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

Pedro M. M. de Castro, Frédéric Cazals, Sébastien Loriot, Monique Teillaud

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

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

N° 6298

Septembre 2007

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--6298--FR+ENG

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 †

Thème SYM — Systèmes symboliques
Projet Geometrica

Rapport de recherche n° 6298 — Septembre 2007 — 46 pages

Abstract: This paper presents a CGAL kernel for algorithms manipulating 3D spheres, circles, and circular arcs. The paper makes three contributions. First, the design of the kernel concept is developed, following the best practices for the design of kernels geared towards curved objects. Second, 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 are located on a reference sphere. In both cases, the mathematical derivations for predicates and constructions are detailed. Third, an application to the construction of the exact arrangement of circles on a sphere is overviewed.

Key-words: Robustness, Curved objects, Generic programming, Spheres, Predicates, Constructions, Geometric kernels, CGAL

* Pedro.Machado@sophia.inria.fr
† firstname.name@sophia.inria.fr

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

Architecture du noyau 3D Spherical Kernel CGAL et applications aux arrangements de cercles sur une sphère

Résumé : Ce travail décrit un nouveau noyau CGAL conçu pour les algorithmes manipulant des sphères, des cercles, et des arcs de cercles en $3D$. Trois contributions sont présentées. Premièrement, on définit l'architecture du concept du noyau en se référant aux principes propres à la conception de noyaux pour objets courbes. Deuxièmement, on montre comment combiner deux cadres: l'un pour le cas général; l'autre dans le cas où tous les objets manipulés sont sur une même sphère, dite de référence. Dans les deux cas, les bases mathématiques sur lesquelles s'appuient les prédicats et les constructions sont décrites. Troisièmement, nous illustrons l'utilisation du noyau pour le calcul de l'arrangement exact de cercles sur une sphère.

Mots-clés : Robustesse, Objects courbes, Programmation Générique, Sphères, Prédicats, Constructions, Noyaux geométriques, CGAL

1 Introduction

Linear versus curved primitives in computational geometry. Geometric algorithms traditionally limit their framework to linear objects in affine Euclidean space (points, segments, triangles, ...), curved objects being discretized by linear elements. On the other hand, handling directly curved objects allows one to skip this discretization process, and also yields algorithms with better combinatorial complexity, as the number of primitives involved is usually much smaller. For these reasons, the direct manipulations of curved objects is an important challenge in computational geometry.

In the realm of curved objects, the simplest primitives are circles and spheres. These primitives are used in various domains, either for representing objects or for approximating them. Manipulations of circular arcs in the plane are of capital importance in some industrial domains like manufacturing and printed or integrated circuit design [18], and there is a recent increasing interest in the computational geometry community for approximations of curves by circular arcs due to their interesting properties [20, 3]. Spheres are also central in molecular modeling.

Spheres and applications in molecular modeling. Two of the most widely used models in molecular modeling are the Van der Waals (VdW) model, where each atom is represented by a ball whose radius depends on the atom type and its chemical environment, and the Solvent Accessible model, where each VdW radius is enlarged by the radius of a water molecule to account for a continuous solvation layer. Such models are ubiquitous in structural studies. For example, the so-called buried surface area [16], derived from the solvent accessible surface (SAS), is one of the main parameters allowing one to classify protein-protein complexes [12]; the SAS allows a simple and physically founded classification of amino-acids in terms of hydrophobicity [38]; solvent accessible surfaces and volumes are instrumental in deriving hydrophobic force fields [30]; surface related quantities have been used to define scoring functions [1]. Apart from these molecular surfaces related quantities, geometric objects underlying collections of balls also proved important to describe VdW models. In particular, α-shapes [22] have been used to describe pockets within macro-molecules [21].

Interestingly, from a geometric standpoint, the previous constructions essentially focused on the boundary of a union of balls, and on the interaction between at most 4 balls —an intrinsic property of α-shapes. In order to go beyond these traditional models, we refined surface area models —see the companion paper [11] for statistics on the so-called ESBI model, which called for the development of an algorithm computing the exact arrangement of circles on a sphere. This arrangement is being used to investigate properties of ligand-binding pockets [2], and to select conformers in the realm of protein flexibility [5]. The development of this algorithm triggered that of specific predicates for circles lying on a reference sphere, as presented in this paper.

From algorithms to implementations. In dealing with curved objects, previous work focused on the design of important functionalities, and on the development of robust, efficient and flexible code, geared towards applications in academia and industry.

Andrade and Stolfi proposed selected exact manipulations of 3D circular objects lying on a given sphere, with an implementation in Modula-3 [4]. Regarding code development, MAPC was probably the first geometric library handling general algebraic curves, but it seems that it did not handle all degenerate cases [32]. The EXACUS prototype library allows to exactly compute arrangements of conics and cubics in the plane [24]. The CGAL arrangement package can compute an arrangement of circular arcs, conics or Bézier curves [39]. CGAL now offers a kernel of exact basic manipulations of circular arcs in the plane [37].

Contributions and paper overview. This work provides the first complete and efficient implementation of essential basic functionalities to manipulate spheres, circles and circular arcs in 3D, as well as their intersections, in an exact way. Following best practices, a clear distinction is made between the algebraic and the geometric aspects on one hand, and on the concepts of a kernel and its implementation on the other hand. The concept is accompanied by an implementation, which is used to compute the exact arrangement of circles on a sphere. As a quality label, this package has been submitted to the CGAL Editorial board and is currently under review.

The paper is organized as follows. Section 2 puts the 3D Spherical Kernel in context with respect to best practices in geometric software development, and with respect to CGAL in particular. The 3D Spherical Kernel design and an extension to the case where all objects manipulated lie on a reference sphere are discussed in sections 3 and 4. An application to compute the exact arrangement of circles on a sphere is presented in section 5. Section 6 develops the mathematical foundations of the predicates and constructions provided by the kernel. Finally, section 7 gives an overview of the proposed implementation of the concepts presented earlier.

2 Geometric programming and CGAL

2.1 CGAL

The goal of the CGAL project is to promote research in computational geometry, so as to make reference algorithms available for academic research, and also to translate results into robust programs for industrial applications [13]. The project formally started in 1996 under the initiative of a consortium of a several teams in Europe and Israel —including our group, and was supported by the European Community for three years. It evolved to an *Open Source* project, allowing other researchers to participate. GEOMETRYFACTORY [1], a start-up launched in January 2003, is selling commercial licenses and supports specific developments based on CGAL.

[1] `http://www.geometryfactory.com/`

At the same period, the *Computational Geometry Impact Task Force Report*, coordinated by Bernard Chazelle, insisted on a few recommendations [14, 15]. Two of them were the production of useful and usable geometric software, and the need for the creation of a rewarding structure for implementations in the academic world. CGAL is striving to meet these two recommendations.

CGAL has high quality standards, which are ensured by a number of different means. First, an editorial board, created in 2001 and currently consisting of 12 members, is in charge of making technical decisions and coordinating CGAL promotion. The committee evaluates proposals of new packages, and manages a review process similar to that of papers submitted to journals. This process guarantees the coherence, homogeneity and quality of the library, and is also rewarding for authors whose packages are getting integrated within CGAL. Second, a detailed documentation featuring a user and a reference manual for each package is made available —over 3000 pages for 57 packages as of July 2007. Third, tests are run every night on internal versions so as to avoid code regressions.

2.2 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'*). Implementing geometric algorithms is notoriously difficult, especially because of numerical issues. Although geometric algorithms are basically of combinatorial and discrete nature, the branching decisions are based upon continuous predicate evaluations, subject to rounding errors when floating point arithmetic is used, which often produces inconsistencies [31].

To get around these difficulties, Yap pioneered the Exact Geometric Computation paradigm [40], which we follow for our kernel design. In this paradigm, predicates are evaluated in an exact way, which guarantees that algorithms relying on them do not fail.

Some predicates may seem to require constructions: for instance, the classical Bentley-Ottmann algorithms for computing an arrangement of input line segments [6] requires comparing the x-coordinates of two intersection points of line segments. Note however that these intersections need not be actually constructed. Such comparisons can indeed be performed by functions of the input line segments, and boil down to evaluations of signs of polynomial expressions in the input data [9], thus avoiding rounding errors inherent to the computation of the coordinates of the intersection points.

Still, evaluating a predicate requires constructing numbers involved in its evaluation, these numbers being functions of the input geometric objects. Instead of evaluating the predicate from the input data each time an object is involved in a decision, we may want to store selected constructed numbers for later use. In that case, a predicate may compare

the x-coordinates of two intersection points using these constructed numbers instead of performing a calculation directly involving the coefficients of the line segments.

The previous discussion is of special interest for curved algebraic objects where the two alternatives correspond to two philosophies to evaluate exact predicates.

On one hand, using an implicit encoding of an algebraic number, by some polynomial vanishing on it, corresponds to an evaluation strategy resorting to resultant-like calculations on the input polynomials. For instance, comparing the x-coordinates of two intersection points of circles again boils down to simply evaluating signs of polynomial expressions in the input data [19]. Though these coordinates are algebraic numbers, computations of square roots are avoided.

On the other hand, constructing effectively the algebraic numbers by relying on specific number types like `Core::Expr` or `leda::real` [17, 33], that accommodate exact computations on algebraic numbers, requires in the end running multi-precisions calculations down to separation bounds in the worst cases [10, 34]. But in doing so, one reuses intermediate calculations, and the design of predicates, a difficult task for algebraic objects, is made easier.

An important point must be emphasized here: evaluating predicates, expressed as signs of polynomial expressions, in an exact way, does not necessarily subsume an expensive exact arithmetic. Indeed, one only requires a sufficient precision to make the correct predicate evaluation. This is the fundamental idea that led to the development of many arithmetic filtering techniques [35]. In a nutshell, computations are carried out using a fast number type that supports only inexact arithmetic operations, typically floating point numbers, and are accompanied by an upper bound evaluation on the error made. This error, which may be pre-computed for static filters, or may be computed on the fly using interval arithmetic, allows one to certify the predicate evaluation in most of the cases. Only if certification fails, the computation is re-launched using an exact arithmetic.

We may also mention that future work on filtered constructions [27, 25] may reconcile the evaluation of predicates and constructions. The implementation of such a framework still needs to be worked out to surpass the approach relying on predicates only.

2.3 CGAL Kernels and Traits Classes

C++ generic programming identifies an abstraction making extensive use of C++ class-templates and function-templates. It is a formal hierarchy of abstract requirements on data types referred to as *concepts*, and a set of classes that conform precisely to the specified requirements, referred to as *models* [28].[2] As detailed in [26], the CGAL library follows the

[2] Let us mention here the CGAL naming scheme that will also be used in this paper:
- Words in the names of everything except concepts should be separated by underscores. For example, one would use function_name or Class_name instead of functionName or Classname.
- Words in the names of concepts (e.g., template parameters) should be separated using capital letters. The only use of underscores in concept names is before the dimension suffix. For example, one should use a name such as ConvexHullTraits_2 for the concept in contrast to Convex_hull_traits_2 for the name of the class that is a model of this concept.

generic programming paradigm: the geometric algorithms are generic, and the geometric classes are instantiated with a *Traits* class, a traits class being a class offering the minimum set of functionalities required by a specific class. Such a template parameter is supposed to fulfill a set of requirements, described and documented as a concept.

Let us mention here that the requirements listed in a Traits concept are only syntactical: they list a set of functionalities, with precise signatures, that must be made available to some geometric class so that its algorithms can run. However, the concept does not assume that these functionalities must be exact; some models may rely on inexact floating point arithmetic for instance. CGAL algorithms are guaranteed to return the correct result if an exact traits class is provided, otherwise they may crash.[3]

In the CGAL library, constant size geometric primitive objects, together with general purpose predicates and constructions on them are made available in *kernels*. CGAL currently offers kernels for linear objects (points, segments, lines, triangles...), and the first version of a kernel for circles and circular arcs in 2D was released in CGAL 3.2 [37]. As kernels can be directly used as traits classes by several CGAL classes, CGAL kernels are documented as C++ concepts. A kernel can also be wrapped into a traits class offering the interface for a selected CGAL class.

The fact that the distinction between predicates and constructions is ambiguous is of no difficulty in designing a kernel concept, as one only needs to make constructions and predicates coherent. On our example above, given two intersecting line-segments, a predicate may indeed be passed the constructed intersection points, or an implicit encoding of its coordinates in terms of input data.

In the case of algebraic objects, again, in designing a kernel concept, both implementation options remain open.

3 Global Software Design

In this section, we present the design perspective of CGAL kernels in general, with an application to the $3D$ spherical kernel.

3.1 Extensibility of CGAL kernels

The general design of a kernel concept for curved objects proposed in [23] is driven by the following concerns:

- interoperability of any model of the kernel concept with CGAL geometric algorithms,
- re-usability of the existing CGAL kernel models for linear objects,
- genericity and flexibility: ability to use other linear kernels than the CGAL ones, and independence from a particular implementation of the algebraic operations needed.

[3] See also "The CGAL Philosophy" on `http://www.cgal.org/`.

The extensibility and adaptability of the CGAL kernel design [29] allows to answer all these concerns. The declaration of a kernel for curved objects is the following:

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

The 2D Circular kernel follows such a design [37]. The geometric level interface provides types already defined by the linear kernel, plus three new types: types for points on circles, circular arcs, and line segments whose endpoints are two points of this new point type. The 3D Spherical Kernel also follows this scheme.

The `LinearKernel` concept will not be presented here. It coincides with the basic CGAL Kernel concept extensively documented in the CGAL manual. The main functionalities of the user interface of the 3D Spherical Kernel is detailed in Section 3.2. The requirements on the `AlgebraicKernel` concept listed in Section 3.3 are guided by the functionalities of the 3D Spherical Kernel that we want to provide the user with. In fact these requirements also describe the user interface of the algebraic kernel. Finally Section 3.4 explains how the algebraic and the geometric layers communicate.

3.2 User Interface

Here we give a summary of the main geometric functionalities available to the user after instantiation of a model of the 3D Spherical Kernel concept.

In the sequel, 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 mention here that the representation data of all objects of the linear kernel (coordinates for points, coefficients of equations for lines, planes, spheres) is supposed to lie in a field number type (i.e., a type providing elementary operations $+, -, \times, /$), that will typically be the rationals. In the sequel, for simplicity, we will always refer to the basic number type as *rational*, and we will say that points, planes, spheres whose equations have rational coefficients are rational.

Types

In a way similar to what the 2D Circular kernel does, the 3D Spherical Kernel presented in this paper extends the 3D linear kernel by

- providing in its interface types taken from the linear kernel, such as
  - `SK::Point_3` that represents rational 3D points;
  - `SK::Line_3` that represents rational lines in 3D;
  - `SK::Plane_3` that represents rational planes in 3D;
  - `SK::Sphere_3` that represents rational spheres;
- defining new basic objects:
  - `SK::Circle_3` that represents circles in 3D;
  - `SK::CircularArcPoint_3` that represents points on 3D circles;

– `SK::CircularArc_3` that represents circular arcs delimited by two `SK::CircularArcPoint_3`;
– `SK::LineArc_3` that represents line-segments delimited by two `SK::CircularArcPoint_3`.

The 3D Spherical Kernel also provides several predicates and constructions, defined on objects of those types, as functors.[4]

Access functions.
We list in the following what we call *access functions*, that must be provided as class member functions.

Note that access functions do not assume any specific representation of the objects: specific models may store objects in such a way that access functions are simply accessing data members, but other models may perform computations to return the result.

– `SK::Circle_3`
  –`center()` returns the center of the circle;
  –`squared_radius()` returns the squared radius of the circle;
  –`supporting_plane()` returns the plane containing the circle;
  –`diametrial_sphere()` returns the diametrical sphere defined by the circle;
– `SK::CircularArcPoint_3`
  – `x()`, `y()` or `z()` returns the x,y or z coordinate of the point;
– `SK::LineArc_3`
  –`source()`,`target()` return the endpoints (as `SK::CircularArcPoint_3`s) of the segment;
  –`supporting_line()` returns the line the segment is drawn on;
– `SK::CircularArc_3`
  –`source()`,`target()` returns the endpoints of a circular arc, as `SK::CircularArcPoint_3`;
  –`supporting_circle()` returns the circle circular arc is drawn on.

Note that a circular arc can be given 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[5] containing the circle.

Predicates.
– Comparing coordinates:
Functors `SK::CompareX_3`, `SK::CompareY_3`, `SK::CompareZ_3` compare Cartesian coordinates of two points of type `SK::CircularArcPoint_3`;
Functors `SK::CompareXY_3` and `SK::CompareXYZ_3` provide a lexicographic comparison;
The value returned by these functors belongs to $\{<, =, >\}$;

[4] Recall that a *functor* is a class object that can be used as a function by defining the `operator()`. Main advantages of such an implementation are that we can inline the underlying methods and store variables in the class —to take into account a current state for example.

[5] In this definition, we say that a plane is positive if its equation is of the form $ax + by + cz + d = 0$ with $(a, b, c) > (0, 0, 0)$ (i.e. $(a > 0)$ or $((a == 0)$ and $(b > 0))$ or $((a == 0)$ and $(b == 0)$ and $(c > 0))$).

– Testing equality:
Functor `SK::Equal_3` tests whether any two objects of the 3D Spherical Kernel, of the same type, are equal;
– Testing relative positioning:
Functor `SK::HasOn_3` tests whether one primitive among plane, sphere, line, segment, circle and circular arc lies on a second one;
Functor `SK::HasOnBoundedSide` that tests whether a point lies inside the ball associated with a sphere;
Functor `SK::CompareZAtXY_3` computes whether a point lies below, on or above a non vertical plane —and similarly for `SK::CompareYAtXZ_3` and `SK::CompareXAtYZ_3`;
Functor `SK::DoOverlap_3` tests whether two segments or two circular arcs overlap, that is to say their intersection is neither empty nor reduced to one or two points;
Functor `SK::DoIntersect_3` tests whether the primitives intersect.

Construction.
– Functor `SK::Intersect_3` returns an iterator over intersections (possibly none) of two or three primitives.

Communication to algebra.
– Functor `SK::GetEquation` returns a polynomial of the algebraic kernel providing an equation of a plane, sphere, line or circle.

3.3 Algebraic kernel

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.

As mentioned already in [23], the algebraic kernel must provide basic types (polynomials and roots of systems), and basic functionalities on them (solving polynomial systems, comparing roots of systems, and computing the sign of a polynomial at the roots of a system).

Let us now list the most important requirements forming the `AlgebraicKernel` concept for manipulating spheres, circles and circular arcs in 3D:

Types
– Polynomial types to store the equations of geometric objects, basically: a special type of polynomial of degree two in three variables, `AK::PolynomialForSpheres_2_3` that represents spheres, and a degree one polynomial in three variables, `AK::Polynomial_1_3` that represents planes;
– Type to store roots of systems of three trivariate polynomials of the types just listed above. This type `AK::RootForSpheres_2_3` is used to encapsulate and abstract the algebraic representation of points on 3D circles.

Main algebraic predicates:
– Several functors are provided to compare the Cartesian coordinates of an `AK::RootForSpheres_2_3`: they reflect the 3D Spherical Kernel user interface;
– Functor `AK::SignAt` computes the sign[6] of an `AK::PolynomialForSpheres_2_3` or of an `AK::Polynomial_1_3` at an `AK::RootForSpheres_2_3`. Basically, it will be used to compute the position of a point wrt a sphere or plane.

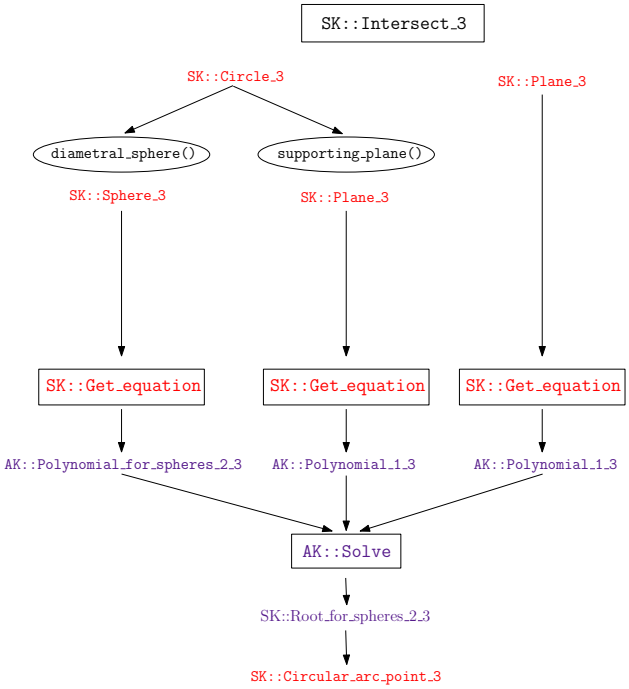
Main algebraic construction:
– Functor `AK::Solve` solves a system of equations defining a zero-dimensional system, i.e. it returns a (possibly empty) iterator over the solutions, each specified by a `AK::RootForSpheres_2_3`. For example, such systems are defined by two spheres and one plane (two `AK::PolynomialForSpheres_2_3` and one `AK::Polynomial_1_3`), by three spheres (three `AK::PolynomialForSpheres_2_3`),...

3.4 Communication algebra-geometry

Figure 1 illustrates how algebra and geometry communicate in this design. In this example, to compute the (non-empty) intersection of a 3D circle and a plane, we first retrieve simpler underlying geometric objects and solve the system of corresponding equations.

[6] $< 0, > 0, = 0$

Figure 1 Example of a geometric construction: computing the intersection of a circle and a plane. Names in boxes represent 3D Spherical Kernel or Algebraic Kernel functors, while names in ellipses are access functions.



4 Expanding the 3D Spherical Kernel for objects on a reference sphere

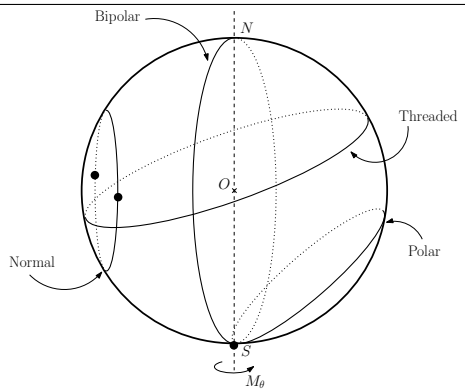
4.1 Objects on a reference sphere

In this section, we expand the basic functionalities of the 3D Spherical Kernel in the special case where all objects handled lie on a common sphere, called a *reference sphere* and denoted S_0 thereafter. We assume the reference sphere is equipped with cylindrical coordinates. This case is naturally more restrictive, but yields new problems, as one may for example wish to compare the cylindrical coordinates of the endpoints of two circular arcs on the same reference sphere. Developing this expansion requires more types together with related predicates and constructions. Note that the Algebraic Kernel is not expanded since being in such a setting does not modify the underlying algebraic objects and methods. This expansion is actually motivated by the computation of arrangements of circles on a sphere, as explained in section 5.

Let M_θ be a parametrized meridian on S_0, with $\theta \in [0, 2\pi($. We first define a classification of circles on the same reference sphere, and related quantities:

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

Figure 2 The four types of circles on a reference sphere. Black dots are the θ-extremal points.



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

Definition. 2 *The θ-extremal points of a normal circle are given by 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 values of θ 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 extremal points are usually called critical points. Notice also that the extremal points should not be confused with the extremities (i.e. the endpoints) of an arbitrary circular arc.

Definition. 4 *A circular arc on a reference sphere is said to be θ-monotonic 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. 2. A circular arc on a normal circle is θ-monotonic if it contains none of its circle θ-extremal point, 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-monotonic. No θ-monotonic circular arc can be found on a bipolar circle.

4.2 User Interface

Types. As an expansion of the previously introduced primitives, we introduce the following new objects: – `SK::CircleOnReferenceSphere_3` that represents circles on a given reference sphere;
– `SK::CircularArcPointOnReferenceSphere_3` that represents points on such circles;
– `SK::CircularArcOnReferenceSphere_3` that represents circular arcs delimited by such points;
– `SK::ThetaRep` that represents the θ coordinate of a point.

These expanded primitives are introduced in order to extend the user interface that now provides several predicates and constructions specific to this setting. Note that the corresponding C++ concepts refine[7] those corresponding to types introduced in section 3, so that the functionalities of the basic concepts are also provided.

Access functions.
Access functions are associated to the previously introduced types:
– `SK::CircleOnReferenceSphere_3`
  – `reference_sphere()` returns S_0;
  – `type_of_circle_on_reference_sphere()` indicates whether a circle is normal, threaded, polar or bipolar;
– `SK::CircularArcPointOnReferenceSphere_3`
  – `reference_sphere()` returns S_0;
  – `theta_rep()` returns an exact representation, of type `SK::ThetaRep`, of the θ coordinate of a point by means of its tangent or cotangent, the alternative being specified by an index corresponding to the decomposition of the unit disk $x^2 + y^2 \leq r_0^2$ —see section 6.3.
– `SK::CircularArcOnReferenceSphere_3`
  – `reference_sphere()` returns S_0.

Predicates.
The following predicates (returning a value in $\{<, =, >\}$) are available:
– Functor `SK::CompareTheta_3` compares the θ values of two points of type `SK::CircularArcPointOnReferenceSphere_3` or two values of θ encoded using `SK::ThetaRep`.
– Functor `SK::CompareThetaZ_3` compares two points of type `SK::CircularArcPointOnReferenceSphere_3` using lexicographic order on (θ, z).

[7] In the generic programming terminology, this means exactly that these concepts extend the requirements of previous concepts.

– Functor `SK::CompareZAtTheta_3` compares two circular arcs along a meridian M_θ of S_0 included in a rational plane.

Similarly, it compares the position of a point on S_0 with respect to a θ-monotonic circular arc drawn on the same sphere. As a precondition, the θ value of the point must be enclosed between the θ values of the endpoints of the circular arc.
– Functor `SK::CompareZToLeft_3` compares two θ-monotonic circular arcs along a meridian located just before the θ value of a point common to both circular arcs.

Constructions.
– Functor `SK::Intersect_3` returns an iterator over the solutions (possibly none) when intersecting two expanded primitives. Note that the solutions are expanded primitives.
– Functor `SK::ThetaExtremalPoint_3` returns an iterator over (possibly none) elements of type
`SK::CircularArcPointOnReferenceSphere_3`, which are the θ-extremal points of a circle or a circular arc.
– Functor `SK::MakeThetaMonotonic_3` returns an iterator over elements of type
`SK::CircularArcOnReferenceSphere_3` defining the θ-monotonic decomposition of a circular arc of type `SK::CircularArcOnReferenceSphere_3` or of a circle of type `SK::CircleOnReferenceSphere_3`.

5 Application to compute exact arrangements of circles on a sphere

In this section, we present the connexion between the 3D Spherical Kernel and a Bentley-Ottmann like algorithm computing the exact arrangement of circles on a sphere [11]. 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 a half-edge data structure. In the following, we explain how primitives from the 3D Spherical Kernel are used, and refer the reader to the companion paper [11] for the details on the algorithm itself.

5.1 Arrangements of circles on a sphere

To the best of our knowledge, there exist two algorithms computing the exact arrangement 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 [8]. But for a surface which is a sphere, great circles only 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, developed in [11]. The algorithm handles all types of circles and any degenerate case.

For the sake of completeness, one should also mention papers dealing with arrangements of quadrics. 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 is described in [7]. In theory, this algorithm could be adapted to the special case of spheres. In [36], an algorithm to compute exact 3D arrangements of quadrics is developed, but no implementation is provided.

It should be noticed that our implementation of the 3D Spherical Kernel concept should open the possibility to write a traits class providing the predicates required to handle all types of circles in the general sweep-line algorithm for curves on a surface [8]. In principle, this endeavor is indeed equivalent to embedding the critical and intersection points into the parameter space of the surface —the reference sphere here.

5.2 Sweeping the sphere using the 3D Spherical Kernel

Algorithm outline. Consider a collection of circles on a reference sphere. Normal and polar circles are first decomposed into θ-monotonic circular arcs, this decomposition being induced by the θ-extremal points. The sweep process consists of sweeping the sphere with a meridian M_θ anchored at the poles, using cylindrical coordinates. This process is tantamount to the classical sweep-line process in the plane, as each θ-monotonic circular arc is intersected in at most one point by the sweep meridian. An event corresponds to an intersection/tangency point between two circular arcs, or to a critical point of a circle. To handle degeneracies, that is events associated to the same point of the reference sphere, events are gathered into a data structure called the event site [11]. The vertical ordering $\mathcal{V}$ stores θ-monotonic circular arcs, while the event queue $\mathcal{E}$ stores event sites —which are created upon insertion of intersection/tangency/critical events. The algorithm is summarized on Table 1. The numerical primitives involved are typeset in typewriter font. These primitives are those from the 3D Spherical Kernel involved in the implementation of the geometric predicates required by the algorithm. To discuss these primitives in the context of the 3D Spherical Kernel, we consider constructions and predicates.

Constructions. These are twofold:
– constructing the intersection point(s) of circles using functor `SK::Intersect_3`,
– constructing the critical point(s) of a circle with functor `SK::ThetaExtremalPoint_3`.

Predicates. Predicates are involved in the manipulation of $\mathcal{V}$ and $\mathcal{E}$. Let us first examine the vertical ordering $\mathcal{V}$. The initialization of $\mathcal{V}$ requires comparing the position of circular arcs along meridian M_0 (at $\theta = 0$) using predicate `SK::CompareZAtTheta_3` with two circular arcs, as shown on Fig. 3(a). For intersections occurring on M_0, one further needs to compare the circular arcs to the left of this point using predicate `SK::CompareZToLeft_3`, as illustrated on Fig. 3(b).

To update $\mathcal{V}$, the only predicate involved is that required to insert the circular arcs of a normal circle starting at a given event point. To do so, we locate along the meridian the

start point [8] of the normal circle amongst circular arcs present in $\mathcal{V}$, the predicate involved being `SK::CompareZAtTheta_3` with one point and one circular arc. This is illustrated on Fig. 4. A comment is in order about the reversal of the order of circular arcs when processing an event site, so as to maintain the vertical ordering $\mathcal{V}$. Our algorithm does not use any predicate to perform this update, as it is shown in [11] that this operation can be handled in a purely combinatorial fashion.

Let us now analyze the event queue $\mathcal{E}$. Its initialization requires all critical points. Given all but threaded circles —using circle access function `type_of_circle_on_reference_sphere_3()`, such points are constructed using functor `SK::ThetaExtremalPoint_3` [9]. The detection of new intersection points from new adjacencies along $\mathcal{V}$ uses predicate `SK::DoIntersect_3`. Finally, all intersection and critical points are sorted using predicates `SK::CompareTheta_3` and `SK::CompareZ_3`.

Table 1 The sweep algorithm of [11], together with the primitives of the 3D Spherical Kernel involved.

0. Classify circles as normal/polar/bipolar/threaded.
- ♦ `SK::CircleOnReferenceSphere_3::type_of_circle_on_reference_sphere_3()`

Compute extremal points and decompose circles into θ-monotonic circular arcs.
- ♦`SK::ThetaExtremalPoint_3`

1. Initialize $\mathcal{V}$: Fill $\mathcal{V}$ with circular arcs intersected by the meridian at $\theta = 0$.
- ♦ `SK::CompareZAtTheta_3`, `SK::CompareZToLeft_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 critical points into $\mathcal{E}$.
  - ♦`SK::CompareTheta_3`, `SK::CompareZ_3`

3. While $\mathcal{E}$ is not empty do
- (a) Insert into/remove from $\mathcal{V}$ the circular arcs of the event(s) starting/ending —if any.
  - ♦`SK::CompareZAtTheta_3`
- (b) Reverse the order of circular arcs in vertical ordering $\mathcal{V}$.
- (c) Insert into $\mathcal{E}$ the intersection detected from the new adjacencies along $\mathcal{V}$.
  - ♦`SK::DoIntersect_3`, `SK::Intersect_3`, `SK::CompareTheta_3`, `SK::CompareZ_3`

[8] The start(end) point is the extremal point of the circle from which the circle starts (ends) being swept by the meridian.

[9] see section 6.3.8 for critical points of polar and bipolar circles.

Figure 3 Initializing $\mathcal{V}$ requires sorting circular arcs along the meridian M_0. Black dots represent critical points.

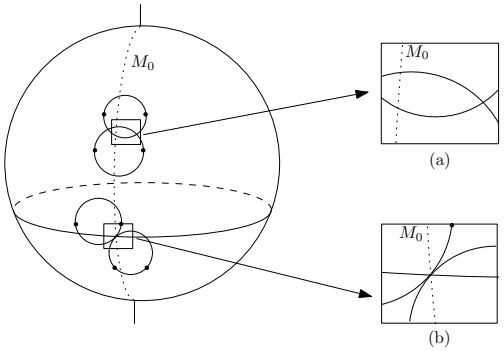
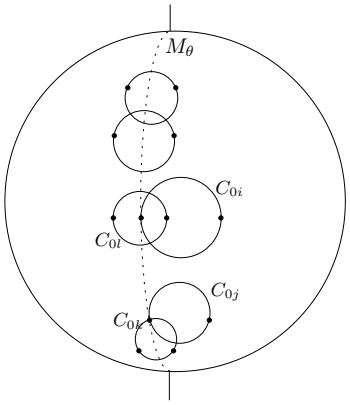


Figure 4 Inserting the circular arcs of a starting circle C_{0i} requires locating its start point amongst the circular arcs found in $\mathcal{V}$. Black dots represent critical points.



6 Mathematical foundations

This section details the analytical geometry involved in the predicates and constructions. Because our focus is a kernel concept, notice different models may use these mathematics differently by storing or recomputing the geometric quantities involved.

6.1 Preliminaries and notations

The center of sphere S_i is denoted $c_i = (x_i, y_i, z_i)$, and its radius r_i. The Cartesian coordinates of the center and the squared radius are called the sphere parameters, and recall that we shall assume these parameters are rational numbers. Without loss of generality, we will consider a reference sphere S_0 centered at the origin.

The x coordinate of point p and vector u are denoted p_x and u_x, and similarly for coordinates y and z. 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 $\mathrm{Sign}(x)$, is such that $\mathrm{Sign}(x) \in \{-1, 0, 1\}$. If o stands for the origin, and p is a point, the vector op is denoted $\overline{p}$.

The power of point p wrt 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 containing the points having equal power with respect to the two spheres. Whenever the spheres intersect, the intersection circle is also defined as the intersection between either sphere and the radical plane.

When considering two intersecting spheres S_i and S_j, the intersection circle is denoted C_{ij}, whose center and radius are denoted c_{ij} and r_{ij}. When considering the reference sphere model, if C_{0j} is a normal circle —refer to Def. 1, 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-monotonic circular arc included into a upper (lower) arc is also termed upper (lower) and a θ-monotonic circular arc on a polar circle passing by the north (south) pole is termed lower (upper).

A root of a degree n polynomial with rational coefficients is called an algebraic number of degree n. As algebraic numbers of degree two play a central role in our predicates and constructions, we just recall the following properties:

Observation. 1 *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.*

Practically, the operations to be performed involve algebraic numbers of degree two in the same extension, as well as sign evaluation. Implementation details are provided in section 7.

Observation. 2 *An equation of the radical plane between spheres S_i and S_j is:*

$$2 < \overline{p}, c_i c_j > + \overline{c_i}^2 - \overline{c_j}^2 + r_j^2 - r_i^2 = 0$$

Proof. The radical plane between the spheres S_i and S_j contains the set of points p satisfying:

$$\begin{cases} c_i p^2 - r_i^2 = 0 \\ c_j p^2 - r_j^2 = 0 \end{cases}$$

since $c_k p^2 - r_k^2 = (-\overline{c_k} + \overline{p})^2 - r_k^2 = \overline{c_k}^2 + \overline{p}^2 - 2 < \overline{p}, \overline{c_k} > -r_k^2$, the difference of the two equations of the previous system implies:

$$2 < \overline{p}, \overline{c_j} > -2 < \overline{p}, \overline{c_i} > + \overline{c_i}^2 - \overline{c_j}^2 + r_j^2 - r_i^2 = 0$$

which leads to the equation of the radical plane: $2 < \overline{p}, c_i c_j > + \overline{c_i}^2 - \overline{c_j}^2 + r_j^2 - r_i^2 = 0$ □

6.2 Constructions

6.2.1 Computing intersections

The developments presented in this section are naturally involved in the implementation of `SK::Intersect_3`, but also in those of a number of constructors (that of a circle from a sphere and a plane, for example), and access functions.

Given the primitives introduced in section 3, we shall consider the following six three dimensional objects: plane, sphere, circle, circular arc, segment and line. We examine the pairwise intersection of any two such primitives, as the intersection of an arbitrary number of them can always be computed from pairwise intersections. When dealing with two objects, we shall assume without loss of genericity that (i) they are different, (ii) they do not overlap —for circular arcs and for segments only, (iii) they do not include one another —circle on sphere and line on a plane, (iv) their intersection is non empty. Since we provide methods to test the excluded case, we cover all cases. The following 21 cases, sorted into four classes as a function of the expected intersection type, have to be accommodated:
– **Line**: {plane ∩ plane}
– **Point with rational Cartesian coordinates**: {plane ∩ line, line ∩ line, plane ∩ segment, segment ∩ segment, line ∩ segment }
– **Circle**: { plane ∩ sphere, sphere ∩ sphere}
– **Point(s) whose Cartesian coordinates are algebraic numbers of degree two:** { sphere ∩ line, sphere ∩ segment, circle ∩ circle, sphere ∩ circle, plane ∩ circle, plane ∩ circular arc, sphere ∩ circular arc, circle ∩ circular arc, circular arc ∩ circular arc, circle ∩ line, circle ∩ segment, circular arc ∩ segment, circular arc ∩ line}.

It is easily seen that we just have to detail one construction per class —the most generic one, since other intersections can be inferred from that case using the predicate testing inclusion —using predicate `SK::HasOn_3` described below.

Plane ∩ Plane. To parameterize the intersection line of two planes, we compute the director vector of the line —the cross product of the normal vectors of the planes, and a common point of the two planes.

Plane ∩ Line. A line can be parametrized by one parameter. So finding the intersection point of a plane and a line is equivalent to solving an equation of degree one.

Sphere ∩ Sphere. The characterization of the center and the squared radius of the intersection circle of two spheres relies on the following:

Observation. 3 *The center of the intersection circle of two spheres S_i and S_j is a linear combination of the two spheres' centers, the coefficients being quotients of polynomial expressions of degree two of the parameters of the spheres.*

The squared radius of an intersection circle is a quotient of polynomials of the same parameters.

Proof. To find the coordinates of the center c_{ij} of the intersection circle, we seek the intersection between the line joining the centers, and the radical plane of the two spheres. If $p = (x, y, z)$, we have the following system defining a line and a plane:

$$\begin{cases} c_i c_{ij} = \alpha c_i c_j \qquad \alpha \in \mathbb{R} \\ 2 < \overline{p}, c_i c_j > + \overline{c_i}^2 - \overline{c_j}^2 + r_j^2 - r_i^2 = 0 \end{cases}$$

The fact that c_{ij} lies on the radical plane can be written by

$$2 < \overline{c_{ij}}, c_i c_j > + \overline{c_i}^2 - \overline{c_j}^2 + r_j^2 - r_i^2 = 0 \Longleftrightarrow \alpha = \frac{c_j c_j^2 - r_j^2 + r_i^2}{2 c_i c_j^2}$$

Whence

$$c_i c_{ij} = \frac{c_i c_j^2 - r_j^2 + r_i^2}{2 c_i c_j^2} c_i c_j \tag{1}$$

For the squared radius r_{ij}^2, we use Pythagora's theorem: $r_{ij}^2 = r_i^2 - c_i c_{ij}^2$. □

Notice that the radical plane is orthogonal to the line joining the centers.

Sphere ∩ Line. An intersection point between a sphere and a line is characterized by:

Observation. 4 *The Cartesian coordinates of intersection points of a sphere and a line are algebraic numbers of degree two in the same extension.*

Suppose the line is defined by $(a_1 t + b_1, a_2 t + b_2, a_3 t + b_3)$ and the sphere by $(x-a)^2 + (y-b)^2 + (z-c)^2 - r^2 = 0$. The Cartesian coordinates of the intersection points of a sphere and a line are $(a_1 t' + b_1, a_2 t' + b_2, a_3 t' + b_3)$ with t' solution of $(a_1 t + b_1 - a)^2 + (a_2 t + b_2 - b)^2 + (a_3 t + b_3 - c)^2 - r^2 = 0$.

6.2.2 Computing θ extremal points of a normal circle

This section is concerned with the foundations of the functor `SK::ThetaExtremalPoint_3`. The following observation sets the case of normal circles on a reference sphere:

Observation. 5 *Consider a normal circle C_{0i} defined by the intersection of two spheres with rational parameters. The z coordinate of its two critical points is the following rational number:*

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

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 $z_i = 0$, a symmetry argument with respect to the plane $z = 0$ imposes that the z coordinate of the critical points is also null. In the following, we therefore assume that $z_i \neq 0$.

In the general case, the critical points of C_{0i} are the two points where the intersection between the meridian M_θ and C_{0i} reduces to one point.

Consider the system involving circle C_{0i} (or equivalently the radical plane RP_{0i}), the half plane $P(\theta)$ defining the meridian M_θ (that is $M_\theta = S_0 \cap P(\theta)$), and the sphere S_0. If

$p = (x, y, z)$, the θ values of the critical points correspond to the solutions of the following system:

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

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 the poles are not critical for such a circle. The previous system is tantamount to:

$$\begin{cases} y = x\tan\theta \\ z = \frac{\overline{c_i}^2 + r_0^2 - r_i^2 - 2x(x_i + y_i\tan\theta)}{2z_i} \\ x^2 + x^2\tan^2\theta + \frac{(\overline{c_i}^2 + r_0^2 - r_i^2 - 2x(x_i + y_i\tan\theta))^2}{4z_i^2} - r_0^2 = 0 \end{cases} \tag{3}$$

Rewrite the last equation

$$Ax^2 + Bx + C = 0, \tag{4}$$

with

$$\begin{cases} A = (1 + \tan^2\theta) + \frac{(x_i + y_i\tan\theta)^2}{z_i^2} \\ B = -\frac{(x_i + y_i\tan\theta)(\overline{c_i}^2 + r_0^2 - r_i^2)}{z_i^2} \\ C = \frac{(\overline{c_i}^2 + r_0^2 - r_i^2)^2}{4z_i^2} - r_0^2 \end{cases} \tag{5}$$

The values of θ sought are such that Eq. (4) 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:

$$D = \overbrace{(-\lambda_i^2 + 4r_0^2z_i^2 + 4r_0^2y_i^2)}^{D_2}T^2 + \overbrace{(8x_iy_ir_0^2)}^{D_1}T + \overbrace{(-\lambda_i^2 + 4r_0^2z_i^2 + 4r_0^2x_i^2)}^{D_0} \tag{6}$$

We proceed by investigating the cases where polynomial D has degree two and one. As we shall see, in the first case, no critical point lies in the plane $x = 0$, while the second one corresponds to the situation where one critical point lies in plane $x = 0$.

Case I: $D_2 \neq 0$: 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 critical points.

▷ The discriminant Δ of D may be written as:

$$\Delta = 4\overbrace{(-2r_0z_i + \lambda_i)}^{\Delta_1}\overbrace{(2r_0z_i + \lambda_i)}^{\Delta_2}\overbrace{(4r_0^2\overline{c_i}^2 - \lambda_i)}^{\Delta_3}$$

Since the power of a pole of S_0 wrt to sphere S_i reads as $(c_i - (0,0,\pm r_0))^2 - r_i^2 = \overline{c_i}^2 + r_0^2 - r_i^2 \pm 2z_i r_0 = \pm 2r_0 z_i + \lambda_i$, polynomial Δ_1 (Δ_2) is the power of the north (south) pole wrt S_i. The power of a point wrt to a sphere is negative (positive,zero) iff the point is inside (outside, on) the sphere. Therefore, Δ_1 is negative (positive,zero) if the north pole is inside (outside, on) S_i. The same observation holds for Δ_2 and the south pole.

Let now consider Δ_3, which we rewrite as: $\Delta_3 = \overbrace{(-\overline{c_i}^2 + (r_0 + r_i)^2)}^{\Delta_{3_1}}\overbrace{(\overline{c_i}^2 - (r_0 - r_i)^2)}^{\Delta_{3_2}}$. The combination of signs of Δ_{3_1} and Δ_{3_2} is reported in table 2, and the discussion of the sign of Δ is summarized in table 3. As expected, one has $\Delta > 0$ for all normal circles, so that equation (6) can be canceled in two different ways corresponding to the two critical points.

▷ Finally, substituting any solution of equation (6) into the expression of z from the system (3) yields $z = \frac{2z_i r_0^2}{\overline{c_i}^2 + r_0^2 - r_i^2}$, which corresponds to the z coordinate of the critical points of a normal circle under the assumption $D_2 \neq 0$.

Table 2 Sign of Δ_3. 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 *balls associated to* S_i *and* S_j *are disjoint.*

Δ_{3_1}	Δ_{3_2}	Δ_3	**sphere configuration**
+	+	+	S_0 and S_i intersect along a 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

Case II: $D_2 = 0$: polynomial D is of degree one. We first show that the circle is a normal circle if $x_i \neq 0$ and $y_i \neq 0$, and then derive the expression of the z value.

▷ Notice that $D_2 = \Delta_3 - 4r_0^2 x_i^2$. $D_2 = 0$ is equivalent to $\Delta_3 \geq 0$.
Consider the product $\Delta_1\Delta_2 = (-2r_0 z_i + \lambda_i)(2r_0 z_i + \lambda_i)$. We have $\Delta_1\Delta_2 = \lambda_i^2 - 4r_0^2 z_i^2 = -\Delta_3 + 4r_0^2(x_i^2 + y_i^2)$. Since $\Delta_3 = 4r_0^2 x_i^2$, $\Delta_1\Delta_2 = 4r_0^2 y_i^2$. So $\Delta_1\Delta_2 \geq 0$. First note that according to table 3, if $x_i = 0$, then S_0 and S_i are tangent, and if $y_i = 0$ the circle goes through at least one pole, which in both cases does not correspond to the case of a normal circle. Assuming that $x_i \neq 0$ and $y_i \neq 0$, we have $\Delta > 0$, and according to table 3, the intersection circle is normal.

▷ We now compute the z coordinate of the critical points of a normal circle such that $D_2 = 0$, $x_i \neq 0$ and $y_i \neq 0$. We directly have $\tan\theta = -\frac{-\lambda_i^2 + 4r_0^2 z_i^2 + 4r_0^2 x_i^2}{8x_i y_i r_0^2}$ from equation (6). Since by

hypothesis $-\lambda_i^2 + 4r_0^2 z_i^2 = -4r_0^2 y_i^2$, one has $\tan\theta = -\frac{x_i^2 - y_i^2}{2x_i y_i}$. Substituting $\tan\theta$ by $-\frac{x_i^2 - y_i^2}{2x_i y_i}$ into the expression of z of system (3) gives $z = \frac{\lambda_i z_i}{2(z_i^2 + y_i^2)}$. The desired expression stems from the fact that $z_i^2 + y_i^2 = \frac{\lambda_i^2}{4r_0^2}$.

▷ The previous derivation gives the solution for only one critical point, since only one value of $\tan\theta$ is obtained. In fact, one value of $\tan\theta$ is defined, since the second value corresponds to $\pm\infty$, i.e. the second θ value is $\pi/2$ or $3\pi/2$. Indeed, swapping x and y in the previous derivation gives us the equations for determining the value of $T = \cot\theta$. In particular, equation (6) becomes

$$D_0 T^2 + (8x_i y_i r_0^2)T + D_2 = 0, \tag{7}$$

which is equivalent to

$$\begin{cases} T = 0 \\ D_0 T + (8x_i y_i r_0^2) = 0 \qquad \text{if } D_0 \neq 0 \end{cases} \tag{8}$$

We find the solution $\cot\theta = 0$ which was expected, and using the symmetric role played by x and y, the expression of z is the same than that obtained with $\tan\theta$. Notice that if D_0 is also null, all what is written above stay valid, but $\tan\theta$ becomes null for the first θ value which means that the θ value of the corresponding critical point is 0 or π. This corresponds to the situation where one critical point is on the plane $x = 0$ and the other one on the plane $y = 0$.

To conclude the case analysis, notice that $\overline{c_i}^2 + r_0^2 - r_i^2 = 0$ corresponds to the case where the circle is bipolar. □

Table 3 Sign of Δ. Δ_1 (Δ_2) is the power of the north (south) pole wrt S_i while Δ_3 indicates whether S_0 intersects S_i.

Δ_1	Δ_2	Δ_3	Δ	**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	S_0 and S_i are tangent if sign of Δ_1 = sign of Δ_2, impossible otherwise
0	0	$\forall$	0	bipolar circle if $\Delta_3 > 0$, impossible otherwise

The observation 5 has an interesting consequence:

Observation. 6 *The critical points of a normal circle C_{0i} are defined by the intersection between the reference sphere, and the line intersection between the radical plane RP_{0i} and the horizontal plane defined by $z = \frac{2z_i r_0^2}{\overline{c_i}^2 + r_0^2 - r_i^2}$. The x and y coordinates of these points are algebraic numbers of degree two by observation 4.*

6.2.3 Decompositions into θ-monotonic circular arcs

Functor `SK::MakeThetaMonotonic_3` decomposes a circular arc or a circle into θ-monotonic circular arcs. This is trivial for circular arcs on threaded circles. For a circular arc on a normal circle, this operation requires computing the critical points of the circle, and checking whether these points lie on the circular arc. This later test requires comparing the z coordinate values of the extremities of the circular arc with the values of the critical points. For a circular arc on a polar circle the operation is similar, considering the pole the circle goes through as critical point. For circles, the inclusion test is not necessary.

6.3 Predicates

6.3.1 Comparing Cartesian coordinates of points on $3D$ circles

As a general prerequisites observe that Cartesian coordinates of points on a sphere are algebraic numbers of degree at most two —observation 4. Comparing these Cartesian coordinates reduces to comparing these algebraic numbers.

6.3.2 Evaluating the sign of a polynomial

The following is concerned with predicates `AK::SignAt`,`SK::CompareZAtXY_3`, `SK::CompareYAtXZ_3`, `SK::CompareXAtYZ_3` and `SK::CompareZAtTheta_3`.

We wish to evaluate the sign of a trivariate polynomial of degree at most two at a point on a sphere. Since the Cartesian coordinates of a point on a sphere are algebraic numbers of degree two in the same extension, following observation 1, the evaluation of the trivariate polynomial is also an algebraic number of degree two in the same extension.

6.3.3 Testing equality

The following is concerned with predicate `SK::Equal_3`.

For objects whose representation is unique, i.e. points and spheres, the test is trivial. For segments, we test equality of endpoints. For planes and lines, we test the non-independence of coefficients of the equations. For circles, we successively test the equality of containing planes, circle centers and squared radii. For circular arcs, we test the equality of the circles and that of the endpoints.

6.3.4 Testing inclusion

The following is concerned with predicate `SK::HasOn_3`.

Consider two objects of the 3D Spherical Kernel, assumed to be topologically closed. We wish to test whether the first one is included into the second one. We examine separately the case of points and the case of remaining primitives.

The case of points. We wish to test whether a point lies on a sphere, a plane, a line, a segment, a circle or a circular arc. For a sphere and plane, this operation is equivalent to the sign evaluation of their associated polynomials using `AK::SignAt` at the input point.
For a line, this is trivial using either two planes defining the line or its parametrization. For a segment, we compute the signs of the dot products of the vectors involving the point and the segment endpoints. For a circle, we test if the input point belongs to the plane of the circle and to the diametrical sphere associated to the circle. For a circular arc we test whether the input point belongs to the supporting circle first, and then check whether it belongs to the arc itself. For this latter test, denote s, t, c the source, target, and center of the circular arc and let N stands for the unit normal vector of the positive plane containing the circle. We suppose that $s \neq t$ else testing inclusion into the associated circle is enough. A circular arc is such that we go counter clockwise from its source to its target as imposed by the normal vector N. In addition, for two points p and q on the circle such that $p \neq q$, let $S_{pq} = \text{Sign}(< cp \wedge cq, N >)$. Notice S_{pq} tells us whether we move from p to q counter clockwise according to N, following the smallest path. Whenever neither points s, t and c nor points s, p and c nor points p, t and c are aligned, all possible cases are summarized on table 4. If c, s, p or c, t, p are collinear, the input point lies on the circular arc iff $S_{st} \leq 0$. If c, s, t are collinear, the input point lies on the circular arc iff $S_{sp} \geq 0$.

Table 4 Deciding whether point p belongs to the closed circular arc delimited by s and t.

S_{st}	S_{sp}	S_{pt}	`SK::HasOn_3`
1	1	1	true
1	-1	± 1	false
1	1	-1	false
-1	1	$\pm$ 1	true
-1	-1	1	true
-1	-1	-1	false

Other primitives. The test is trivial using `SK::Equal_3` for circle or a circular arcs with respect to a plane, and for a circular arc on a circle. A plane contains a line, iff the normal vector of the plane and the director vector of the line are orthogonal and the plane contains (`SK::HasOn_3`) one point of the line. A line or a plane contains a segment if it contains its two endpoints. A circle belongs to a sphere S, iff sphere S belongs to the pencil of spheres defined by the diametrical sphere of the circle and its supporting plane.

6.3.5 Testing overlap

The following is concerned with predicate `SK::DoOverlap_3`.

This predicate, which is only relevant for circular arcs and segments, returns true if two arcs or two segments overlaps, that is to say if they have more than two points in common. The test consists of first checking whether the circles (lines) associated to the arcs (segments) are identical, and if so, of checking the position of the endpoints of one arc (segment) with respect to the other arc using predicate `SK::HasOn_3`.

6.3.6 Testing intersection

The following is concerned with predicate `SK::DoIntersect_3`.

In the following, we suppose that the objects handled are different and that they do not overlap. Similarly to section 6.2.1, we focus on the description of one case per class of intersection result, the other cases being handled from the base case together with predicate `SK::HasOn_3`. The base cases are:

– **Plane** $\cap$ **Plane** $\neq \emptyset \iff$ Normal vectors are not collinear.
– **Plane** $\cap$ **Line** $\neq \emptyset \iff$ Normal vector and director vector are not orthogonal.
– **Sphere** $\cap$ **Line** $\neq \emptyset \iff$ Number of real roots of the polynomial specified in Observation 4 must be greater or equal to one.
– **Sphere** $\mathbf{S_i}$ $\cap$ **Sphere** $\mathbf{S_j} \neq \emptyset \iff \left((c_i - c_j)^2 - r_i^2 - r_j^2\right)^2 \leq 4r_i^2 r_j^2$.

6.3.7 Classifying a circle on a reference sphere

The following is concerned with member function of the type representing circles on a reference
`type_of_circle_on_reference_sphere_3()`.

To classify a circle on the reference sphere, as specified in Def. 1, we intersect the circle by the plane $y = 0$:

Observation. 7 *A circle is:*

–Threaded iff its intersection with the plane $y = 0$ yields two intersection points with x coordinates of opposite signs.

–Polar iff its intersection with the plane $y = 0$ yields two intersection points such that one has $x \neq 0$ and one $x = 0$, or a single intersection point at $x = 0$ and $y = 0$.

–Bipolar iff the circle lies in the plane $y = 0$, or if its intersection with the plane $y = 0$ yields two intersection points such that $x = 0$.

6.3.8 Comparing cylindrical coordinates of points on the reference sphere

The following is concerned with predicate `SK::CompareTheta_3` and constructor of the class `SK::ThetaRep`.

Predicate `SK::CompareTheta_3` is available for points represented under the type `SK::CircularArcPointOnReferenceSphere_3` only. The comparison strategy consists of comparing $\tan\theta$ or $\cot\theta$ based on the Cartesian coordinates of the points. We first present the general principle, and then indicate how to accommodate particular cases i.e. the poles.

Quadrants, tangents and cotangents. Assume the reference sphere is endowed with cylindrical coordinates —which is most convenient when sweeping the sphere with a rotating meridian as in section 5. Comparing the values of θ of two points involves inverse trigonometric functions and is non trivial. To avoid such calculations, we resort to a decomposition of the punctured disk $x^2 + y^2 \leq r_0^2$ into open half-quadrants and line-segments, as depicted on Fig. 5. Puncturing the disk corresponds to removing the origin, as poles only project onto it. As explained in the next paragraph, poles deserve a special treatment. More precisely, we decompose the punctured disk into 16 cells: eight open half quadrants, and eight line-segments. Each quadrant is associated an index in the range 1..8 in increasing order, while the segments are assigned a rational index equal to the average of the two neighbors' indices. (For the segment between half quadrants 1 and 8 we take 1/2 as a convention.)

To compare the θ values of two points, we first assign each point to its cell. Computing the sign of x and y, we can easily isolate the containing segment or the two candidate adjacent half quadrants. In this latter case, comparing $\mid x \mid$ and $\mid y \mid$ yields the solution.

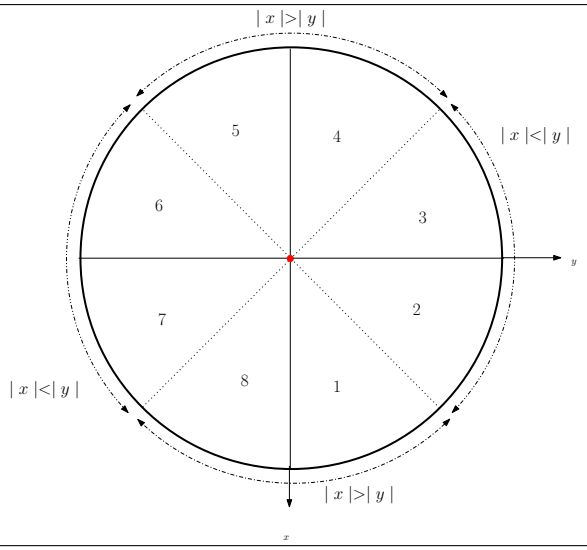
Two points falling in different cells are compared from the cells' indices. Inside a cell, two points may have the same θ —all points on a ray meet this condition. To check this condition or find the relative ordering of the points, we have $\tan\theta = x/y$ or $\cot\theta = y/x$. More precisely, as $\tan\theta$ is not defined for $\theta = \pi/2 \mod \pi$, while $\cot\theta$ is not so for $\theta = 0$

mod π, comparison inside two-dimensional cells consists of comparing

$$\begin{cases} \tan\theta \text{ for cells of index } 8,1,4,5, \\ \cot\theta \text{ for cells of index } 2,3,6,7. \end{cases} \tag{9}$$

The Cartesian coordinates of a point being algebraic numbers of degree two in the same extension, so are the ratios x/y and y/x —if defined.

Figure 5 Decomposition of the punctured disk $x^2 + y^2 \leq r_0^2$ into open half quadrants and line segments.



θ-extremal values for polar and bipolar circles. The comparison strategy based on the case analysis of (9) consists of comparing $\tan\theta$ or $\cot\theta$ based on the Cartesian coordinates, which is valid away from poles but poses a problem at the poles. On the other hand, the θ-extremal values of polar and bipolar circles have been defined in Def. 3.

In order to have a homogeneous comparison method based on Cartesian coordinates for all critical and intersection points, we associate to each θ-extremal value of a polar or bipolar circle a point away from the pole, whose x and y are such that $\tan\theta = x/y$ and/or $\cot\theta = y/x$. For circle C_{0i}, the two points associated to the two θ-extremal values are $p_1 = (y_i, -x_i, 0)$ and $p_2 = (-y_i, x_i, 0)$. The start and end point of a bipolar circle are such that the θ value associated to the start point is smaller. For a polar circle, while sweeping from its start point to its end points, the half-plane defining the meridian —as intersection with S_0, encounters the center of the circle.

Intersection points, critical points, and roots of degree two polynomials. Cartesian coordinates of intersection points between two circles, as well as critical points are

algebraic numbers of degree two. As such points usually come into pairs, we explain in the following how to identify the point of the pair having the smallest θ value for the ordering just introduced. The case of points falling in different cells or in the same line-segment being trivial, assume the two points fall in the same quadrant. Let (x_F, y_F, z_F) and (x_S, y_S, z_S) be two such points. To identify the smallest θ value, we need to compute $\text{Sign}(y_F x_S - x_F y_S)$. By observation 4, there exists four rational numbers a, b, c, d such that

$$\begin{cases} x_F = aX + b & y_F = cX + d, \text{ with } X \text{ algebraic number of degree two} \\ x_S = aY + b & y_S = cY + d, \text{ with } Y \text{ algebraic number of degree two} \end{cases} \tag{10}$$

Therefore, $\text{Sign}(y_F x_S - x_F y_S) = \text{Sign}((bc - ad)(X - Y))$. Since X and Y are roots of the same polynomial, we only have to evaluate $\text{Sign}(bc - ad)$.

6.3.9 Comparing circular arcs along a meridian

The following is concerned with predicate `SK::CompareZAtTheta_3` involving two θ-monotonic circular arcs.

Consider a meridian given by a rational half-plane, and assume we want to compare θ-monotonic circular arcs along this meridian. We proceed in two steps. First, we consider the plane containing the meridian and select intersection points with circular arcs that are in the cell of the meridian. Second, the selected intersection points are sorted using their z coordinates which are algebraic numbers of degree two —cf observation 4.

Notice that the case of a general meridian is not directly covered, since the circle associated to such a meridian is not defined by a sphere with rational parameters in general.

6.3.10 Sorting circular arcs at a common point

The following is concerned with predicate `SK::CompareZToLeft_3`.

Given an intersection point p common to two θ-monotonic circular arcs and lying on meridian M_θ, we wish to find the relative position (above, below) of the two arcs to the left of p, i.e. for the angle value $\theta - \varepsilon$ with ε arbitrarily small. The two spheres defining the corresponding circles on S_0 are denoted S_i and S_j. Notice that this predicate is not defined if point p is a pole, and that the associated circles are of any type but bipolar as no θ-monotonic circular arc is defined on such circles. The critical points of a normal circle decompose it into one *upper* and one *lower* circular arcs, while a circular arc on a polar circle passing by the north (south) pole is considered as a *lower* (*upper*) one.

The intersection point is also a critical point. Assume the common point matches a critical point of at least one of the two circles —the corresponding circle being a normal one:

Observation. 8 *Consider two θ-monotonic circular arcs A_i and A_j sharing a common point p. Suppose point p is a critical point of the normal circle of A_i. One has:*

– If p is not a critical point for the circle of A_j, then A_i is above (below) A_j if A_i is a upper (lower) arc of its circle.

– If p is a critical point for the normal circle associated to A_j, then (i) if a circular arc is an upper arc, it is above one which is a lower arc (ii) if the circular arcs are both upper (lower) arcs, then the one with largest (smallest) radius is above.

The intersection point is not a critical point. We first define tangent vectors at the common point, provide solutions depending on whether or not the intersection point features a tangency or not, and conclude with the algebraic complexity analysis.

▷ *Tangents to the circles at point p and their relative position.* If the circular arcs are not tangent at p, ordering them is tantamount to ordering their tangents. We shall denote t_k, $k = i, j$, the tangent vector associated to circle C_{0k} at point p. Tangents t_i and t_j are chosen so as, locally in the tangent space of S_0, to point towards *increasing* values of θ. Because t_i and t_j are orthogonal to $\overline{p}$, the relative position of the tangents is given by Sign(Δ), with

$$\Delta =< t_i \wedge t_j, \overline{p} > . \tag{11}$$

The details are as follows. The tangent to a circle is supported by the direction of 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 defined by:
– $t_k = \beta_k \overline{c_{0k}} \wedge n_k$, with $\beta_k = -1$ (1) if the circular arc where common point lies is an upper (a lower) one, for a **normal** or a **polar** circle.
– $t_k = \alpha_k \overline{c_k} \wedge n_k$ with $\gamma_k = \pm 1$ such that $\gamma_k \overline{c_k}.z > 0$, for a **threaded** circle.

▷ *Case I: circles $\mathbf{C_{0i}}$ and $\mathbf{C_{0j}}$ are not tangent at p.* Whenever $\Delta \neq 0$, the ordering along M_θ is as follows:
– If $\Delta < 0$ ($\Delta > 0$), the arc contributed by C_{0i} is below (above) that contributed by C_{0j}. See Fig. 6.
▷ *Case II: circles $\mathbf{C_{0i}}$ and $\mathbf{C_{0j}}$ are tangent at p.* Denote p_{kz} the z coordinate of the critical points of a normal circle C_{0k}. When $\Delta = 0$, we distinguish two sub-cases:

- If at least one of the two circles is threaded, say C_{0i}:
  - if C_{0j} is not threaded, we conclude from the sign of $p_z - p_{jz}$.
  - if C_{0j} is also threaded, as threaded circles with centers having the same z coordinate cannot be tangent, we compare the z coordinates of the centers of C_{0i} and C_{0j}.
- None of the circles is threaded. See Fig. 7 for illustration.

  For $k = i, j$, let $\overline{r}_k = r_{0k}^2 \text{Sign}(p_z - p_{kz})$. Notice $\overline{r_k}$ is always different from 0 because we suppose p is not a critical point.

Figure 6 Ordering the circular arcs using the tangents to circles C_{0i} and C_{0j} at p: (a) The arc of C_{0i} is an upper arc, that of C_0j a lower arc; to the left of p, the arc of C_{0i} is below the arc of C_{0j} (b) The arcs of C_{0i} and C_{0j} are lower arcs; to the left of p, the arc of C_{0i} is above the arc of C_{0j}

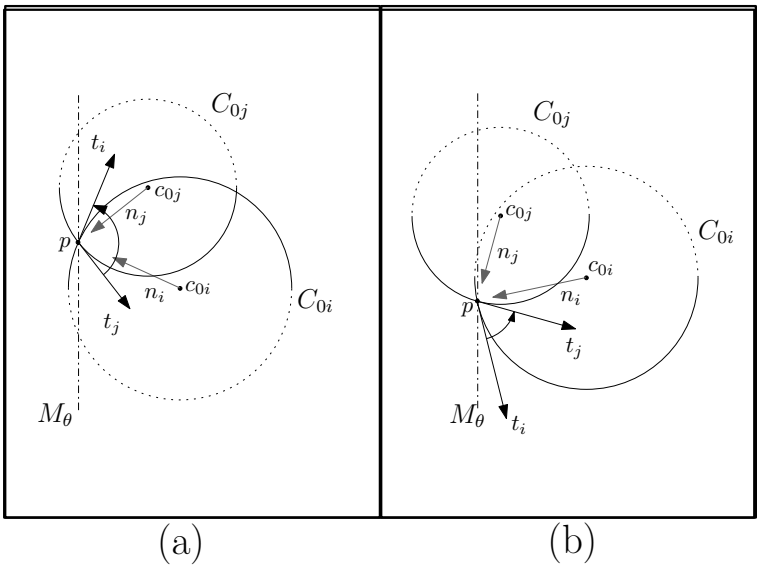
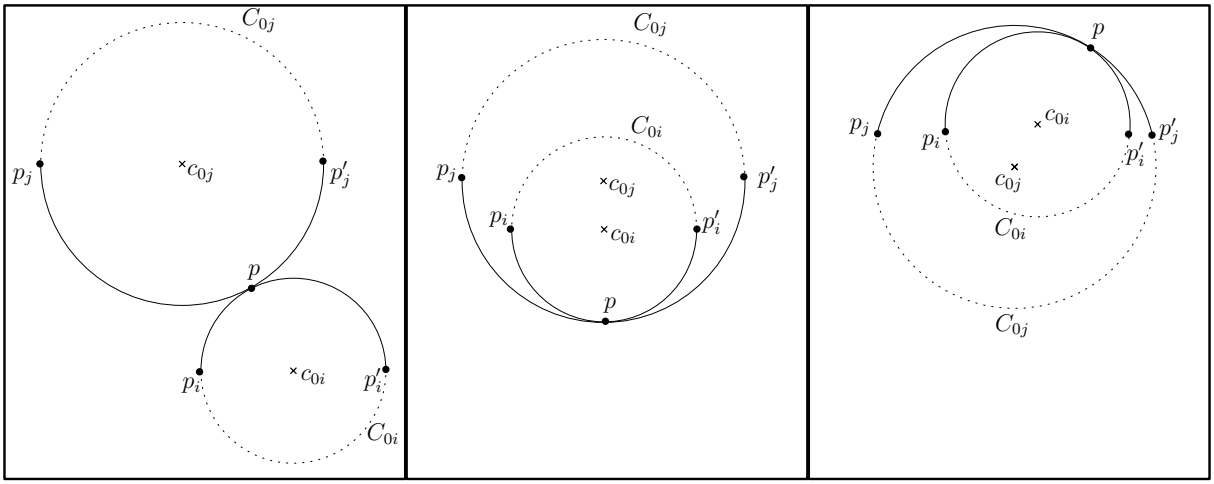


Figure 7 Ordering about tangency point p; black dots are critical points: (a) $\overline{r}_i\overline{r}_j < 0$ and $\overline{r}_i > 0$: C_{0i} is below C_{0j} ; (b) $\overline{r}_i\overline{r}_j > 0$, $\overline{r}_i < 0$ and $r_i < r_j$: C_{0i} is above C_{0j} ; (c) $\overline{r}_i\overline{r}_j > 0$, $\overline{r}_i > 0$ and $r_i < r_j$: C_{0i} is below C_{0j};



- if $\overline{r}_i\overline{r}_j > 0$: if $\overline{r}_i < 0$ ($\overline{r}_i > 0$), assuming $r_{0i} < r_{0j}$, the arc contributed by C_{0i} is above (below) that contributed by C_{0j}.
- if $\overline{r}_i\overline{r}_j < 0$: if $\overline{r}_i > 0$ ($\overline{r}_i < 0$), the arc contributed by C_{0i} is below (above) that contributed by C_{0j}.

▷ *Algebraic complexity analysis.* Let us now focus on the cost of evaluating the relative position of the two circular arcs.

Case II is easily ruled out, as it relies on sign of $p_z - p_{k_z}$, which is an algebraic number of degree two. Note that if $p_z - p_{k_z}$ is negative (positive), the circular arc of C_{0k} involved is a lower (upper) one.

Case I relies on Δ, which is an algebraic number of degree two obtained doing 9 products [10]

of algebraic numbers of degree two in the same extension. In the following we show that Sign(Δ) can actually be obtained with at most two products of algebraic numbers of degree two in the same extension. Recall that the circles C_{0i} and C_{0j} are not bipolar, therefore the tangent vectors are not in the plane of a meridian. We have the following elementary observation:

[10] The vector (dot) product requires 6 (3) products of algebraic numbers of degree two.

Observation. 9 *Let* $\Delta' = \frac{t_{j_z}}{\|t_j\|} - \frac{t_{i_z}}{\|t_i\|}$. *The sign of* Δ *is the same than the sign of* Δ'.

From which we conclude as follows:

Observation. 10 *The sign of* Δ' *can be evaluated using the sign of at most one algebraic number of degree two obtained using at most two products of two algebraic numbers of degree two in the same algebraic extension.*

Proof. In the following cases, the sign of Δ' is trivially inferred: $t_{i_z} = 0$ and $t_{j_z} = 0$; $t_{i_z} = 0$ or $t_{j_z} = 0$; t_{i_z} and t_{j_z} have different signs.

In the following, we suppose the signs of t_{i_z} and t_{j_z} are identical and non null. Recall that the center of circle $C_{0k}, k = i, j$ has been derived in Eq. (1). Since $\overline{c_{0k}}$ and $c_{0k}p$ are orthogonal, the sine of the angle between the vectors is 1 and we have:
– If $t_k = \pm\overline{c_{0k}} \wedge c_{0k}p$,

$$t_k^2 = \underbrace{\overline{c_{0k}}^2}_{\alpha_k^2\overline{c_k}^2}\underbrace{c_{0k}p^2}_{r_{0k}^2} \qquad \text{and} \qquad t_{k_z} = \pm\alpha_k(c_{kx}p_y - c_{ky}p_x)$$

with α_k a rational number given in proof of observation 3.
– If $t_k = \pm\overline{c_k} \wedge c_k p$,

$$t_k^2 = \overline{c_k}^2\underbrace{c_{0k}p^2}_{r_{0k}^2} \qquad \text{and} \qquad t_{k_z} = \pm(c_{kx}p_y - c_{ky}p_x)$$

The sign of Δ' is the same than that of $\|t_i\|t_{j_z} - \|t_j\|t_{i_z}$. But

$$t_i^2 t_{j_z}^2 - t_j^2 t_{i_z}^2 = (\|t_i\|t_{j_z} + \|t_j\|t_{iz})(\|t_i\|t_{j_z} - \|t_j\|t_{iz}) \tag{12}$$

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

$$\text{Sign}(\Delta') = \text{Sign}(t_{j_z})\text{Sign}(t_i^2(t_{j_z})^2 - t_j^2(t_{i_z})^2). \tag{13}$$

Since Cartesian coordinates of p are algebraic numbers of degree two in the same extension, so are t_{i_z} and t_{j_z}. If t_{j_z} and t_{i_z} have the same sign, since the square norms of the tangent vectors are rational numbers we need two products of algebraic numbers of degree two to compute the squared value of t_{i_z} and t_{j_z}. □

Remark. 1 *A special case of the previous predicate is involved with the initialization of the vertical ordering of the Bentley-Ottmann algorithm while computing the spherical arrangement of circles. Consider the case of the meridian* M_0 *located at* $\theta = 0$. *Given a set of circular arcs intersecting at a point p located on this meridian, evaluations of the sign of the* Δ' *need no manipulation of algebraic numbers. Indeed, we know that* $p_x > 0$ *and* $p_y = 0$. *Using expressions of* t_{k_z} *in the previous proof, we can factorize by* p_x *or* p_x^2 *and compute the sign of* Δ' *using rational numbers only.*

6.3.11 Locating a point wrt θ-monotonic circular arcs

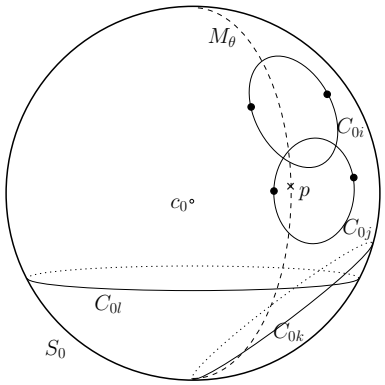
The following is concerned with predicate `SK::CompareZAtTheta_3` for one point and one θ-monotonic circular arc and more generally with predicate `SK::CompareZAtXY_3`.

Given a θ-monotonic circular arc and a point p on a meridian M_θ intersecting this circular arc, we would like to know the position (above, on, below) of point p w.r.t. the circular arc. To do so, we shall partly rely on the plane supporting the circular arc. We distinguish two cases.

The arc is associated to a normal circle. A normal circle C_{0i} can be split in two θ-monotonic circular arcs, a upper and a lower one. A such circle decomposes S_0 into two regions of unequal areas. (Great circles split S_0 into regions of equal areas, but such circles are bipolar or threaded). Given a normal circle C_{0i}, we shall use a predicate stating whether point p belongs to the cap of least area induced by circle C_{0i} on S_0. This predicate consists of evaluating the sign (with `AK::SignAt`) of the polynomial expression obtained by plugging the Cartesian coordinates of point p into the equation of the plane associated to the circle, using as normal vector $\overline{c_{0i}}$. We examine the two following cases:
– If point p lies inside the spherical caps of least area: point p is below the upper circular arc of C_{0i} and above lower one.
– If not, the position of p is given by the sign of the difference of z coordinates of p and of the critical point of C_{0i}.

The arc is associated to a polar or a threaded circle. The position of p w.r.t. any arc of such a circle is that of point p w.r.t. the plane of the circle, using as normal vector $u = \pm\overline{c_i}$ such that $u_z > 0$.

Figure 8 Position of point p wrt two normal circles, one threaded circle and one polar circle. Point p is inside (outside) the spherical caps of smallest area defined by C_{0j} (C_{0i}). p is above circles C_{0k} and C_{0l}



7 Implementation

In this section, we sketch our implementation of the kernel concept developed in this paper. For a given class and to limit space requirements, the emphasis is on data members. To make the presentation self-contained, we first present C++ prerequisites.

7.1 C++ prerequisites

Generalities. In C++, the keyword `typedef` allows to create a shortcut for a given type name. Inside a template class, types provided by the template parameter are accessed using the `typename` keyword. In the same way, inside a template class, while instantiating a nested template type which depends on a template parameter of the class, the keyword `template` is used to disambiguate the syntax. Finally, as a general remark, all classes of the 3D Spherical Kernel are instantiated with the 3D Spherical Kernel itself, so as to access all features provided by the 3D Spherical Kernel.

Reference counting. Reference counting is a classical strategy used to speed up copies of complicated objects. The strategy is simple: the object allocated is accompanied by an integer counting the number of references to that object. The value of this integer is increased (decreased) when a copy of the object is created (deleted). When the value of the integer vanishes, the object is de-allocated. This mechanism is implemented in CGAL using the template class `CGAL::Handle_for<T>`. To make reference counting optional —a common practice for CGAL kernels, we use a mechanism provided by the template class `SK::Handle<T>`. This template class provides the `SK::Handle<T>::type` type, which is either the template parameter `T` itself —no reference counting, or the `CGAL::Handle_for<T>` type —reference counting. To accommodate both options, the `CGAL::get` function always returns the object of type `T`.

Algebraic numbers. The 3D Spherical Kernel relies on two number types: a field number type —see section 3.2, from the linear kernel —`Linear_kernel::FT`, and an algebraic number of degree two type from the algebraic kernel —`AK::Root_of_2`. This former type —`double` for example, may come with a built-in square root function. If not, the square root is provided by the template class `CGAL::Root_of_2<FT>`, whose underlying representation is $\alpha + \beta\sqrt{\gamma}$ where α, β and γ are of the template type `FT`. To accommodate both options, the `AK::Root_of_2` is retrieved from the template class `CGAL::Root_of_traits<FT>`, namely as `CGAL::Root_of_traits<Linear_kernel::FT>::Root_of_2`.

7.2 3D Spherical Kernel

▷ A circle on a sphere is defined as the intersection of a sphere and a plane. The corresponding class thus stores these two primitives. The class corresponding to circle on a reference sphere inherits from the previous class, so that the reference sphere is retrieved from the data member of the base class.

```
template <class SK>
class Circle_3{
  typedef std::pair<typename SK::Sphere_3,typename SK::Plane_3>  Rep;
  typename SK::template Handle<Rep>::type  base;
};

template <class SK>
class Circle_on_reference_sphere_3
    :public Circle_3<SK>{
  public:
  const typename SK::Sphere_3& reference_sphere() const
    {return CGAL::get(this->base).first;}
};
```

▷ The class used to represent the θ coordinate of a point stores one algebraic number of degree two for the exact value of $\tan\theta$ or $\cot\theta$, together with an index specifying whether tan or cot is used —see section 6.3. The index is of type `SK::Hq_indices`, an enumerated type.

```
template <class SK>
class Theta_rep{
  typedef std::pair<typename SK::Hq_indices,typename SK::AK::Root_of_2>  Rep;
  typename SK::template Handle<Rep>::type  base;
};
```

▷ The class representing a point on a sphere is fully specified by the type of its coordinates, this type being provided by the algebraic kernel. The class corresponding to points on a reference sphere inherits from the previous class, and has two additional members: the reference sphere, and the exact representation of the θ coordinate of the point.

```
template <class SK>
class Circular_arc_point_3{
  typedef typename AK::Root_for_spheres_2_3  Rep;
  typename SK::template Handle<Rep>::type  base;
};

template <class SK>
class Circular_arc_point_on_reference_sphere_3
    :public Circular_arc_point_3<SK>{
  typedef std::pair<typename SK::Sphere_3,typename SK::Theta_rep>  Expanded_rep;
  typename SK::template Handle<Expanded_rep>::type  expanded_base;
  public:
  const typename SK::Sphere_3& reference_sphere() const
    {return CGAL::get(expanded_base).first;}
```

```
  const typename SK::Theta_rep& theta_rep() const
    {return CGAL::get(expanded_base).second;}
};
```

▷ The circular arc class stores the circle supporting the arc and the two endpoints. The circular arc on a reference sphere class inherits from this class, but the types of the circle and that of the endpoints differ. This difference accounts for the second and third template parameters of the base class which are not the default ones.

```
template <class SK,class Circle=typename SK::Circle_3,
           class Endpoint=typename SK::Circular_arc_point_3>
class Circular_arc_3{
  typedef CGAL::Triple<Circle,Endpoint,Endpoint> Rep;
  typename SK::template Handle<Rep>::type base;
};

template <class SK>
class Circular_arc_on_reference_sphere_3
    :public Circular_arc_3<SK,typename SK::Circle_on_reference_sphere_3,
      typename SK::Circular_arc_point_on_reference_sphere_3>{
  typename SK::template Handle<typename SK::Sphere_3>::type  ref_sphere;
  public:
  const typename SK::Sphere_3& reference_sphere() const
    {return CGAL::get(ref_sphere);}
};
```

▷ The segment class is defined by a triple: a line and two endpoints.

```
template <class SK>
class Line_arc_3{
  typedef CGAL::Triple<typename SK::Line_3,typename SK::Circular_arc_point_3,
                       typename SK::Circular_arc_point_3>  Rep;
  typename SK::template Handle<Rep>::type  base;
};
```

7.3 Algebraic Kernel

▷ The class representing equations of planes, of the form $ax + by + cz + d = 0$, stores the 4 coefficients in an array. The template parameter determines the number type used for the coefficients.

```
template <class FT>
class Polynomial_1_3{
  FT rep[4]; // stores a, b, c, d
};
```

▷ The class representing equations of spheres, of the form $(x-a)^2+(y-b)^2+(z-c)^2-R^2=0$, stores the 4 parameters in an array. The template parameter determines the number type used for the parameters.

```
template <class FT>
class Polynomial_for_spheres_2_3{
  FT rep[4]; // stores a, b, c, R^2
};
```

▷ The algebraic class representing a point on a sphere stores one algebraic number of degree two per Cartesian coordinate. The template parameter determines the number type on which are built the algebraic numbers.

```
template <class FT>
class Root_for_spheres_2_3 {
  typename Root_of_traits<FT>::Root_of_2 x_;
  typename Root_of_traits<FT>::Root_of_2 y_;
  typename Root_of_traits<FT>::Root_of_2 z_;
};
```

7.4 An example

The following example piece of code checks whether three spheres intersect in two points.

```
#include <CGAL/Cartesian.h>
#include <CGAL/Algebraic_kernel_for_spheres_2_3.h>
#include <CGAL/Spherical_kernel_3.h>
#include <CGAL/MP_Float.h>
#include <CGAL/Quotient.h>

typedef CGAL::Quotient< CGAL::MP_Float>                      NT;
typedef CGAL::Cartesian<NT>                                  Linear_k;
typedef CGAL::Algebraic_kernel_for_spheres_2_3<NT>           Algebraic_k;
typedef CGAL::Spherical_kernel_3<Linear_k,Algebraic_k>    SK;

int main(){
  //construction of 3 spheres from their centers and squared radii
  SK::Sphere_3 s1(SK::Point_3(0,0,0),2);
  SK::Sphere_3 s2(SK::Point_3(0,1,0),1);
  SK::Sphere_3 s3(SK::Point_3(1,0,0),3);

  SK::Intersect_3 inter;
  SK::Compare_xyz_3 cmp;
  std::vector< CGAL::Object > intersections;
```

```
  inter(s1,s2,s3,std::back_inserter(intersections));

  std::pair<SK::Circular_arc_point_3,unsigned> p1,p2;
  //unsigned integer indicates multiplicity of intersection point
  if (intersections.size() >1){
    //as intersection can return several types (points with multiplicity,
    //circle,...), CGAL::Object and CGAL::assign are used to recover
    //the expected type
    if (CGAL::assign(p1,intersections[0]) && CGAL::assign(p2,intersections[1]))
      std::cout << "Two different intersection points" << std::endl;
    else
      std::cout << "Error" << std::endl;
  }

  //intersection points are sorted lexicographically
  CGAL_assertion(cmp(p1.first,p2.first)==CGAL::SMALLER);
  return 0;
}
```

8 Conclusion

High quality geometric code relies on four virtues: robustness, efficiency, modularity, reusability. Although spheres are amongst the most elementary geometric objects, no virtuous 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, circle arcs and points in 3D. A clear distinction is made between the algebraic and the geometric aspects on 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 developed in a companion paper, provide the first 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] R. Abagyan and M. Totrov. Contact area difference (cad): A robust measure to evaluate accuracy of protein models. *J. Mol. Biol.*, 268, 1997.

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

[3] Oswin Aichholzer, Franz Aurenhammer, Thomas Hackl, Bert Jüttler, Margot Oberneder, and Zbynek Sir. Computational and structural advantages of circular boundary representation. In *Proc. 10th International Workshop on Algorithms and Data Structures (WADS)*, 2007. To appear.

[4] M. V. A. Andrade and J. Stolfi. Exact algorithms for circles on the sphere. *Internat. J. Comput. Geom. Appl.*, 11:267–290, 2001.

[5] K. Bastard, F. Cazals, C. Prevost, and S. Sachdeva. On the selection of best representative conformers. 2007. In preparation.

[6] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979.

[7] 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.

[8] Eric Berberich, Efi Fogel, Dan Halperin, and Ron Wein. Sweeping and maintaining two-dimensional arrangements on surfaces. In *Proceedings of 23rd European Workshop on Computational Geometry*, pages 223–226, Graz, Austria, March 2007. Technische Universitaet Graz.

[9] Jean-Daniel Boissonnat and Franco P. Preparata. Robust plane sweep for intersecting segments. *SIAM Journal on Computing*, 29:1401–1421, 2000.

[10] Christoph Burnikel, Stefan Funke, Kurt Mehlhorn, Stefan Schirra, and Susanne Schmitt. A separation bound for real algebraic expressions. In Friedhelm Meyer auf der Heide, editor, *Proc. 9th European Symposium on Algorithms*, volume 2161 of *Lecture Notes Comput. Sci.*, pages 254–265, 2001.

[11] 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`.

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

[13] CGAL, Computational Geometry Algorithms Library.
`http://www.cgal.org`.

[14] Bernard Chazelle et al. Application challenges to computational geometry: CG impact task force report. Technical Report TR-521-96, Princeton Univ., April 1996.

[15] Bernard Chazelle et al. Application challenges to computational geometry: CG impact task force report. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 407–463. American Mathematical Society, Providence, 1999.

[16] C. Chotia and J. Janin. Principles of protein-protein recognition. *Nature*, 256:705–708, 1975.

[17] CORE Number Library.
`http://cs.nyu.edu/exact/core_pages`.

[18] Pedro M. M. de Castro, Sylvain Pion, and Monique 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 No 6091, Exact and efficient computations on circles in CGAL and applications to VLSI design, `https://hal.inria.fr/inria-00123259`.

[19] Olivier Devillers, Alexandra Fronville, Bernard Mourrain, and Monique Teillaud. Algebraic methods and arithmetic filtering for exact predicates on circle arcs. *Comput. Geom. Theory Appl.*, 22:119–142, 2002.

[20] Scot Drysdale, Günter Rote, and Astrid Sturm. Approximation of an open polygonal curve with a minimum number of circular arcs. In *Proceedings of the 22nd European Workshop on Computational Geometry (EWCG)*, pages 25–28, 2006.

[21] H. Edelsbrunner, M. Facello, and J. Liang. On the definition and the construction of pockets in macromolecules. *Discrete Appl. Math.*, 88:83–102, 1998.

[22] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72, January 1994.

[23] Ioannis Z. Emiris, Athanasios Kakargias, Sylvain Pion, Monique Teillaud, and Elias P. Tsigaridas. Towards an open curved kernel. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 438–446, 2004.

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

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

[26] Efi Fogel and Monique Teillaud. Generic programming and the CGAL library. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*. Springer-Verlag, Mathematics and Visualization, 2006.

[27] Stefan Funke and Kurt Mehlhorn. Look: A lazy object-oriented kernel for geometric computation. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 156–165, 2000.

[28] D. Gregor, J. Järvi, J. Siek, B. Stroustrup, G. Dos Reis, and A. Lumsdaine. Concepts: linguistic support for generic programming in C++. *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming languages, systems, and applications*, pages 291–310, 2006.

[29] Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Michael Seel. An adaptable and extensible geometry kernel. In *Proc. Workshop on Algorithm Engineering*, volume 2141 of *Lecture Notes Comput. Sci.*, pages 79–90. Springer-Verlag, 2001.

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

[31] Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap. Classroom examples of robustness problems in geometric computations. In *Proc. 12th European Symposium on Algorithms*, volume 3221 of *Lecture Notes Comput. Sci.*, pages 702–713. Springer-Verlag, 2004.

[32] John Keyser, Tim Culver, Dinesh Manocha, and Shankar Krishnan. MAPC: a library for efficient and exact manipulation of algebraic points and curves. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 360–369, New York, NY, USA, 1999. ACM Press.

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

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

[35] Chen Li, Sylvain Pion, and Chee 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.

[36] 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.

[37] Sylvain Pion and Monique 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`.

[38] G.D. Rose, A.R. Geselowitz, G.J. Lesser, R.H. Lee, and M.H. Zehfus. Hydrophobicity of amino acid residues in globular proteins. *Science*, 229:834–8, 1985.

[39] Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin. 2D arrangements. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. `http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/packages.html#Pkg:Arrangement2`.

[40] 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.

Contents

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