

Arrangements

Efi Fogel, Dan Halperin*, Lutz Kettner, Monique Teillaud, Ron Wein, and Nicola Wolpert

1.1 Introduction

Arrangements of geometric objects have been intensively studied in combinatorial and computational geometry for several decades. Given a finite collection S of geometric objects (such as lines, planes, or spheres) the *arrangement* $\mathcal{A}(S)$ is the *subdivision* of the space where these objects reside into cells as induced by the objects in S . Figure 1.1 illustrates a planar arrangement of circles, which consists of vertices, edges, and faces: a *vertex* is an intersection point of two (or more) circles, an *edge* is a maximal portion of a circle not containing any vertex, and a *face* is a maximal region of the plane not containing any vertex or edge. For convenience we also introduce two (artificial) vertices in each circle at the x -extreme points splitting the circle into two x -monotone arcs (thus each edge of the arrangement now has two distinct endpoints).

Arrangements are defined and have been investigated for general families of geometric objects. One of the best studied type of arrangements is that of lines in the plane. This means that arrangements may have unbounded edges

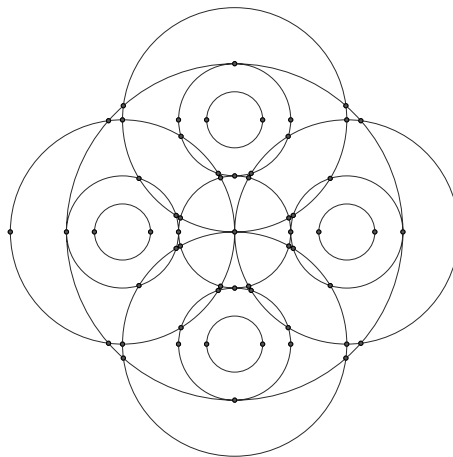


Fig. 1.1. An arrangement of 14 circles in the plane, with 53 faces (one of which is unbounded), 106 edges, and 59 vertices

* Chapter coordinator

and faces. Furthermore, arrangements are defined in any dimension. There are, for example, naturally defined and useful arrangements of hypersurfaces in six-dimensional space, arising in the study of rigid motion of bodies in three-dimensional space.

Written evidence of the study of arrangements goes back to the nineteenth century (see [249]). We notice three periods in the *computational* study of arrangements. From the inception of computational geometry in the seventies till the mid eighties the focus was almost exclusively on the theoretical study of arrangements of unbounded linear objects, of hyperplanes. Many of the results obtained during this period are summarized in the book by Edelsbrunner [130]. The central role of arrangements in computational geometry has fortified in the following period, from the mid-eighties to the mid-nineties, where the theoretical focus has shifted toward arrangements of curves and surfaces. Many of the results obtained in those years are summarized in the book by Sharir and Agarwal [312] and in the survey papers [15] and [196]. From the mid nineties till the time of writing this chapter, a new more practical aspect of the study of arrangements has strengthened, emphasizing implementation and usage. The goal of robustly implementing algorithms for arrangements continues to raise numerous challenging technological and scientific problems. This chapter is devoted to the developments in this applied direction, with an emphasis on curves and surfaces. It should be noted however, that the appearance of a new topic of study in arrangements never replaced previous trends, and to this very day the theoretical study of arrangements of hyperplanes or of arrangements of algebraic curves is a thriving and fruitful domain.

Besides being interesting in their own right, arrangements are useful in a variety of applications. They have been used in solving problems in robot motion planning, computer vision, GIS (geographic information systems), computer-assisted surgery, statistics, and molecular biology, to mention just a few of the application domains. What makes arrangements such a useful structure is that they enable the accurate discretization of continuous problems, without compromising the exactness or completeness of the solution.

When coming to solve a problem using arrangements, we first need to cast the problem in “arrangement terms”. To this end, there is a large arsenal of techniques, most of which are rather simple. They include duality transforms (used to cast problems on point configurations as problems on arrangements of hyperplanes), Plücker coordinates [287], and the so-called locus method that analyzes criticalities in a problem and transforms the criticalities into hypersurfaces in the space where the problem is studied.

The next stage is to understand the combinatorial complexity of the relevant arrangement or of a portion thereof. Quite often, one does not need to construct the entire arrangement in order to solve a problem, and having only a substructure is sufficient (e.g., a single connected component of the ambient space often suffices in solving motion planning problems). This analysis gives a lower bound on the resources required by the algorithms and data structures that will be used in the solution.

Choosing or devising efficient algorithms and data structures to build the required (sub)arrangement is the next step of solving a problem using arrangements. This and the above two steps have been studied for decades.

Then comes the stage of effective implementation of the solution, which has various aspects to it. Sometimes asymptotically efficient algorithms are not necessarily practically the best. Precision and robustness of the solution is a central issue and raises questions such as: (i) how to cope with degeneracies, which are typically ignored in theory under the *general position* assumption; (ii) is the ready-made computer arithmetic sufficient or do we need to use more sophisticated machinery? These are the topics that this chapter focuses on.

The next section of the chapter gives a brief overview of the state-of-the-art in constructing arrangements. In Sect. 1.3 we survey the main advances in exact construction of planar arrangements, focusing mostly on the sweep-line approach, but explaining also what is needed for incremental construction. In Sect. 1.4 we review the software implementation details of these methods. The more recent work on the three-dimensional case is discussed in Sect. 1.5. Stepping away from exact computing is the topic of Sect. 1.6. A tour of implemented applications of curves and surfaces is given in Sect. 1.7. We conclude in Sect. 1.8 with suggestions for further reading and open problems.

1.2 Chronicles

In 1995, CGAL, the *Computational Geometry Algorithms Library*, was founded as a research project with the goal of making the large body of geometric algorithms developed in the field of computational geometry available for industrial applications with correct and efficient implementations [2, 222, 156]. It has since then evolved to an Open Source Project and is now representing the state-of-the-art in implementing computational geometry software in many areas. CGAL contains an elaborate and efficient implementation of arrangements that supports general types of curves.

The recent ECG project, which stands for *Effective Computational Geometry for Curves and Surfaces*, running from 2001 to 2004, extended the scope of implementation research towards curved objects.¹ Arrangements of curves and surfaces were an important theme in this project, and several different approaches to the topic were taken. This body of work is now collected and presented in a uniform manner in the following sections of the current chapter. In this brief section however, we present how the different results evolved.

The first branch of research, undertaken at Tel-Aviv University, Israel, is founded on the CGAL arrangement computation. Originally the CGAL arrangements package supported line segments, circular arcs and restricted types of parabolas. Wein [331] extended the CGAL implementation to ellipses and

¹<http://www-sop.inria.fr/prisme/ECG/>

arcs of conics, where the conics can be of any type. Following the newly emerging requirements from curves on the software, Fogel et al. [167, 166] improved and refined the software design of the CGAL arrangement package. This new design formed a common platform for a preliminary comparison documented in [165] of the different arrangement computation approaches described here. Recently the whole package has been revamped [333] leading to more compact, easier-to-use code, which in certain cases is much faster than the results reported in [165], sometimes by a factor of ten or more. The description in Section 1.4 pertains to this latest design.

The second branch of research, undertaken at the Max-Planck Institute of Computer Science in Saarbrücken, Germany, produces a set of C++ libraries in the project EXACUS (*Efficient and Exact Algorithms for Curves and Surfaces*) [4] as contribution in the ECG project with support for the CGAL arrangement class. The design of the EXACUS libraries is described in Berberich et al. [48]. The theory and the implementations for the different applications behind EXACUS are described in the following series of papers: Berberich et al. [49] computed arrangements of conic arcs based on the improved LEDA [251] implementation of the Bentley-Ottmann sweep-line algorithm [46]. Eigenwillig et al. [140] extended the sweep-line approach to cubic curves. A generalization of Jacobi curves for locating tangential intersections is described by Wolpert [341]. Berberich et al. [50] recently extended these techniques to special quartic curves that are projections of spatial silhouette and intersection curves of quadrics, and lifted the result back into space. The recent extension to algebraic curves of general degree by Wolpert and Seidel [310] exists currently only as a Maple prototype.

The third branch of research, undertaken at INRIA Sophia-Antipolis in France and The National University of Athens in Greece, proposes a design for a more systematic support of non-linear geometry in CGAL focusing on arrangements of curves. This implementation was limited to circular arcs and yielded preliminary results for conical arc, restricted to elliptic arcs [145]. More work on this kernel is in progress. Their approach is quite different from the approach taken in the EXACUS project in that it avoids the direct manipulation of algebraic numbers. Instead, by using algebraic tools like Sturm sequences (that are statically precomputed for small degrees for increased efficiency), they reduce operations such as comparison of algebraic numbers to computing *signs of polynomial expressions*, which can be done with exact integer arithmetic. In addition, this allows the use of efficient filtering techniques [116, 144]. This method is expected to compare favorably against the algebraic numbers used elsewhere.

Efforts towards exact and efficient implementations have been made in the libraries MAPC [226] and ESOLID [225], which deal with algebraic points and curves and with low-degree surfaces, respectively. Both libraries are not complete in the sense that they require surfaces to be in general position.

Computer algebra methods, based on exact arithmetic, guarantee correctness of their results. A particularly powerful and complete method related to

our problems is *cylindrical algebraic decomposition* invented by Collins [101, 100] and subsequently implemented (with numerous refinements). Our approach to curve and curve pair analysis can be regarded as a form of cylindrical algebraic decomposition of the plane in which the lifting phase has been redesigned to take advantage of the specific geometric setting; in particular, to reduce the degree of algebraic numbers in arithmetic operations.

Much of the work described in this chapter relies on having effective algebraic software available, like algebraic number types (as the ones provided by LEDA and CORE), or other algebraic tools. Exactness and efficiency also require the use of adapted number types and filtering techniques. Developments in these areas, which are independent of computing arrangements, are discussed in Chapter 3.

1.3 Exact Construction of Planar Arrangements

Implementing geometric algorithms in a robust way is known to be notoriously difficult. The decisions made by such algorithms are based on the results of simple geometric questions, called *predicates*, solved by the evaluation of *continuous* functions subject to rounding errors (if we use the standard floating-point computer arithmetic), though the algorithms are basically of combinatorial and *discrete* nature. The *exact* evaluation of predicates has become a research topic on its own in recent years, in particular for the case of arrangements of curves.

Algorithms for computing arrangements of curves (or segments of curves) require several operations to be performed exactly. One essential predicate, for example, is the *xy*-comparison that takes two endpoints or intersection points of two segments and compares them lexicographically. This predicate is known to be very sensitive to numerical errors: If the relation given by the comparison test is not transitive, due to erroneous numerical computations, the algorithm may fail.

Let us assume that we are given a set \mathcal{B} of planar *x*-monotone curves that are pairwise disjoint in their interior — that is, two curves in \mathcal{B} may have a common endpoint but cannot have any other intersection points. The *doubly-connected edge list* (DCEL for short) is a data structure that allows a convenient and efficient representation of the planar subdivision $\mathcal{A}(\mathcal{B})$ induced by \mathcal{B} . We represent each curve using a pair of directed *halfedges*, one going from the left endpoint (lexicographically smaller) of the curve to its right endpoint, and the other (its *twin* halfedge) going in the opposite direction. The DCEL consists of three containers of records: *vertices* (associated with planar points), *halfedges*, and *faces*, where halfedges are used to separate faces and to connect vertices. We store a pointer from each halfedge to its *incident face*, which is the face lying to its left. In addition, every halfedge is followed by another halfedge sharing the same incident face, such that the destination vertex of the halfedge is the same as the origin vertex of its following halfedge.

Effective Computational Geometry for Curves and
Surfaces

Boissonnat, J.-D.; Teillaud, M. (Eds.)

2006, XII, 344 p., Hardcover

ISBN: 978-3-540-33258-9