.t. a double arithmetic [MP05]. Notice these two lines are independent, as the constructions allowed by the lazy kernel may be inter-twined with static filters to re-use intermediate calculations.

10.3 Comparison to previous work

The only running times we are aware of to compute an arrangement of circles on a sphere are those obtained with the algorithm based on explicit perturbations of spheres [EH05]. We may compare the performances although this code reports a perturbed arrangement —and not the exact one. The results found in Table 4 were produced using a bi-proc Pentium III(R) 1GHz with 2GB of RAM computer for [EH05], and a Pentium III(R) 1Ghz with 1GB of RAM in our case. On these examples, our code is about 65% faster (20% slower) using the double (exact filtered) variant. But as mentioned when discussing numerics, a progression margin remains in the exact realm.

10.4 Applications in structural biology

Protein - Protein interfaces. The Solvent Accessible Surface (SAS) of a molecule is defined as the boundary of its VdW balls, expanded by a water probe of $r_w = 1.4$Å. Given two molecules (or subunits or partners) forming a complex, the Buried Surface Area (BSA) is defined as the area of the SAS of the partners which is getting buried upon formation of the complex. The BSA is a parameter of major interest in all structural studies of interfaces in macro-molecular complexes [CJ02, BCRJ04]. To go beyond this traditional model, we resort to the arrangement computed on each atom from the intersection circles with neighboring atoms.

Consider a complex featuring two partners A (red) and B (blue), and assume the arrangement has been computed for each atom. For a given face of the arrangement of a given atom, we adopt the following model, which we call the ESBI model. The face is termed: *Exposed* if it contributes to the boundary of the union of balls in the complex; *Self* if it is covered by atoms of the same subunit only; *Buried* if it is exposed in its own subunit, but buried in the complex; *Interaction* if it is buried in its own subunit, but also covered by atoms of the partner in the complex.

In the complex, the Interaction Surface Area (ISA) is thus defined as the sum of the areas of all interaction faces. As reported in Table 6, the ISA is about 4 times larger than the BSA. This dramatic difference 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. A detailed bio-chemical discussion of this result is beyond the scope of this paper, and the reader is referred to [BCLon], which also investigates applications to statistical potentials modeling multi-body inter-atomic contacts.

Protein - Drug interfaces. The notion of BSA and ISA can naturally be extended to protein-drug complexes. In trying to predict which regions of a protein may accommodate a drug –call such a region a pocket, a major problem consists of identifying properly the pockets. As a prerequisite, we may examine co-crystallized protein-drug complexes, and define the pocket as those atoms in contact with the drug. In the following, we sketch one way to do so using the ESBI model, referring the reader to [AC+07] for the details. The protein-drug complexes used are from the ASTEX dataset [HVC+07] —a curated dataset featuring *reliable* protein - drug complexes.

Consider a co-crystallized complex. We define the pocket as the protein atoms respectively satisfying $B > 0$, and $\{B = 0, I > 0\}$. These two classes of atoms form two layers, as illustrated on Fig. 2. Similarly to the protein-protein case, this model goes beyond the traditional ones which were focusing on solvent accessible properties. A number of bio-chemical properties can be investigated for each layer: composition in terms of amino-acids, rotameric state of the corresponding amino-acids, etc. May be more interestingly in the perspective of the prediction of a given complex from the isolated partners, if the protein of the complex (call it the holo form) also has a crystallized form on its own (call it the apo form), we may investigate the conformational transitions underwent by the atoms of the two layers upon formation of the complex. To do so, we consider the ratio $\sum_{L_i \text{in holo}}(E+B)/\sum_{L_i \text{in apo}} E$, where L_i stands for the first or second layer just defined. For the first (second) layer, this ratio tends to be larger (smaller) than one. This implies that atoms of the first layer atoms increase their accessibility to the solvent/ligand, while the second layer atoms decrease it. This partially encodes the conformational changes of the protein upon binding the drug, and holds great promises to score putative complexes from the partners.

Figure 21 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 $674 - 1384 + 712 = 2$.

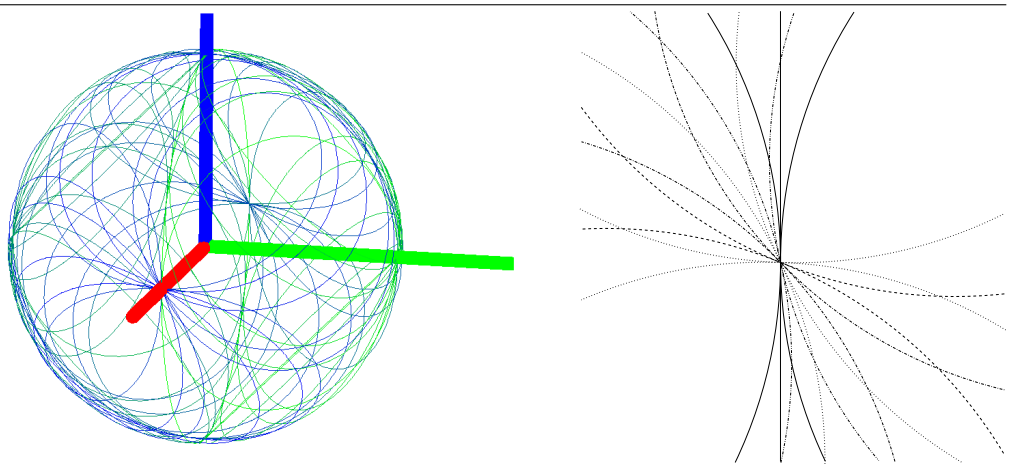


Figure 22 Observed vs theoretical complexities for random arrangements

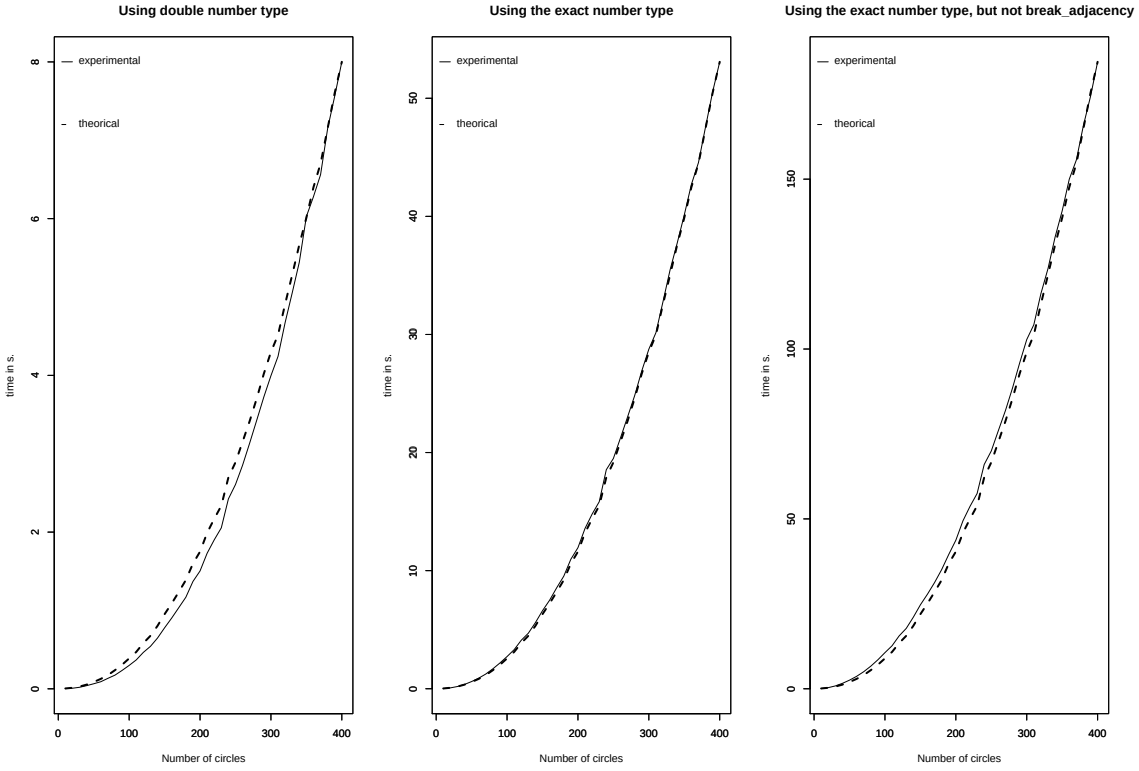


Table 4 Tests on 4 macro-molecular structures. Total time (in seconds) for computing the surface (including the perturbation) vs. total time of computing the exact arrangement of circles on each atomic spheres.

Input file	#atoms	[EH05]	Double	Exact	Exact filtered
1bzm	2049	8.31	4.98	45.51	9.47
1jky	5734	26.94	15.75	149.44	29.73
7at1	7169	28.40	17.00	162.70	32.89
1l7x	12912	54.50	33.00	311.17	64.32

Figure 23 On the importance of maintaining a linear size event queue. (Left) Maximum number of events in $\mathcal{E}$ during the computation of the arrangement. (Right) Number of intersection points for each distribution of circles versus number of intersection points that become non valid along the algorithm.

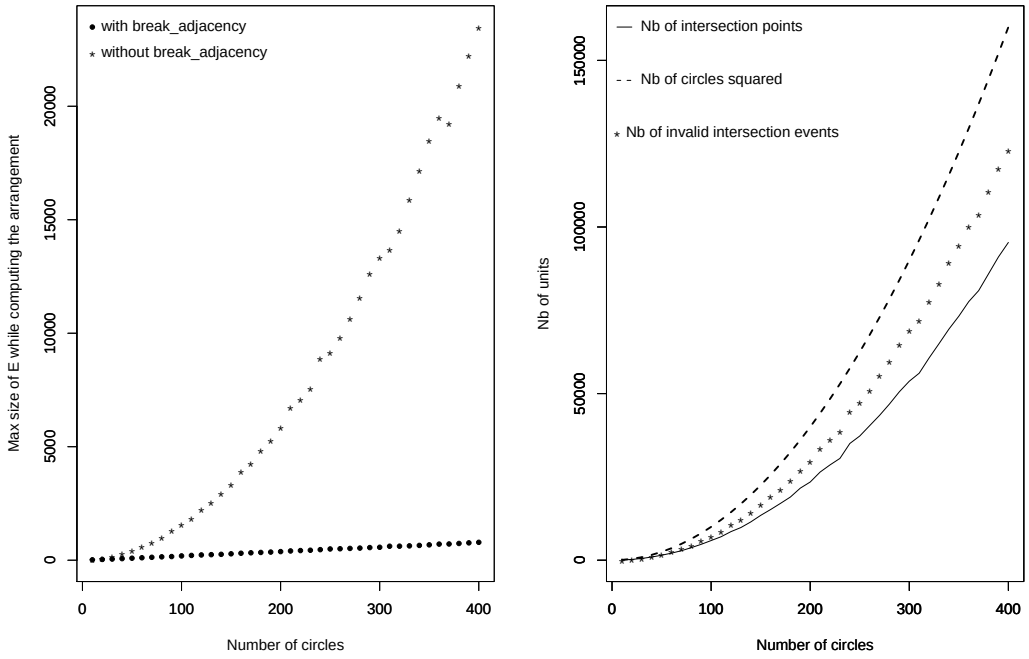


Table 5 Comparing Number Types (NT) for complex 1acb (2433 atoms): run-times to report the neighbors, compute the arrangements, and compute the surface areas of the cells on each sphere.

NT	**neighbor**	**argt**	**area**	**total**
double	0.11s	2.14s	0.71s	2.96s
exact	0.38s	21.47s	7.57s	29.42s
exact filtered	0.23s	4.01s	1.67s	5.91s

Table 6 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

11 Conclusion

This paper presents a generalization of the Bentley-Ottmann algorithm to compute the exact arrangement of circles on a sphere. The algorithm is non trivial due to the degeneracies and the algebraic specification of events, and is the first one able to cope with general circles on a sphere in an exact fashion. In developing the algorithm, we evidence the importance of maintaining a linear size event queue, an observation of general interest for Bentley-Ottmann algorithms dealing with curved objects. 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, so as to capture the subtlety of multi-body interactions between non-covalent contacts.

A number of perspectives remain open. On the numerical side, arithmetic profiling shows that performances should be improved by dedicated static filters. On a more general algorithmic perspective, as our algorithm is dedicated to circles on a sphere, the question of coming up with more general algorithms featuring the same performances deserves investigations.

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

This work is partially supported by the IST Programme of the 6th Framework Programme of the EU as a STREP (FET Open Scheme) Project under Contract No IST-006413 - ACS (Algorithms for Complex Shapes with certified topology and numerics) - `http://acs.cs.rug.nl/`.

References

[AC+07] S. Afraoui, F. Cazals, , S. Loriot, P. Tufféry, and B. Villoutreix. A morphological study of pockets in protein-drugs complexes. 2007. In preparation.

[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.

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

[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.

[BEHH02] E. Berberich, A. Eigenwillig, M. Hemmer, and S. Hert. A computational basis for conic arcs. and boolean operations on conic polygons. In *ESA*, 2002.

[BFHW] E. Berberich, E. Fogel, D. Halperin, and R. Wein. Sweeping over curves and maintaining two-dimensional arrangements on surfaces. Manuscript, Max-Planck-Institut für Informatik, Saarbrücken, and Tel Aviv University. December, 2006.

[BHK+05] E. Berberich, M. Hemmer, L. Kettner, E. Schömer, and N. Wolpert. An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 99–106, 2005.

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

[CCJ99] L.L. Conte, C. Chothia, and J. Janin. The atomic structure of protein-protein recognition sites. *J. Mol. Biol*, 285:2177–2198, 1999.

[CGT04] A-C. Camproux, R. Gautier, and P. Tuffery. A hidden markov model derived structural alphabet for proteins. *J. Mol. Biol.*, pages 591–605, 2004.

[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.

[CLdCTon] F. Cazals, S. Loriot, P. Machado de Castro, and M. Teillaud. Design of the cgal 3*d* spherical kernel. *In preparation*, In preparation.

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

[CPBJ06] 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.

[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.

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

[FHK+06] E. Fogel, D. Halperin, L. Kettner, M. Teillaud, R. Wein, and N. Wolpert. Arrangements. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 1–66. Springer-Verlag, Mathematics and Visualization, 2006.

[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.

[Hum99] G. Hummer. Hydrophobic force field as a molecular alternative to surface-area models. *J. Am. Chem. Soc.*, 121:6299–6305, 1999.

[HVC+07] M.J. Hartshorn, M.L. Verdonk, G. Chessari, S.C. Brewerton, WT Mooij, P.N. Mortenson, and C.W. Murray. Diverse, high-quality test set for the validation of protein-ligand docking performance. *J. Med. Chem*, 50(4):726–741, 2007.

[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.

[MN00] Kurt Mehlhorn and Stefan Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.

[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.

[MTT05] Bernard Mourrain, Jean-Pierre Técourt, and Monique Teillaud. On the computation of an arrangement of quadrics in 3d. *Computational Geometry: Theory and Applications*, 30:145–164, 2005. Special issue, 19th European Workshop on Computational Geometry.

[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.

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

[SLD05] C. Summa, M. Levitt, and W. DeGrado. An atomic environment potential for use in protein structure prediction. *JMB*, 352, 2005.

[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.

12 Appendix: pseudo-code

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

This section provides the pseudo-code of the main functions. 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.

The pseudo-code of Algorithm 3 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 2 Bentley-Ottmann: pseudo-code

```
1: classify_circles {find circle types }
2: Initialize vertical order V
3: Initialize event queue E
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
```

Algorithm 3 handle_event_site

1: $arc\_sup = \mathbf{NULL}$
2: $arc\_inf = \mathbf{NULL}$
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** $(arc\_sup = \mathbf{NULL})$ **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 circle is 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_CT_block_bounds`
44: { if arc_sup and arc_inf are **NULL**, they are respectively updated to the upper and lower bounding arcs of the block defined by $\mathcal{CT}$ }
45: Reverse block of all arcs defined by $\mathcal{CT}$ in-between arc_sup and arc_inf
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 top,bottom and lower,upper of bounding arcs of the blocks
51: Update $\mathcal{E}$ accordingly
52: **end if**
53: `topo_handle_above_below`

13 Appendix: handling the topology at (bi)polar events

This section complements the algorithms presented in section 7.2 by providing the pseudo-code for algorithms `topo_handle_polar_event_site` and `topo_handle_bipolar_event_site`. 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.

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 have to manage the half-edges passing by its corresponding pole. This can be done using the function `topo_handle_polar_event_site` 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_site` 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. 15 for illustration).
▷ **3.** We relaunch the routine `topo_handle_polar_event_site` but for the south pole.

These operations are summarized in algorithm `topo_handle_bipolar_event_site`. 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 is `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 is `in`(C) (`out`(C)) at a start event (end event), with the correct intersection point associated to one extremity of the half-edge.

Algorithm 4 `topo_handle_polar_event_site`

```
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

Algorithm 5 Algorithm `topo_handle_bipolar_event_site`

1: L_h=`topo_new_left_halfedge`
2: R_h=`topo_new_right_halfedge`
3: `topo_handle_polar_event_site`(NORTH_POLE)
4: event_site *evt_pol*=$\mathcal{E}$.`pop`{Bipolar event site}
5: *current* = pointer on the successor of the north pole in $\mathcal{V}$
6: *stop* = pointer on the 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* = pointer on the lower bounding arc of the 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* =pointer on the top arc of the block defined by lists of $\mathcal{E}$.`top`
14: **else**
15: *stop* = pointer on the 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_site`(SOUTH_POLE)

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