Design of the CGAL Spherical Kernel and application to arrangements of circles on a sphere

Pedro de Castro, Frédéric Cazals, Sebastien Loriot, Monique Teillaud

Design of the CGAL 3D Spherical Kernel and application to arrangements of circles on a sphere [*]

Pedro M. M. de Castro [†‡] Frédéric Cazals [†] Sébastien Loriot [†§]
Monique Teillaud [†]

October 30, 2008

Abstract

This paper presents a CGAL kernel for algorithms manipulating 3D spheres, circles, and circular arcs. The paper makes three contributions. First, the mathematics underlying two non trivial predicates are presented. Second, the design of the kernel concept is developed, and the connexion between the mathematics and this design is established. In particular, we show how two different frameworks can be combined: one for the general setting, and one dedicated to the case where all the objects handled lie on a reference sphere. Finally, an assessment about the efficacy of the 3D Spherical Kernel is made through the calculation of the exact arrangement of circles on a sphere. On average while computing arrangements with few degeneracies (on sample molecular models), it is shown that certifying the result incurs a modest factor of two with respect to calculations using a plain double arithmetic.

[*]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/`

[†]INRIA, BP93, 06902 Sophia Antipolis cedex, France, `Firstname.Lastname@sophia.inria.fr`, `http://www-sop.inria.fr/geometrica/team/`

[‡]Pedro.Machado@sophia.inria.fr

[§]IMB - Université de Bourgogne

1 Introduction

1.1 Linear versus Curved Primitives in Computational Geometry.

Until recently geometric algorithms used to limit their framework to linear objects in affine Euclidean space (points, segments, triangles, ...), curved objects being discretized by linear elements. However, handling directly curved objects allows one to skip this discretization process, thus reducing the number of primitives and yielding algorithms with better combinatorial complexity. For these reasons, the direct manipulation of curved objects is an important challenge in computational geometry. Of particular interest in the realm of curved objects are the simplest primitives, namely circles, circular arcs and spheres, which are ubiquitous to either represent or approximate more elaborate objects.

Regarding one dimensional primitives, circular arcs in the plane are involved in the design and the manufacturing of printed and integrated circuits [12], and they also generate an increasing interest for approximating curves [19, 2].

Balls and spheres are probably even more central, especially to approximate 3D objects. Given a known 3D object, balls can be used to provide hierarchical approximations, for example to efficiently perform collision detection [1] or multi-scale visualization [36]. Given a 3D point cloud sampling an unknown object, balls are also key for the inference of geometric and topological properties of the unknown object from the samples. Topologically speaking, balls can indeed be used to reconstruct the object by mimicking the medial axis transform, which is the bottom-line of say the power crust algorithm [3]. From a more geometric standpoint, a number of quantities can be inferred from balls centered at the sample points. One may mention the so-called boundary measure of a point cloud, from which singular points and sharp features of the sampled model can be estimated [16]. This inference requires computing the volume of a ball contained within its Voronoi region, a calculation involving geometric constructions on a sphere [33, 5]. Given a 3D polyhedron, additional geometric modeling constructions involving spheres are the computation of offset surfaces. (Notice that the offset surface calculation covers the 3D Minkowski sum between a polyhedron and a sphere.) Interested readers may also consult chapter 11.12 of the Visionbib bibliography project at `http://www.visionbib.com/bibliography/describe482.html`, which contains an exhaustive list of representations involving spheres.

Finally, spheres are also central in molecular modeling since a Van der Waals model features a collection of balls whose radii depend upon the atom type and its covalent environment. In particular, the combinatorial and geometric properties of the three main definitions of surface of a molecule used (the Van der Waals surface, the solvent-accessible surface and the molecular surface) read from the boundary of a union of balls [17].

1.2 From Algorithms to Implementations

Predicates versus Constructions in the Exact Geometric Computation Paradigm. Recent implementations of geometric algorithms usually distinguish predicates and constructions. In standard folklore, a *predicate* is a test function returning an element of a discrete set of values (for instance: *do the two curves c and c' intersect?*). A *construction* is a function that constructs new objects (for instance: *compute the intersection points between curves c and c'*). Since rounding errors due to floating point calculations often produces inconsistencies [28], the robustness of geometric algorithms relies on the exact evaluation of predicates [39]. This problem usually boils down to evaluating the sign of polynomial expressions on the input data, which can be done efficiently resorting to arithmetic filtering techniques [31]. Aside this classical strategy, one further refinement consists of storing, in the course of the predicate evaluation, intermediate geometric objects to be reused later on. In the realm of curved objects, the former strategy typically consists of using an implicit encoding of an algebraic number by some polynomial vanishing on it, which calls for resultant-like calculations on the input polynomials. The latter relies on exact representation of algebraic numbers [18, 30], and requires in the end running multi-precision calculations down to separation bounds in the worst cases [10, 32]. But in doing so, one reuses intermediate calculations, and the design of predicates, a difficult task for algebraic objects, is made easier. In fact, work on filtered constructions [25, 23] may reconcile the evaluation of predicates and constructions. The implementation of such a framework still needs to be worked out to possibly surpass the approach relying on predicates without intermediate constructions.

The CGAL Library, Kernels and Traits Classes. The goal of the CGAL open source project is to promote research in computational geometry, so as to make reference algorithms available as robust programs for academic research and industrial applications. We refer the reader to the CGAL web site and to the literature for more background on the history and running of the CGAL project [15, 24]. Geometric algorithms are generic, and the geometric classes are instantiated with a *Traits* class—offering the minimum set of functionalities required. Such a template parameter is supposed to provide a set of functionalities, with prescribed signatures, described and documented as a *concept.*
In the CGAL library, constant size geometric primitive objects, together with general purpose predicates and constructions on them, are made available in *kernels.* As kernels can be directly used as traits classes by several CGAL classes, CGAL kernels are documented as C++ concepts—we say that one implementation of a given concept is a *model* of the concept.

1.3 Circles and Spheres: from Primitives to Algorithms

As a follow-up to algorithms geared towards linear primitives, a recent trend in computational geometry has been the design of algorithms and programs geared towards curves and surfaces. The work mostly focused on computing arrangements in 2D and 3D, which involves primitive computations such as intersection and relative position of objects.

As far as we know, in the planar case MAPC was the first geometric library handling general algebraic curves [29], but all degenerate cases were not handled. The EXACUS prototype library allows to exactly compute arrangements of conics and cubics in the plane [21]. The first version of a kernel for non-linear objects (circles and circular arcs in 2D) was released in CGAL 3.2 [35]. The CGAL arrangement package can use this kernel to compute an arrangement of circular arcs in the plane; it can also deal with conics or Bézier curves [37].

In the three dimensional setting, Andrade and Stolfi proposed selected exact manipulations of 3D circular objects lying on a given sphere, with an implementation in Modula-3 [4]. Their representation of objects uses Plücker coordinates and is restricted to the case of circular arcs on a given sphere, whereas our framework handles general circles and circular arcs in 3D. To the best of our knowledge, there exist two algorithms computing the exact arrangement induced by a set of circles on a sphere. The first one is generic as its specification handles general curves on a parametric surface, the strategy consisting of sweeping the parametric domain of the surface [7]. But for a surface which is a sphere, only great circles are supported in the current implementation. The second one is an algorithm which consists of sweeping a sphere with a meridian anchored at the poles. The algorithm handles all types of circles and any degenerate case [14]. We report in section 5 how the 3D Spherical Kernel is used in the implementation of this algorithm.

While discussing arrangements and for the sake of completeness, one should also mention papers dealing with quadrics in 3D. The first complete and exact implementation computing a planar map induced by the intersection curves of a set of quadrics running on the surface of one of them, say Q, is described in [8]. The algorithm reduces the problem to a planar arrangement of algebraic curves using a projection into a plane. Moreover, it is shown how this planar arrangement can be used to compute the arrangement on Q—a lifting step which can be avoided when directly computing the arrangement on Q in 3D. Finally, an algorithm to compute the exact volumetric decomposition induced by an arrangement of quadrics is developed in [34]. The strategy does not resort to surface arrangements, but no implementation is provided.

1.4 Contributions and Paper Overview

This work provides the first complete and efficient implementation of essential basic functionalities to manipulate in an exact way spheres, circles and circular arcs in 3D, so as to match most of the needs mentioned in section 1.1. As opposed to the general algorithms discussed in 1.3, the perspective is that of a hand-crafted package geared towards the aforementioned objects. Following best practices, a clear distinction is made between the algebraic and the geometric aspects on the one hand, and on the concepts of a kernel and its implementation on the other hand. The 3D Spherical Kernel concept is accompanied by an implementation, which is used to compute the exact arrangements of circles on a sphere. As a quality label, the basic part of the package, related with general spheres, circles and circular arcs in 3D, has been reviewed and accepted by the CGAL Editorial Board [13]. The second part of the package, offering functionalities on a reference sphere, will follow the process in a second step. In particular, the diffusion of our code will enable a number of endeavors. As an example, it will foster the development of a traits class providing the predicates required to handle all types of circles in the general sweep-line algorithm for curves on a surface [7].

The paper is organized as follows. Section 2 presents the mathematics underlying two non trivial predicates. Section 3 presents the design of the 3D Spherical Kernel and of the Algebraic Kernel. In Section 4, we bridge the gap between the geometric calculations presented in Section 2 and the functionalities of the kernel. The application to computing an arrangement of circles on a sphere is presented in Section 5. Finally, experiments on random arrangements and molecular models are reported in Section 6. Due to space limitations, the reader is referred to [11] for additional details skipped in this paper —calculations for predicates and constructions as well as example C++ code.

2 Mathematical Background

This section focuses on general basics underlying the 3D Spherical Kernel, as well as on the mathematics needed for two non-trivial operations: The construction of the θ-extremal points of a circle on a given sphere, and the computation of the relative orientation of the tangents of two intersecting circular arcs on a given sphere.

2.1 Preliminaries and Notations

The center of a sphere S_i is denoted $c_i = (x_i, y_i, z_i)$, and its radius r_i. We assume the Cartesian coordinates of the center and the squared radius to be rational numbers. All objects (planes, lines, spheres, circles) whose equations have rational coefficients are termed *rational.*

The x-coordinate of point p and vector u are denoted p_x and u_x, and similarly for y- and z-coordinates. The dot and vector products of two vectors u and v are respectively denoted $< u, v >$ and $u \wedge v$. The squared norm of vector u is denoted $u^2 =< u, u >$, and its norm $\|u\|$. The sign of a real number, denoted $\text{Sign}(x)$, is such that $\text{Sign}(x) \in \{-1, 0, 1\}$. The vector defined by two points p and q is denoted pq. If o stands for the origin, and p is a point, the vector op is denoted $\overline{p}$.

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_{ij} of any two spheres S_i and S_j is the plane consisting of the points having equal power with respect to the two spheres. Its equation is $2 < \overline{p}, c_i c_j > + \overline{c_i}^2 - \overline{c_j}^2 + r_j^2 - r_i^2 = 0$. Whenever these two spheres intersect, their intersection circle C_{ij} is also defined as the intersection between either sphere and their radical plane. The center and radius of C_{ij} are denoted c_{ij} and r_{ij}. It can be checked that $c_i c_{ij} = \frac{c_i c_j^2 - r_j^2 + r_i^2}{2 c_i c_j^2} c_i c_j$ and $r_{ij}^2 = r_i^2 - c_i c_{ij}^2$.

A root of a degree n polynomial with rational coefficients is called an algebraic number of degree n. The following observation shows that algebraic numbers of degree two play a central role in our predicates and constructions:

Observation 1 *The Cartesian coordinates of the intersection points of a rational sphere and a rational line are algebraic numbers of degree two in the same extension.*

We also recall the following properties that will be used in our computations:

Observation 2 *Consider two algebraic numbers a and b, of degree at most two in the same algebraic extension. Then $a + b$, $a \times b$ and $1/a$ (if $a \neq 0$) are algebraic numbers of degree at most two, and they also belong to the same algebraic extension.*

2.2 Objects on a Reference Sphere

In the following and without loss of generality, we will consider a sphere S_0 centered at the origin called the *reference sphere.* S_0 is equipped with cylindrical coordinates (θ, z). Let M_θ be a parametrized meridian on S_0, with $\theta \in (0, 2\pi]$. We first define a classification of circles on S_0, and related quantities:

Definition 1 *Consider a circle on a reference sphere S_0. Such a circle is said to be* polar *(respectively* bipolar*) if it goes through one pole (respectively the two poles) of S_0. A circle which separates S_0 into two connected components, each containing one pole, is termed* threaded*. Any other circle is termed* normal*.*

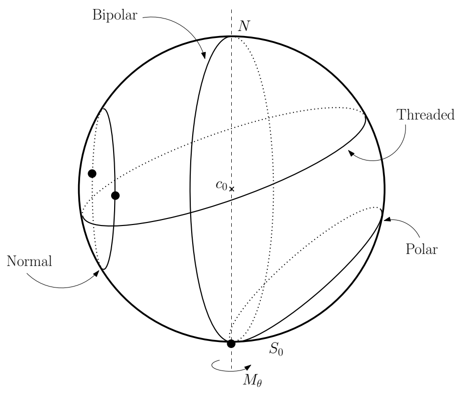


Figure 1: The four types of circles on a reference sphere. Black dots are the θ-extremal points.

For polar and normal circles, we introduce the notion of θ-extremal points.

Definition 2 *The* θ-extremal points *of a normal circle are the two points where a meridian of S_0 is tangent to the circle. The* θ-extremal point *of a polar circle is the pole the circle goes through.*

For all but threaded circles, we define the notion of θ-extremal value(s):

Definition 3 *For a normal circle, the* θ-extremal values *are the θ-coordinates of its θ-extremal points. For a polar circle, the* θ-extremal values *are the two values of θ such that M_θ is tangent to the circle at the pole. For a bipolar circle, the* θ-extremal values *are the two values of θ such that M_θ is contained in the plane of the circle.*

Notice that in standard analysis and geometric folklore, the θ-extremal points are also called critical points. Notice also that the θ-extremal points should not be confused with the endpoints of an arbitrary circular arc.

Definition 4 *A circular arc on a reference sphere is said to be* θ-monotone *if its intersection with any half-plane bounded by the line going through the two poles is empty or reduces to one point.*

Consider the four types of circles, as illustrated on Fig. 1. A circular arc on a normal circle is θ-monotone if it contains none of the θ-extremal points of its supporting circle, excepted maybe as an endpoint. The same remark is valid on a polar circle, considering its θ-extremal point. Any circular arc constructed on a threaded circle is by definition θ-monotone while no θ-monotone circular arc can be found on a bipolar circle.

If C_{0i} is a normal circle, the two circular arcs delimited by the θ-extremal points are called the *upper* or *lower* circular arcs as evidenced by their relative position along the z-axis. By extension, a θ-monotone circular arc included into the upper (lower) arc is also termed upper (lower) and a θ-monotone circular arc on a polar circle passing through the north (south) pole is termed lower (upper).

2.3 Computing the θ-extremal Points of a Normal Circle

We start with a proposition concerned with the z-coordinate of θ-extremal points:

Proposition 1 *Consider a normal circle C_{0i} defined by the intersection of two rational spheres. The z-coordinate of its two θ-extremal points is the following rational number:*

$$z = \frac{2z_i r_0^2}{\overline{c_i}^2 + r_0^2 - r_i^2}.$$

Proof. Recall that the reference sphere S_0 is centered at the origin, and let C_{0i} be a normal circle intersection of S_0 and S_i. If the z-coordinate c_{0i_z} of its center c_{0i} is equal to 0, a symmetry argument with respect to the plane $z = 0$ imposes that the z-coordinate of the θ-extremal points is also null. In the following, we therefore assume that $c_{0i_z} \neq 0$.

Consider the zero-dimensional intersection set between the circle C_{0i} (or equivalently the intersection $S_0 \cap RP_{0i}$) and the half-plane $P(\theta)$ defining the meridian M_θ (i.e. $M_\theta = S_0 \cap P(\theta)$). The θ-extremal points of the circle correspond to the case where this intersection set reduces to one point. Denoting $p = (x, y, z)$ such an intersection point, the corresponding polynomial system reads as:

$$\begin{cases} P(\theta): & -x\sin\theta + y\cos\theta = 0 \\ RP_{0i}: & 2 < \overline{p}, c_0c_i > +\overline{c_0}^2 - \overline{c_i}^2 + r_i^2 - r_0^2 = 0 \\ S_0: & x^2 + y^2 + z^2 - r_0^2 = 0 \end{cases} \tag{1}$$

Assuming $x \neq 0$, or equivalently $\theta \neq \frac{\pi}{2}\,[\pi]$, we investigate this system using $\tan\theta$. (If $x = 0$, we work with $\cot\theta$ by swapping the roles of x and y. Notice that x and y cannot vanish simultaneously for a normal circle, as such a circle does not contain a pole.) The previous system is tantamount to:

$$\begin{cases} y = x\tan\theta \\ z = \frac{\overline{c_i}^2 + r_0^2 - r_i^2 - 2x(x_i + y_i\tan\theta)}{2z_i} \\ x^2 + x^2\tan^2\theta + \frac{(\overline{c_i}^2 + r_0^2 - r_i^2 - 2x(x_i + y_i\tan\theta))^2}{4z_i^2} - r_0^2 = 0 \end{cases} \tag{2}$$

Rewrite the last equation of the previous system

$$Ax^2 + Bx + C = 0, \tag{3}$$

with

$$\begin{cases} A = (1 + \tan^2\theta) + \frac{(x_i + y_i\tan\theta)^2}{z_i^2} \\ B = -\frac{(x_i + y_i\tan\theta)(\overline{c_i}^2 + r_0^2 - r_i^2)}{z_i^2} \\ C = \frac{(\overline{c_i}^2 + r_0^2 - r_i^2)^2}{4z_i^2} - r_0^2 \end{cases} \tag{4}$$

The values of θ sought are such that Eq. (3) has a single solution, namely $x = -B/(2A)$. Imposing that the discriminant of this polynomial vanishes yields the condition $D = 0$, with $D = B^2 - 4AC$, whence $\tan\theta$ as an algebraic number of degree two. Letting T stand for $\tan\theta$ and denoting $\lambda_i = \overline{c_i}^2 + r_0^2 - r_i^2$, we have:

$$D = \overbrace{(-\lambda_i^2 + 4r_0^2z_i^2 + 4r_0^2y_i^2)}^{D_2}T^2 + \overbrace{(8x_iy_ir_0^2)}^{D_1}T + \overbrace{(-\lambda_i^2 + 4r_0^2z_i^2 + 4r_0^2x_i^2)}^{D_0} \tag{5}$$

Let us investigate the cases where polynomial D has degree two and one. As we shall see, in the first case, no θ-extremal point lies in the plane $x = 0$, while the second one corresponds to the situation where one θ-extremal point lies in plane $x = 0$. The latter case is left to the reader and we describe here only the former one.

We assume that $D_2 \neq 0$: Polynomial D is of degree two. We first make the connexion between the discriminant δ of D and the geometry, and proceed with the value of the z-coordinate of θ-extremal points. The discriminant δ of D may be written as:

$$\delta = 4\overbrace{(-2r_0z_i + \lambda_i)}^{\delta_1}\overbrace{(2r_0z_i + \lambda_i)}^{\delta_2}\overbrace{(4r_0^2\overline{c_i}^2 - \lambda_i)}^{\delta_3} \tag{6}$$

Since the power of a pole of S_0 w.r.t. to sphere S_i reads as $(c_i - (0, 0, \pm r_0))^2 - r_i^2 = \overline{c_i}^2 + r_0^2 - r_i^2 \pm 2z_ir_0 = \pm 2r_0z_i + \lambda_i$, polynomial δ_1 (respectively δ_2) is the power of the north (respectively south) pole w.r.t. S_i. The power of a point w.r.t. to a sphere is negative (respectively positive, zero) iff the point is inside (respectively outside, on) the sphere. Therefore, δ_1 is negative (respectively positive, zero) if the north pole is inside (respectively outside, on) S_i. The same observation holds for δ_2 and the south pole. Finally, for δ_3, observe that:

$$\delta_3 = \overbrace{(-\overline{c_i}^2 + (r_0 + r_i)^2)}^{\delta_{3_1}}\overbrace{(\overline{c_i}^2 - (r_0 - r_i)^2)}^{\delta_{3_2}}. \tag{7}$$

The combinations of signs of δ_{3_1} and δ_{3_2} are reported in Table 1, and the discussion of the sign of δ is summarized in Table 2. These tables prove that as expected, one has $\delta > 0$ for all normal circles, so that Eq. (5) can be cancelled in two different ways corresponding to the two θ-extremal points.

Finally, substituting any solution of Eq. (5) into the expression of z from the system (2) proves the claim under the assumption $D_2 \neq 0$. Notice that $\overline{c_i}^2 + r_0^2 - r_i^2 = 0$ corresponds to the case where the circle is bipolar. □

$\boldsymbol{\delta_{3_1}}$	$\boldsymbol{\delta_{3_2}}$	$\boldsymbol{\delta_3}$	**sphere configuration**
$+$	$+$	$+$	S_0 and S_i intersect along a non-degenerate circle
$+$	$-$	$-$	S_i inside S_0 or S_0 inside S_i, no intersection
$-$	$+$	$-$	S_i outside S_0, no intersection
$-$	$-$	$+$	impossible
0	$+$	0	S_i and S_0 tangent, no inclusion
0	$-$	0	impossible
$+$	0	0	S_i and S_0 tangent, S_i inside S_0 or S_0 inside S_i
$-$	0	0	impossible

Table 1: Sign of δ_3 from Eq. (7). Abusing terminology, S_i *inside* S_j stands for S_i *inside the ball associated to* S_j while S_i *outside* S_j stands for the *balls associated to* S_i *and* S_j *are disjoint.*

$\boldsymbol{\delta_1}$	$\boldsymbol{\delta_2}$	$\boldsymbol{\delta_3}$	$\boldsymbol{\delta}$	**circle type**
$+$	$+$	$+$	$+$	normal circle
$+$	$+$	$-$	$-$	S_0 and S_i do not intersect
$+$	$-$	$+$	$-$	threaded circle
$-$	$+$	$+$	$-$	threaded circle
$+$	$-$	$-$	$+$	impossible
$-$	$+$	$-$	$+$	impossible
$-$	$-$	$+$	$+$	normal circle
$-$	$-$	$-$	$-$	S_0 and S_i do not intersect
0	$\neq 0$	≥ 0	0	polar circle if $\delta_3 > 0$, S_0 and S_i are tangent otherwise
$\neq 0$	0	≥ 0	0	polar circle if $\delta_3 > 0$, S_0 and S_i are tangent otherwise
0	$\neq 0$	$-$	0	impossible
$\neq 0$	0	$-$	0	impossible
$\pm$	$\pm$	0	0	tangent spheres if $\delta_1\delta_2 > 0$, impossible otherwise
0	0	$\{\pm, 0\}$	0	bipolar circle if $\delta_3 > 0$, impossible otherwise

Table 2: Sign of δ from Eq. (6). δ_1 (respectively δ_2) is the power of the north (respectively south) pole w.r.t. S_i while δ_3 indicates whether S_0 intersects S_i.

Proposition 1 has an interesting consequence:

Corollary 1 *The θ-extremal points of a normal circle C_{0i} are defined by the intersection between the reference sphere, and the intersection line between the radical plane RP_{0i} and the horizontal plane defined by $z = \frac{2z_i r_0^2}{\overline{c_i}^2 + r_0^2 - r_i^2}$. Therefore, by Observation 1, the x- and y-coordinates of these points are algebraic numbers of degree two in the same extension.*

2.4 Computing the Relative Orientation of the Tangents of two Circles

Consider two circles, each supported by a rational plane, and intersecting transversally at point p on S_0. Under the assumptions that none of the circles is bipolar and that p is neither a pole nor a θ-extremal point of a normal circle, we wish to compute the relative orientation of the tangents of these two circles at p.

Under the previous assumptions, denote t_k, $k = i$ or j, the tangent vector associated to circle C_{0k} at point p, as illustrated on Fig. 2. Tangents vectors t_i and t_j are chosen so as to point towards *increasing* values of θ, locally in the tangent space of S_0 at p. Because t_i and t_j are orthogonal to $\overline{p}$, the relative orientation of the tangents is given by Sign(Δ), with

$$\Delta = < t_i \wedge t_j, \overline{p} > . \tag{8}$$

The details are as follows. The tangent to a circle is supported by the line intersection of the plane of the circle together with the tangent plane of S_0 at p. More precisely, consider the vectors $n_i = c_{0i}p$

and $n_j = c_{0j}p$. For $k = i, j$ the tangent vectors are obtained as follows: For a normal or a polar circle, $t_k = \beta_k \overline{c_{0k}} \wedge n_k$, with $\beta_k = -1$ (respectively 1) if the circular arc containing p is an upper (respectively a lower) one; For a threaded circle, let m_k be $\overline{c_{0k}}$ if the circle is not a great circle and otherwise any normal vector of the plane of the circle, then $t_k = \gamma_k(m_k \wedge n_k)$, with $\gamma_k = \pm 1$ such that $\gamma_k m_{kz} > 0$.

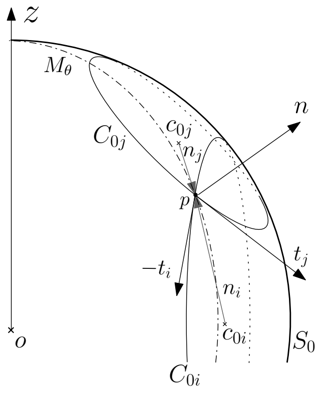


Figure 2: Tangent vectors t_i and t_j to circles C_{0i} and C_{0j} at one of their intersection points p. Vector n is the normal to S_0 at p.

The Cartesian coordinates of p are three algebraic numbers of degree two in the same extension. Computing tangent vectors using the formulation $t_k = \pm(c_{0k_y}p_z - c_{0k_z}p_y, c_{0k_z}p_x - c_{0k_x}p_z, c_{0k_x}p_y - c_{0k_y}p_x)^t$, we now give the cost of computing Δ in term of products $\Pi_{T;T'}$ and sums $\Sigma_{T;T'}$, where T and T' are either algebraic numbers of degree two (denoted AN_{op}) or rational numbers (denoted R). AN_{op} is a trivariate rational polynomial expression in the coordinates of p. This ensures that it is an algebraic number of degree two lying in the same extension as p. If considering the tree of arithmetic operations representing such an expression, the subscript op is equal to the maximum number of edges between the final expression and a coordinate of p. For instance each coordinate of t_k is a AN_2 obtained through two $\Pi_{R;AN_0}$ followed by one $\Sigma_{AN_1;AN_1}$. A naive computation of Δ requires two $\Pi_{AN_2;AN_2}$ and one $\Sigma_{AN_3;AN_3}$ for each coordinate of the vector product; three $\Pi_{AN_4;AN_0}$, one $\Sigma_{AN_5;AN_5}$ and one $\Sigma_{AN_6;AN_5}$ for the dot product. In Observation 4, omitting products of rationals, we show that if no circle is a great circle, we can actually compute the sign of Δ with only two $\Pi_{AN_2;AN_2}$, two $\Pi_{R;AN_3}$ and one $\Sigma_{AN_4;AN_4}$. We use the following elementary observation:

Observation 3 *Let $\Delta' = \frac{t_{j_z}}{\|t_j\|} - \frac{t_{i_z}}{\|t_i\|}$. The sign of Δ matches that of Δ'.*

Notice that the sign of Δ' is trivially inferred in the following three cases: $t_{i_z} = 0$ and $t_{j_z} = 0$; $t_{i_z} = 0$ or $t_{j_z} = 0$; t_{i_z} and t_{j_z} have different signs.

Otherwise we have:

Observation 4 *If none of the circles involved is a great circle and the signs of t_{i_z} and t_{j_z} are identical, the sign of Δ' is given by:*

$$Sign(\Delta') = Sign(t_{j_z})Sign\left(\overline{c_{0i}}^2 r_{0i}^2 (t_{j_z})^2 - \overline{c_{0j}}^2 r_{0j}^2 (t_{i_z})^2\right)$$

Proof. In the following, we suppose without loss of generality that the signs of t_{i_z} and t_{j_z} are identical and non null. The sign of Δ' is the same as that of $\|t_i\|t_{j_z} - \|t_j\|t_{i_z}$. and we have:

$$t_i^2 t_{j_z}^2 - t_j^2 t_{i_z}^2 = (\|t_i\|t_{j_z} + \|t_j\|t_{i_z})(\|t_i\|t_{j_z} - \|t_j\|t_{i_z}) \tag{9}$$

And since we supposed that $\text{Sign}(t_{j_z}) = \text{Sign}(t_{i_z})$, we also have

$$\text{Sign}(\Delta') = \text{Sign}(t_{j_z})\text{Sign}(t_i^2(t_{j_z})^2 - t_j^2(t_{i_z})^2). \tag{10}$$

Recall that the Cartesian coordinates of the center of circle $C_{0k}, k = i, j$, are rational numbers. As $\overline{c_{0k}}$ and $c_{0k}p$ are orthogonal, the sine of the angle between the vectors is 1. Since $t_k = \pm\overline{c_{0k}} \wedge c_{0k}p$, we have $t_k^2 = \overline{c_{0k}}^2 c_{0k}p^2$, which we can rewrite $t_k^2 = \overline{c_{0k}}^2 r_{0k}^2$.

□

3 Software Design

In this section, we sketch the design perspective of CGAL kernels in general, and focus on the 3D Spherical Kernel.

3.1 CGAL Kernels: Rationale

The general design of a kernel concept for curved objects initially proposed in [20] is driven by the following goals: (i)interoperability of any model of the kernel concept with CGAL geometric algorithms; (ii)re-usability of the existing CGAL kernel models for linear objects; (iii)genericity and flexibility: ability to use other linear kernels than the CGAL ones, and independence from a particular implementation of the algebraic operations needed. The declaration of a kernel for curved objects is the following:

```
template < typename LinearKernel, typename AlgebraicKernel >
class Curved_kernel;
```

Practically, to indicate whether an object is a member of the 3D Spherical Kernel or of the Algebraic Kernel, we use the following prefixes `SK::` and `AK::` respectively.

Let us now comment briefly on the two template arguments. Using the extensibility and adaptability scheme of the CGAL kernel [27], the `Curved_kernel` inherits from the model of `LinearKernel` given as template parameter, which allows it to directly benefit from all the functionalities on linear objects. The `LinearKernel` concept will not be presented here. It coincides with the basic CGAL kernel concept extensively documented in the CGAL manual.
The requirements on the `AlgebraicKernel` concept listed in Section 3.3 are guided by the functionalities offered by the 3D Spherical Kernel (Section 3.2). Notice that this Algebraic Kernel concept is actually called `AlgebraicKernelForSpheres` in the CGAL package because it is restricted to functionalities required by the 3D Spherical Kernel. Nevertheless, we use the shorter name `AlgebraicKernel` in this paper.

At the embedding level, the number type used to represent the objects of the linear kernel (coordinates for points, coefficients of equations for planes, spheres) is supposed to lie in a *field number type* (a type providing elementary operations $+, -, \times, /$), that will typically be the rationals. From now on and for the sake of simplicity, we shall refer to this basic number type as *rational*.

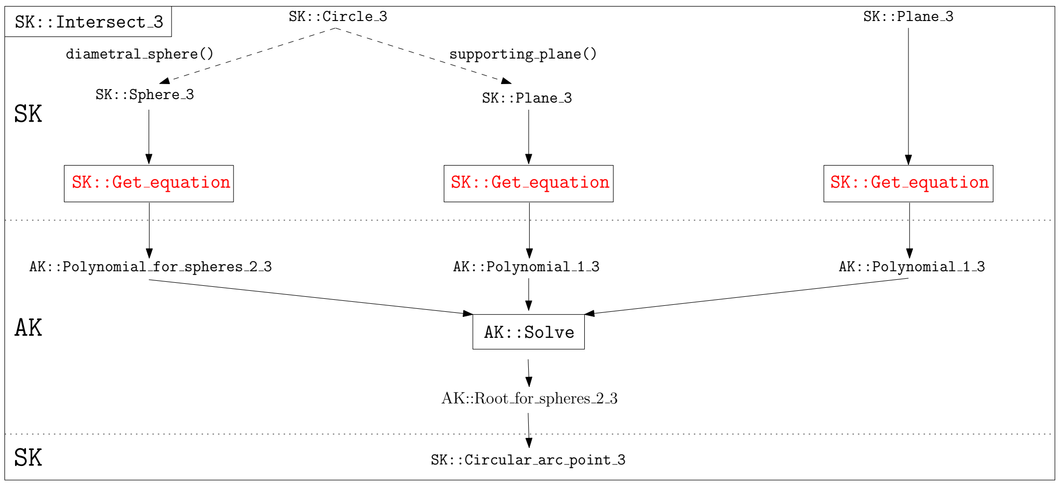
Regarding algebra, the key idea in this design is that each primitive object of the 3D Spherical Kernel is internally represented by a corresponding object of the Algebraic Kernel–its equation or coordinates. Moreover, each functionality of the 3D Spherical Kernel is obtained by a combination of calls of functionalities of the Algebraic Kernel on the algebraic representations of its arguments. Fig. 3 illustrates how algebra and geometry communicate, on the example of the computation of the intersection of a circle and a plane: We first retrieve the underlying primitive geometric objects, get their equations as polynomials of the Algebraic Kernel, and solve the system of corresponding equations using the solver provided by the Algebraic Kernel. Finally, the root of the system is used to retrieve the geometric intersection.

3.2 3D Spherical Kernel

To list the main geometric functionalities offered upon instantiation of a model of the 3D Spherical Kernel concept, we discuss in turn the types provided, the predicates and constructions, and the connexion between geometry and algebra.

Types. As seen from Table 3, the types decompose into three groups. The first four types are inherited from the 3D Linear Kernel, whereas the other objects are introduced by the 3D Spherical Kernel itself. Then, the 3D Spherical Kernel concept can be decomposed into two different parts: The first one considers general objects, while the second part introduces specific concepts for objects lying on a given reference sphere—as evidenced by the suffix `ORS` which stands for `OnReferenceSphere`. Note that the concept `ThetaRep`, designed to represent the θ-coordinate of a point, requires no accessor. It must only be comparable through function object `CompareTheta_3`.

Note that a circular arc can be defined by a supporting circle and two endpoints. It is unambiguously defined as the set of points lying on the circle, when walking counterclockwise from its source to its target in the positive plane containing the circle—a plane is positive if its equation is of the form $ax+by+cz+d = 0$ with $(a, b, c) > (0, 0, 0)$ according to the lexicographic order.
Predicates and Constructions. Predicates and constructions are provided by the kernel as C++ functions objects (called functors). Note that a functor can be used as a template argument.



Names in boxes represent 3D Spherical Kernel or Algebraic Kernel functors, while names on dashed arrows are access functions.

Figure 3: An example of an implementation of a geometric construction: computing the intersection of a circle and a plane.

Types	**Basic accessors**
`SK::Point_3`	`x(), y(), z()`
`SK::Line_3`	`point(int), to_vector()`
`SK::Plane_3`	`a(), b(), c(), d()`
`SK::Sphere_3`	`center(), squared_radius()`
`SK::Circle_3`	`center(), squared_radius(),` `supporting_plane(), diametral_sphere()`
`SK::CircularArcPoint_3`	`x(), y(), z()`
`SK::CircularArc_3`	`source(), target(), supporting_circle()`
`SK::LineArc_3`	`source(), target(), supporting_line()`
`SK::CircleORS_3`	`type_of_circle_on_reference_sphere(),` `reference_sphere()`
`SK::CircularArcPointORS_3`	`reference_sphere()`
`SK::CircularArcORS_3` `SK::ThetaRep`	`reference_sphere(), theta_rep()`

Table 3: Types of the 3D Spherical Kernel. The three groups of the table read as follows: inherited from the linear kernel; general objects; objects on a reference sphere. Notice that objects from the third group inherit the functionalities of objects of the second group.

Predicates	**Returns**	**Arg.**	**Meaning**
`SK::CompareX_3`	{<,=,>}	(p_i, p_j)	p_i.x() vs. p_j.x()
`SK::CompareY_3`	{<,=,>}	(p_i, p_j)	p_i.y() vs. p_j.y()
`SK::CompareZ_3`	{<,=,>}	(p_i, p_j)	p_i.z() vs. p_j.z()
`SK::CompareXY_3`	{<,=,>}	(p_i, p_j)	Lexicographic order
`SK::CompareXYZ_3`	{<,=,>}	(p_i, p_j)	Lexicographic order
`SK::Equal_3`	`bool`	(w_i, w_j)	Geometric equality
`SK::HasOn_3`	`bool`	(w_i, w_j)	$w_j \subseteq w_i$
`SK::HasOnBoundedSide`	`bool`	(p, S)	$p \in$ ball bounded by S
`SK::CompareZAtXY_3`	{<,=,>}	(p, P)	p vs. P $(a, b, d \neq 0)$
`SK::CompareYAtXZ_3`	{<,=,>}	(p, P)	p vs. P $(a, c, d \neq 0)$
`SK::CompareXAtYZ_3`	{<,=,>}	(p, P)	p vs. P $(b, c, d \neq 0)$
`SK::DoOverlap_3`	`bool`	(A_i, A_j)	Overlap test
`SK::DoIntersect_3`	`bool`	(w_i, w_j)	Intersection test
`SK::CompareTheta_3`	{<,=,>}	(p_i^r, p_j^r)	Compare θ-coordinates
		(θ_i, θ_j)	Compare θ-values
`SK::CompareThetaZ_3`	{<,=,>}	(p_i^r, p_j^r)	Lexicographic order
`SK::CompareZAtTheta_3`	{<,=,>}	$(A_i^m, A_j^m, \mathbb{M})$	Order along $\mathbb{M}$
	{<,=,>}	(p^r, A_i^m)	p^r vs. A_i^m along $M_\theta \ni p^r$
`SK::CompareZToRight_3`	{<,=,>}	(A_i^m, A_j^m)	Right of common pt order

Notations: p, A, P, $\mathbb{M}$, S, θ, w, stand respectively for a point, a circular arc, a plane, a meridian included in a rational plane, a sphere, a θ-value of type `ThetaRep`, any object of the 3D Spherical Kernel. Two instances of any type are distinguished using a subscript i and j. A superscript m on a circular arc indicates that this arc is θ-monotone—obviously on a reference sphere. A superscript r on a point indicates that such a point is on a reference sphere. Notice that for equality test, w_i and w_j are of the same type.

Table 4: Predicates of the 3D Spherical Kernel.

A model of the 3D Spherical Kernel must provide the functionality described in the sequel. We list only the most important requirements in Table 4 (predicates) and Table 5 (constructions). A construction returns an iterator over a (possibly empty) set of solutions associated to the problem solved by the function.
Communication with Algebra. Functor `SK::GetEquation` returns a polynomial of the Algebraic Kernel providing an equation of a plane or a sphere. As explained at the beginning of Section 3, this is the essential link between the 3D Spherical Kernel and the Algebraic Kernel.

3.3 Algebraic Kernel for Spheres

The operations provided by the CGAL 3D Spherical Kernel make heavy use of algebraic operations. The `AlgebraicKernel` parameter has a crucial role in particular for the robustness of the 3D Spherical Kernel.

Construction	**Arguments**	**Returned objects**
`SK::Intersect_3`	Two or three primitives	Their intersection(s)
`SK::ThetaExtremePoint_3`	A circle or a circular arc, on a reference sphere	Its θ-extremal point(s)
`SK::MakeThetaMonotone_3`	A circle or a circular arc, on a reference sphere	Its θ-monotone sub-arc(s)

Table 5: Constructions of the 3D Spherical Kernel.

Type	**Represents**
`AK::PolynomialForSpheres_2_3`	Equation of a sphere; degree 2, 3 variables
`AK::Polynomial_1_3`	Equation of a plane; degree 1, 3 variables
`AK::RootForSpheres_2_3`	Root of a system of 3 polynomials of the previous types

The `AK::RootForSpheres_2_3` *type is used to represent the coordinates of a point of the 3D Spherical Kernel.*

Table 6: Types of the `AlgebraicKernel`.

Types. As already mentioned in [20], the Algebraic Kernel must provide basic types (polynomials and roots of systems, see Table 6), and basic functionalities on them. In the sequel, we focus on the most important requirements only.

Main Algebraic Predicates and Constructions. Several elementary functors reflecting the 3D Spherical Kernel user interface are provided for predicates, for instance to compare the coordinates of two `AK::RootForSpheres_2_3`. Moreover, the functor `AK::SignAt` allows one to evaluate the sign of a polynomial of type `AK::PolynomialForSpheres_2_3` or `AK::Polynomial_1_3` at a `AK::RootForSpheres_2_3`.

The main construction is provided by the functor `AK::Solve`, which solves a zero-dimensional polynomial system featuring polynomials whose types are listed in Table 6. The return value is an iterator over the solution set (possibly empty), each solution being given as a `AK::RootForSpheres_2_3`.

4 Implementation Details

In this section, we bridge the gap between the geometric calculations presented in Section 2 and the functionalities of the kernel: We describe the way concepts developed in the previous section are implemented.

Operations Involving Points. In the current CGAL implementation, an algebraic number A of degree two is represented as a triple of rationals (a, b, c) such that $A = a + b\sqrt{c}$. By Corollary 1 and Observation 1, the Cartesian coordinates of an intersection point and of a θ-extremal point of a normal circle are algebraic numbers of degree two. Comparison of Cartesian coordinates of such points is easily handled using the CGAL representation. A similar observation holds for the evaluation of a degree two polynomial at such a point—a calculation involved in predicates `AK::SignAt`, `SK::CompareZAtXY_3`, `SK::CompareYAtXZ_3`, `SK::CompareXAtYZ_3`.

Predicates `SK::CompareTheta_3` and `SK::CompareThetaZ_3` compare two θ-coordinates using a partition of the interval $(0, 2\pi]$ into eight open intervals of length $\frac{\pi}{4}$ and eight value $\frac{k\pi}{4}$ with $k = 0 \dots 7$. The comparison of θ-coordinates of points falling in different intervals is trivial, while that of points falling in the same interval requires comparing $\tan\theta$ or $\cot\theta$. This boils down to comparing two algebraic number of degree two—each constructed as a quotient (Observation 2) of two degree two algebraic numbers in the same extension using the x- and y-coordinates of each point. This strategy is valid even for events at a pole, using $(y_i, -x_i, 0)$ and $(-y_i, x_i, 0)$ as fictitious points whose θ-coordinates match the θ-extremal values of a polar or bipolar circle C_{0i}. Indeed, one can see that a plane containing the poles and these two points is tangent to or contains C_{0i}.

Identifying Intersection Points. The intersection points of two circles and the θ-extremal values of a circle come into pairs, so that finding the element of the pair with smallest θ-coordinate is an important primitive. Using the afore-mentioned partition of $(0, 2\pi]$, this is trivial if the two points fall within different intervals. If not, let p and q be two such points. In such a setting, identifying the smallest θ-coordinate, is equivalent to computing $\mathrm{Sign}(p_y q_x - p_x q_y)$. By Observation 1 and Corollary 1, there exist four rational numbers a, b, c, d and a rational polynomial of degree two P such that:

$$\begin{cases} p_x = aR_1 + b \quad p_y = cR_1 + d, \text{ with } P(R_1) = 0 \\ q_x = aR_2 + b \quad q_y = cR_2 + d, \text{ with } P(R_2) = 0 \end{cases} \tag{11}$$

Therefore, $\mathrm{Sign}(p_y q_x - p_x q_y) = \mathrm{Sign}((bc - ad)(R_1 - R_2))$. Since R_1 and R_2 are roots of the same polynomial, upon creation of p and q we only have to evaluate $\mathrm{Sign}(bc - ad)$.

Sorting Circular Arcs at a Common Point. Let p be an intersection point of two θ-monotone circular arcs and suppose that p is on a meridian M_θ. The predicate `SK::CompareZToRight_3` consists of finding the relative position (above, on or below) of the two arcs to the right of p, i.e. for the angle value $\theta + \varepsilon$ with ε arbitrarily small. Notice that the predicate is not defined if point p is a pole, and that the associated circles are of any type but bipolar as no θ-monotone circular arc is defined on such circles. The two supporting circles are denoted C_{0i} and C_{0j}. We now consider three exclusive cases:
– If p matches a θ-extremal point of at least one of the two circles, then this circle is normal. The ordering is trivial resorting to the radii of circles and upper/lower status of θ-monotone circular arcs.
– If the circular arcs intersect transversally, we use the sign of Δ in Eq. (8).
– If the circular arcs are tangent, for $k = i$ or j, we define z_k to be either the z-coordinate of the θ-extremal points of C_{0k} if it is a normal circle, or the z-coordinate of the pole C_{0k} goes through if it is a polar circle. We have two sub-cases:
(i)If one of the two circles is threaded, say C_{0i}, we conclude from the sign of $c_{0i_z} - c_{0j_z}$ if C_{0j} is threaded too, and from the sign of $p_z - z_j$ otherwise.
(ii)If both circles are not threaded, we conclude from the radii of the circles and the sign of $p_z - z_k$, $k = i, j$.
Note that if $p_z - z_k$ is negative (respectively positive), the circular arc of C_{0k} involved is a lower (respectively upper) one.
Ordering θ-monotone Circular Arcs. We consider two problems: ordering two circular arcs intersecting M_θ included in a rational plane, and positioning a point p along a meridian with respect to a circular arc intersected by any given meridian. These two operations are provided by the functor `SK::CompareZAtTheta_3`.

For the first predicate, meridian M_θ is included in a rational plane. We explicitly construct and compare the z-coordinates of the intersection points between the meridian and the circular arcs.
For the second predicate, observe that a non-great circle decomposes S_0 into two regions of unequal areas—the region of largest area and the center of S_0 are on the same side of the plane containing the circle. To begin with, we look for the position of the point p w.r.t. the supporting plane of the circle of the circular arc. This information is sufficient to conclude if p lies on the plane or if the corresponding circle is threaded or polar. When the circular arc lies on a normal circle: If p lies inside the cap of least area defined by the circle on S_0 then p is below the upper circular arc and above the lower one; otherwise the position of p relatively to the circular arc is given by the sign of the difference of p_z and the z-coordinate of the θ-extremal points of the circle.
Misc. For testing equality of two geometric objects, we compare either their parameters or their algebraic representations. Overlapping tests are performed using underlying geometric objects and inclusion of endpoints. Intersection tests and computations are based on classical mathematical inequalities. Predicate `SK::HasOnBoundedSide` obviously relies on evaluation of the sign of sphere equation at the point considered. The inclusion tests performed by predicate `SK::HasOn_3` are easily triggered using the underlying geometric or algebraic representation. For a point on a circular arc, we additionally need some orientation tests.

5 Application: Computing Spherical Arrangements

In this section, we present the connexion between the 3D Spherical Kernel and a Bentley-Ottmann like algorithm [6] computing the exact arrangement of circles on a sphere [14]. In a nutshell, the algorithm takes as input a collection of circles, and returns a decomposition of the sphere into regions whose interiors are connected—the decomposition being stored in an extended half-edge data structure handling holes in faces. In the following, we explain how primitives from the 3D Spherical Kernel are used, and refer the reader to [14] for the details on the algorithm.

Algorithm Description. Given a collection of circles on a reference sphere, they are first decomposed into θ-monotone circular arcs. Then, the algorithm consists of sweeping these arcs using a meridian M_θ anchored at the poles and moving from 0 to 2π using cylindrical coordinates. This process is tantamount to the classical sweep-line process in the plane, as each θ-monotone circular arc is intersected in at most one point by the sweep meridian. An event corresponds to an intersection point (either a transverse intersection or a tangency point) between two circular arcs, or to a θ-extremal point of a circle. Degeneracies occur when several events are associated to the same point of the reference sphere. To handle them, events are gathered into a data structure called the *event site*. The *vertical ordering* $\mathcal{V}$ stores the order of θ-monotone circular arcs along the meridian during the sweep. The *event queue* $\mathcal{E}$ stores event sites, which are created upon insertion of intersection and θ-extremal events.

Connexion between the Algorithm and the 3D Spherical Kernel. The algorithm features three main constructions. The first two correspond to the construction of θ-extremal points and the creation of θ-monotone circular arcs from input circles using `SK::ThetaExtremePoint_3` and `SK::MakeThetaMonotone_3` respectively. The third one is concerned with the construction of intersection points and is achieved using `SK::Intersect_3`.

Predicates are involved in the manipulation of $\mathcal{V}$ and $\mathcal{E}$. Let us first examine the vertical ordering $\mathcal{V}$: (i)The initialization of $\mathcal{V}$ requires comparing the position of two circular arcs along meridian M_θ with $\theta = 0$, using predicate `SK::CompareZAtTheta_3`. For intersections occurring on M_0, one further needs to compare the circular arcs to the right of this point using predicate `SK::CompareZToRight_3`.
(ii)To update $\mathcal{V}$, the only predicate involved is that required to insert the circular arcs of a normal circle starting to be swept at a given θ-extremal point p. To do so, we locate p along the meridian amongst circular arcs present in $\mathcal{V}$, the predicate involved being `SK::CompareZAtTheta_3` with one point and one circular arc. The case of several circles having p as θ-extremal point is easily handled once the positions of the upper and lower arcs of the circle of greatest radius have been determined—this a pencil of circles tangent at p.
(iii) To maintain a linear size event queue [9], within an event site, only intersection events corresponding to a pair of circular arcs adjacent along $\mathcal{V}$ are stored. This also prevents re-inserting a given event—which is harmful since such a re-insertion causes arithmetic filter failures while seeking this event in the queue. Finally, concatenating appropriately blocks of arcs involved within an event site allows one to maintain $\mathcal{V}$ without any numerical operation [14].

Let us now analyze the event queue $\mathcal{E}$. Its initialization requires all θ-extremal points. Given all but threaded circles using access function of circles `type_of_circle_on_reference_sphere_3()`, such points are constructed using functor `SK::ThetaExtremePoint_3`. The detection of new intersection points from new adjacencies along $\mathcal{V}$ uses predicate `SK::DoIntersect_3`. All intersection and θ-extremal points are sorted using `SK::CompareTheta_3` and `SK::CompareZ_3`.

The algorithm is summarized in Table 7, the primitives pulled out from the 3D Spherical Kernel being typeset in typewriter font. From this algorithm, the construction of the half-edge data structure storing the arrangement uses two Union-Find processes, as explained in [14].

```
0. Classify circles as normal/polar/bipolar/threaded.
     ♦SK::CircleORS_3::type_of_circle_on_reference_sphere_3()
   Compute $\theta$-extremal points and decompose circles into $\theta$-monotone arcs.
     ♦SK::ThetaExtremePoint_3, SK::MakeThetaMonotone_3
1. Initialize $\mathcal{V}$: Fill $\mathcal{V}$ with circular arcs intersected by the meridian at $\theta = 0$.
     ♦SK::CompareZAtTheta_3, SK::CompareZToRight_3
2. Initialize $\mathcal{E}$:
   (a) Look for intersections between circular arcs adjacent in $\mathcal{V}$ at $\theta = 0$,
       ♦SK::DoIntersect_3
       and insert the corresponding intersection points into $\mathcal{E}$.
       ♦SK::Intersect_3, SK::CompareTheta_3, SK::CompareZ_3
   (b) For all but threaded circles, insert $\theta$-extremal points into $\mathcal{E}$.
       ♦SK::CompareTheta_3, SK::CompareZ_3
3. While $\mathcal{E}$ is not empty do
   (a) Remove from $\mathcal{V}$ the circular arcs of the event(s) ending.
   (b) Insert into $\mathcal{V}$ the circular arcs of the event(s) starting.
       ♦SK::CompareZAtTheta_3
   (c) Swap in $\mathcal{V}$ the circular arcs intersecting transversally at the event.
   (d) Insert into $\mathcal{E}$ the intersection detected from the new adjacencies along $\mathcal{V}$.
       ♦SK::DoIntersect_3, SK::Intersect_3, SK::CompareThetaZ_3
```

Table 7: The Bentley-Ottmann algorithm for circles on a sphere, together with the primitives of the 3D Spherical Kernel involved.

6 Experiments on Spherical Arrangements

In this section we present two types of results for the algorithm of section 5: On the one hand, we investigate the practical complexity of the algorithm, and on the other hand, we compare to previous work using molecular models.

Variants. The arrangement algorithm being implemented as a generic C++ class, we investigate the following three variants:
– The *double variant* instantiates the algorithm using a plain `double` number type as rational type. The main interest is to estimate the overhead imposed by the certification of the results. No guarantee is provided either on the termination or on the correctness of the arrangement.
– The *exact variant* instantiates the algorithm using `CGAL::Lazy_Exact_NT<Gmpq>` as rational type. This is a filtered version of `Gmpq` [26] using the double interval type `CGAL::Interval_nt`. It uses the `Gmpq` exact rational number type when intervals are not sufficient to certify the answer to a predicate. This number type stores for each number an approximated value and the exact value encoded as a DAG of the arithmetic operations needed to construct it—the value is computed upon request.
– The *exact filtered variant* instantiates the algorithm twice, respectively with `CGAL::Interval_nt` and `CGAL::Lazy_Exact_NT<Gmpq>`, these instantiations being articulated as follows. We first launch the instantiation based upon `CGAL::Interval_nt`. If the certification of a predicate fails, we restart the whole arrangement calculation using the instantiation resorting to `CGAL::Lazy_Exact_NT<Gmpq>`. Notice that while the standard filtering strategy consists of re-computing a quantity at the predicate level, we do the same but for a whole arrangement. Notice also that this strategy makes sense if the first calculation fails with low probability.
Running Times of an Arrangement of Circles. As shown in [14], the Bentley-Ottmann algorithm on a sphere has complexity $c(n + k + l) \log n$, with n the number of circles, k the number of intersection points, and l the number of faces bounded by exactly two arcs ($l = O(k)$ in non-degenerate cases). To scale the running time with the theoretical complexity, we wish to measure to constant c of the algorithm. To investigate this question, tests were run on collections of random circles on the 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. The number of circles chosen varies from 10 to 400 by step size of 10. Computations were run on a bi-proc Pentium IV(R) 3.06Ghz with 2.5GB of RAM.

We performed a curve fitting on running times. As illustrated on Fig. 4, the theoretical curve is in excellent agreement with the experimental curves, for the double and exact variants. Setting the constant c to match the experimental time obtained for $n = 400$ and the measured k and l, we observe that the constant factor is 2.1×10^{-5} for the double variant and 10.2×10^{-5} for the exact variant.

Running Times on Molecular Models. Given a molecular model, we wish to compute one arrangement for each atom—on its surface. This arrangement features the circles defined by the intersections between this atom and its neighbors. These neighbors are retrieved using a regular grid partitioning the axis aligned bounding box of the model into *cells*. The length of each edge in the grid is the diameter of the largest sphere of the model. Each atomic sphere is associated a cell in the grid and all the spheres intersecting it are either in the same cell or in adjacent ones.

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 controlled perturbations of spheres [22]. Because the perturbation used is global, the running times reported correspond to the cumulative cost of the arrangements on all spheres. Practically, four protein models from the Protein Data Bank [38] were used—PDB codes: `1bzm`, `1jky`, `7at1`, `1l7x`. Table 8 features our results on these models, using a Pentium III(R) 1Ghz with 1GB of RAM in our case, as opposed to a bi-proc Pentium III(R) 1GHz with 2GB of RAM for [22]. On these examples, our code is about 65% faster using the double variant and 20% slower using the exact filtered variant.

Consider Tables 8 and 9. 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. 5. Failures also happen for molecular models, since 6 atoms had to be removed to get the *double* row in Table 9.

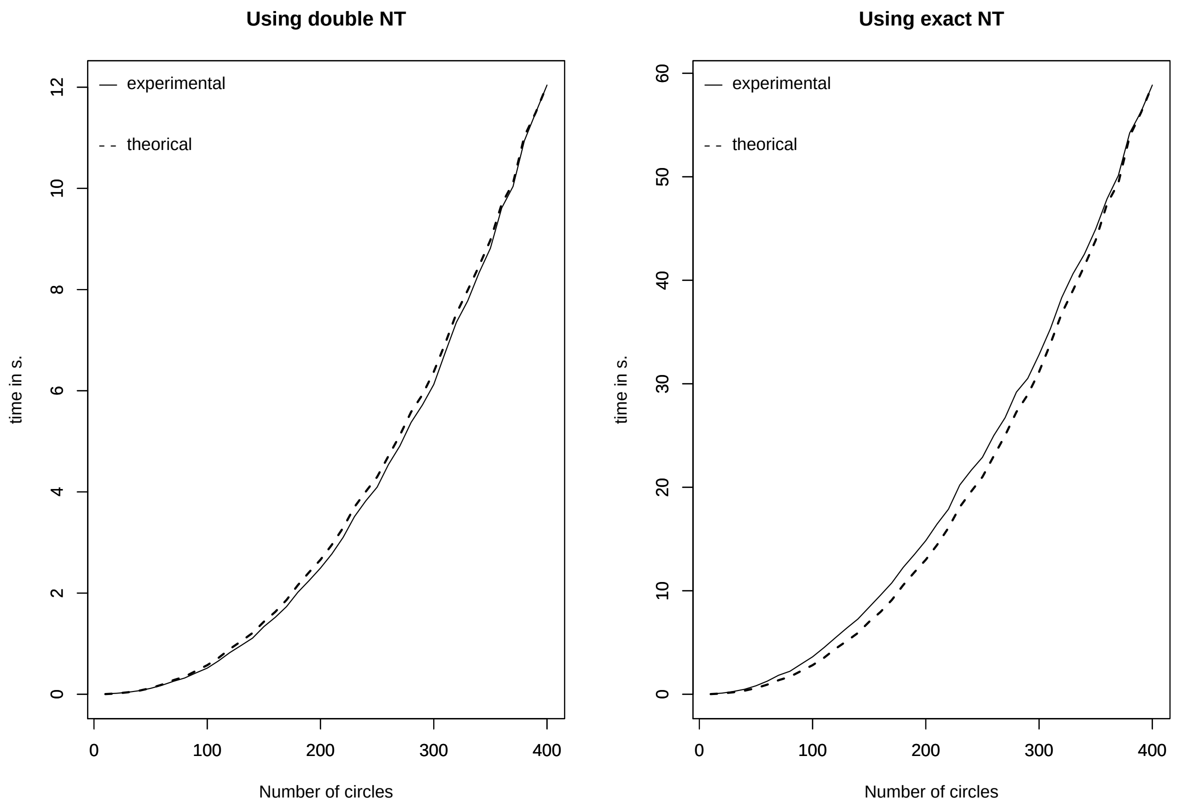


Figure 4: Observed vs theoretical complexities for random arrangements.

Input file	#atoms	From [22]	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

Table 8: Tests on 4 macro-molecular structures. Total time (in seconds) for computing the Van der Waals surface, including the perturbation, taken from [22] vs. total time of computing the arrangement of circles on each atomic sphere for the three variants.

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 9: 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 faces on each atomic sphere.

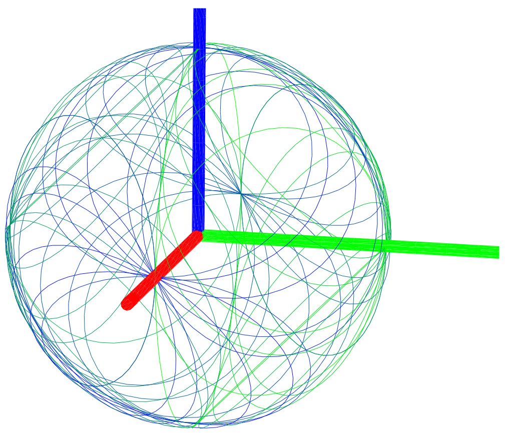
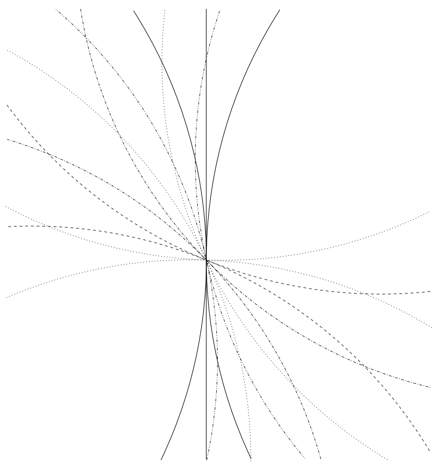


Figure 5: Left: degenerate arrangement of 47 circles on S_0. Right: zoom about the point $(r_0, 0, 0)$: six different directions of tangency for thirteen circles. This point is a θ-extremal point for two circles. Ten circles (two among them being polar) intersect at that point and a bipolar circle passes through it. This arrangement features 674 vertices, 1384 edges, 712 faces and no holes. Euler characteristic reads as $674 - 1384 + 712 = 2$.

7 Conclusion

High quality geometric code relies on four virtues: robustness, efficiency, modularity, re-usability. Although spheres are amongst the most elementary geometric objects, no library of essential primitives was available to deal with them. This paper answers this need, by developing the CGAL 3D Spherical Kernel concept, i.e. a concept featuring the basic types and operations required to deal with spheres, planes, circles, circular arcs and points in 3D. A clear distinction is made between the algebraic and the geometric aspects on the one hand, and on the concepts of a kernel and its implementation on the other hand. The concept is accompanied by an implementation. This implementation, together with a generalization of the Bentley-Ottmann algorithm on a sphere, provides the first effective solution to the problem of computing the exact arrangement of circles on a sphere. Applications in structural biology to investigate protein-protein and protein-drugs interfaces are being developed.

Acknowledgments

The authors wish to thank Sylvain Pion for helpful discussions.

References

[1] P. Agarwal, L. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision detection for deforming necklaces. *Computational Geometry: Theory and Applications*, 28:137–163, 2004.

[2] O. Aichholzer, F. Aurenhammer, T. Hackl, B. Jüttler, M. Oberneder, and Z. Sir. Computational and structural advantages of circular boundary representation. In *Proc. 10th International Workshop on Algorithms and Data Structures*, volume 4619 of *Lecture Notes Comput. Sci.*, pages 374–385, 2007.

[3] N. Amenta, S. Choi, and R. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry : Theory and Applications*, 19(2):127–153, 2001.

[4] M. V. A. Andrade and J. Stolfi. Exact algorithms for circles on the sphere. *International Journal of Computational Geometry and Applications*, 11:267–290, 2001.

[5] D. Avis, B. K. Bhattacharya, and H. Imai. Computing the volume of the union of spheres. *The Visual Computer*, 3(6):323–328, 1988.

[6] J. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 28(9):643–647, 1979.

[7] E. Berberich, E. Fogel, D. Halperin, K. Mehlhorn, and R. Wein. Sweeping and maintaining two-dimensional arrangements on surfaces: A first step. In *Proc. 15th European Symposium on Algorithms*, volume 4698 of *Lecture Notes Comput. Sci.*, pages 645–656, 2007.

[8] 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. *Proc. 21st Annual Symposium on Computational Geometry*, pages 99–106, 2005.

[9] K. Q. Brown. Comments on "Algorithms for reporting and counting geometric intersections". *IEEE Transactions on Computers*, 30(2):147–148, 1981.

[10] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. In *Proc. 9th European Symposium on Algorithms*, volume 2161 of *Lecture Notes Comput. Sci.*, pages 254–265, 2001.

[11] P. M. M. de Castro, F. Cazals, S. Loriot, and M. Teillaud. Design of the CGAL spherical kernel and application to arrangements of circles on a sphere. Research Report 6298, INRIA, 2007. `https://hal.inria.fr/inria-00173124`.

[12] P. M. M. de Castro, S. Pion, and M. Teillaud. Exact and efficient computations on circles in CGAL. In *Abstracts 23rd European Workshop on Computational Geometry*, pages 219–222. Technische Universität Graz, Austria, 2007. Full version available as INRIA Research Report 6091, Exact and efficient computations on circles in CGAL and applications to VLSI design, `https://hal.inria.fr/inria-00123259`.

[13] P. M. M. de Castro and M. Teillaud. 3D spherical kernel. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.4 edition, 2008. To appear.

[14] F. Cazals and S. Loriot. Computing the exact arrangement of circles on a sphere, with applications in structural biology. Research Report 6049, INRIA, 2006. `https://hal.inria.fr/inria-00118781`.

[15] Cgal, Computational Geometry Algorithms Library.
`http://www.cgal.org`.

[16] F. Chazal, D. Cohen-Steiner, and Q. Mérigot. Stability of boundary measure. Research Report 6219, INRIA, 2007. `http://hal.inria.fr/inria-00154798`.

[17] M. L. Connolly. Molecular surfaces: a review. *Network Science*, 14, 1996. `http://www.netsci.org/Science/Compchem/feature14.html`.

[18] Core Number Library.
`http://cs.nyu.edu/exact/core_pages`.

[19] R. S. Drysdale, G. Rote, and A. Sturm. Approximation of an open polygonal curve with a minimum number of circular arcs and biarcs. *Computational Geometry : Theory and Applications*, 41:31–47, 2008.

[20] I. Z. Emiris, A. Kakargias, S. Pion, M. Teillaud, and E. P. Tsigaridas. Towards an open curved kernel. In *Proc. 20th Annual Symposium on Computational Geometry*, pages 438–446, 2004.

[21] Exacus, Efficient and Exact Algorithms for Curves and Surfaces.
`http://www.mpi-inf.mpg.de/projects/EXACUS`.

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

[23] A. Fabri and S. Pion. A generic lazy evaluation scheme for exact geometric computations. In *Proc. 2nd Library-Centric Software Design*, pages 75–84, 2006.

[24] E. Fogel and M. Teillaud. Generic programming and the CGAL library. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 313–320. Springer-Verlag, Mathematics and Visualization, 2006.

[25] S. Funke and K. Mehlhorn. LOOK: A Lazy Object-Oriented Kernel for geometric computation. In *Proc. 16th Annual Symposium on Computational Geometry*, pages 156–165, 2000.

[26] GMP, GNU Multiple Precision Arithmetic Library.
`http://www.swox.com/gmp`.

[27] S. Hert, M. Hoffmann, L. Kettner, S. Pion, and M. Seel. An adaptable and extensible geometry kernel. *Computational Geometry : Theory and Applications*, 38:16–36, 2007.

[28] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. K. Yap. Classroom examples of robustness problems in geometric computations. *Computational Geometry : Theory and Applications*, 40:61–78, 2008.

[29] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. MAPC: a library for efficient and exact manipulation of algebraic points and curves. In *Proc. 15th Annual Symposium on Computational Geometry*, pages 360–369, 1999.

[30] Leda, Library for Efficient Data Types and Algorithms.
`http://www.algorithmic-solutions.com/enleda.htm`.

[31] C. Li, S. Pion, and C. K. Yap. Recent progress in exact geometric computation. *Journal of Logic and Algebraic Programming*, 64(1):85–111, July 2005. Special issue on the practical development of exact real number computation.

[32] C. Li and C. K. Yap. A new constructive root bound for algebraic expressions. In *Proc. 12th Symposium on Discrete Algorithms*, pages 496–505, 2001.

[33] B. J. McConkey, V. Sobolev, and M. Edelman. Quantification of protein surfaces, volumes and atom-atom contacts using a constrained Voronoi procedure. *Bioinformatics*, 18(10):1365–1373, 2002.

[34] B. Mourrain, J.-P. Técourt, and M. 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.

[35] S. Pion and M. Teillaud. 2D circular kernel. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.2 and 3.3 edition, 2006 and 2007. `http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/packages.html#Pkg:CircularKernel2`.

[36] S. Rusinkiewicz and M. Levoy. QSplat: a multiresolution point rendering system for large meshes. *Proc. 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 343–352, 2000.

[37] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. 2D arrangements. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.2 and 3.3 edition, 2006 and 2007. `http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/packages.html#Pkg:Arrangement2`.

[38] Worldwide Protein Data Bank.
`http://www.wwpdb.org`.

[39] C. K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition, 1995.