TEL-AVIV UNIVERSITY
RAYMOND AND BEVERLY SACKLER
FACULTY OF EXACT SCIENCES
SCHOOL OF COMPUTER SCIENCE

Controlled Perturbation for Arrangements of Circles

Thesis submitted in partial fulfillment of the requirements for the M.Sc. degree in the School of Computer Science, Tel-Aviv University

by

Eran Leiserowitz

The research work for this thesis has been carried out at Tel-Aviv University under the supervision of Prof. Dan Halperin

July 2003

Acknowledgments

I deeply thank Prof. Dan Halperin for his great help during this research. I wish to thank him for his patience and support in many hours of discussions. I would also like to thank all the fellow students in the Geometry lab for their assistance in the course of work. Finally, I would like to thank my family and Maya for all their support.

Contents

Chapter 1

Introduction

Computational Geometry algorithms are often designed and proved to be correct under the assumption of the "real RAM" computation model. This model assumes that the computation is done using unlimited precision real numbers, on a computer with random-access memory. Many times, the computation cost of using an exact number type turns out to be too expensive. Simply exchanging the exact number type to a finite precision number type (e.g., machine float) could lead to fast, yet unstable programs. In the next section we will demonstrate how geometric algorithms are prone to error when implemented using finite precision arithmetic.

Furthermore, Computational Geometry algorithms often assume general position of the input (e.g., no three lines intersect in a common point, no three points are collinear, etc.). This assumption simplifies both the theoretical analysis of the algorithm, and its practical implementation. However, one cannot assure that the real input will always be in general position. Thus, both the analysis and the implementation of the algorithm, should also take into account the degenerate cases (when the input is not in general position, it is said to be degenerate). If one wishes to use finite precision arithmetic (to achieve fast running time), then even if the input *is* in general position, round-off errors may cause the algorithm to fail.

In the next section, we give several examples that illustrate the different robustness issues that can arise in geometric algorithms. In Section 1.2 we present our approach to robust handling of arrangements of circles (namely the subdivision of the plane into vertices, edges and faces induced by the circles). Our method allows the construction of the arrangement using finite precision arithmetic. It also eliminates all the degeneracies, thus resulting in a less complicated source code, and faster programs.

1.1 Robustness Issues in Computational Geometry

We start with a classic example, Ramshaw's braided lines, in which naively applied floating-point arithmetic can yield results which contradict elementary geometric axioms. Consider two lines, $l_1 : y = 4.3x/8.3$ and $l_2 : y = 1.4x/2.7$ (Figure 1.1). Using floating-point arithmetic with mantissa of two decimal digits causes the two lines to behave like step functions which intersect more than once, thus contradicting a basic axiom of planar geometry. Further details on this example are given in [33].

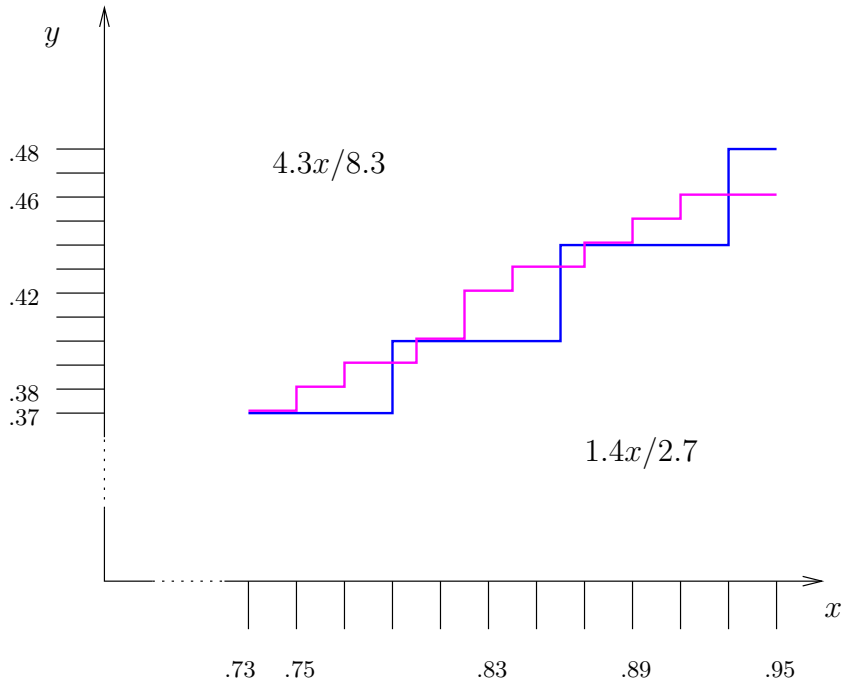


Figure 1.1: Ramshaw's braided lines.

For the second example (based on the one given in [36]), consider a box and a plane H (Figure 1.2 (a)). We wish to classify the corners of the box to three groups: (i) above H, (ii) on H, and (iii) below H.

If the plane H is almost parallel to the top side of the box, and very close to it, then the use of finite precision arithmetic could lead to false classification, in which one pair of opposite corners is classified as below H and the other pair is classified as above H (Figure 1.2 (b)). This is a violation of Euclidean geometry, in which a pair of distinct planes can intersect in at most one line.

It appears that in the field of Computational Geometry, robustness problems due to precision issues are somewhat more difficult to handle than in other fields in computer science. The reason lies in the fact that many Computational Geometry algorithms rely on data structures that combine both *topological* and *geometric* data. For example, consider a set of line segments in the plane (Figure 1.3 (a)). The

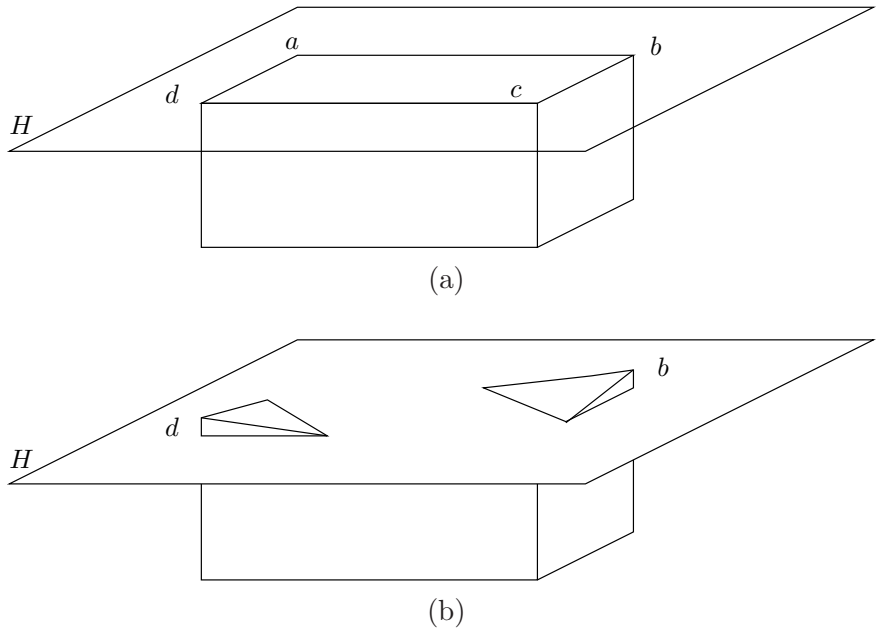


Figure 1.2: (a) A box and a plane. (b) False classification of the corners.

topological data is the planar graph in which each intersection point becomes a vertex, and two vertices share an edge, if their underlying intersection points lie on the same line and there is no other intersection point on that line between them (Figure 1.3 (b)). The geometric data in this case could be the underlying line equations and the segments end-points. Errors during the computation of the intersection points may lead to the construction of a graph which is inconsistent with the real input lines, which in turn could lead to incorrect behavior of algorithms that use this graph.

Computational Geometry algorithms depend on predicates. Predicates are the building blocks of the algorithm. Every predicate has an underlying mathematical expression which it evaluates. Usually we are only concerned with the sign the expression returned by the predicate. Correct predicates are crucial to the correctness of geometric algorithms and their implementation. In order for the predicate to always give correct answers, with no assumptions on the input, we have to use exact computation.

1.2 Our Approach

In our scheme, to avoid the use of exact computation during the evaluation of the predicates, we will perturb the geometric objects (circles, in our case) such that we can certify correct results of the predicates even when we use finite precision arithmetic.

A degeneracy occurs when a predicate evaluates to zero. The goal of our perturbation scheme is to cause all the predicates that we use during the algorithm to evaluate

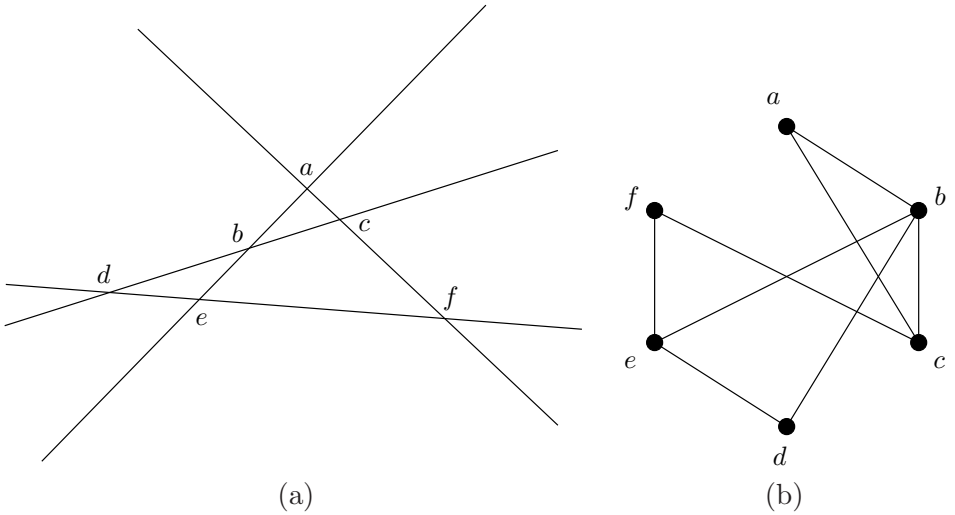


Figure 1.3: (a) An arrangement of line segments. The intersection points are labeled with letters. (b) The corresponding graph.

sufficiently far away from zero so that our finite precision arithmetic could enable us to safely determine whether they are positive or negative. Hence, while certifying the correctness of the predicates, we are also eliminating all the degeneracies.

Arrangements are widely used in Computational Geometry [1, 21]. Throughout the years, algorithms for building different kind of arrangements have been proposed. In Chapter 2 we describe several such algorithms (i.e., [13, 24, 38]). Once an arrangement has been constructed, it can be used to perform many operations, such as *point location* (Section 6.3), finding the intersection or union of objects whose boundaries it represents, and more. Many geometric algorithms use arrangements as a data structure on which they operate (e.g., motion planning algorithms). Hence, it is an important tool in Computational Geometry.

In the case of arrangements of circles (namely the subdivision of the plane into vertices, edges and faces induced by the circles), general position of the input means that there is no outer or inner tangency between two circles, and that no three circles intersect at a common point (see Figure 1.4 for a degenerate arrangement).

While building the arrangement in an incremental fashion (that is, adding one circle at a time), we will check if there is a potential degeneracy induced by the newly added circle, and if so, we will move that circle, so no degeneracies will occur. The main idea is to carefully relocate the circle — move the circle enough to avoid the degeneracies, but not too much. Depending on the precision of the machine floating-point representation, and some properties of the arrangement to be handled, we determine a bound δ on the magnitude of the perturbation, namely, we guarantee that any input circle will not be moved by a distance greater than δ. Since the perturbed version of the input (on which the algorithm will work) is known, our scheme achieve the property of *backward stability*. Figure 1.5 shows a few possible

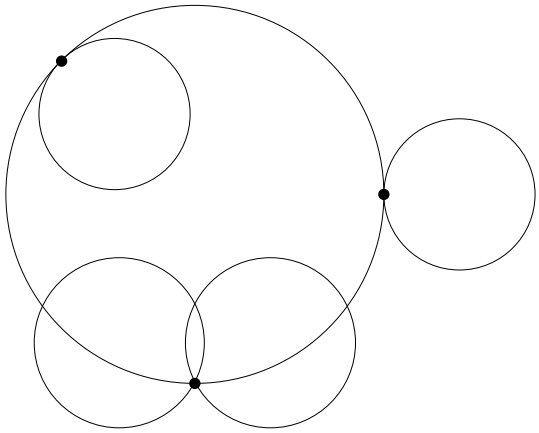


Figure 1.4: Arrangement of circles with several degeneracies.

results of our perturbation algorithm, applied to the arrangement shown in Figure 1.4.

Such a perturbation scheme, as was described above, could be useful for the following reasons:

- Floating-point arithmetic is usually supported by hardware, making computations very fast. If the built-in "double" number type is insufficient, we can use a number type with greater precision (e.g., LEDA's "Bigfloat" [28]). Notice that the length of the mantissa is fixed prior to the beginning of the program. This is different than using a "real" number type (e.g., LEDA's "real" or CORE's "Expr" [26]) which could require arbitrary precision.

- Degeneracies are eliminated (general position of the input is indeed achieved, even for the fixed, limited precision), consequently an algorithm is made easier to analyze and implement. This reduces the need to handle many special cases. The number of special cases induced by degeneracies can be in the dozens already for simple algorithms.

- The geometric objects retain their geometric structure. That is, the circles are not transformed into pseudo-circles (in contrast with, for example, snap rounding — see below). Thus, all the geometric rules and axioms regarding the geometric objects (circles, in our case) will still be valid.

- Implementations using exact arithmetic with floating-point filtering (Section 2.2), can be sped up, since the perturbation will cause the predicates to be evaluated using the floating-point filters, thus avoiding the use of exact computation.

- Using a multiprecision floating-point arithmetic library, we can set the number type precision such that the size of the perturbation will be as small as we wish. That is, if the user of the algorithm requires certain accuracy (no geometric object shall be moved by a measure greater than a predefined δ), we can deduce the floating-point precision needed that will satisfy such δ demand.

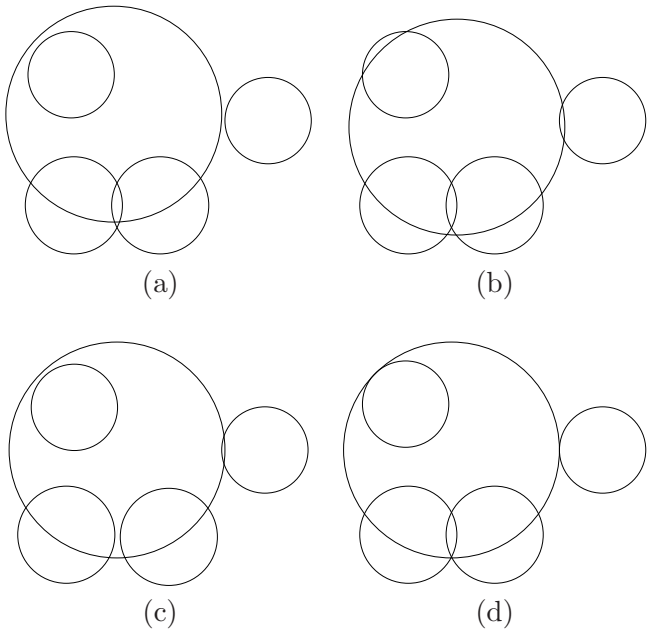


Figure 1.5: (a) - (c) A few possible results of our perturbation algorithm applied to the arrangement shown in Figure 1.4. For illustration purposes, the magnitude of the perturbation is rather large. (d) The real result of our perturbation algorithm (as was computed using the software that we developed) — the perturbation is too small to be discernible in the figure.

The main drawback of our scheme, is that the objects are actually moved from their original placement. Still, in many situations, the original input data is inaccurate to begin with (due to, for example, measuring errors or approximate modeling), so the damage incurred by perturbing slightly is negligible.

It should be noticed that the predicates that arise in the construction of arrangements of circles include expressions that contain division and square-root operations. Those operation are usually more difficult to handle robustly than addition, subtraction and multiplication. Furthermore, implementing such predicates using exact number types would require the representation of irrational numbers.

1.3 Thesis Outline

Robustness and precision issues in Computational Geometry have been intensively studied in recent years. In the next chapter, we survey some of the work in this area. We also explain the new contribution of this work, with respect to previous works on *Controlled Perturbation*, namely, obtaining good *resolution bound*, which is a key element in Controlled Perturbation.

An overview of the perturbation scheme is given in Chapter 3. There, we establish the formal terminology that we will use throughout this thesis. In particular, we define the resolution bound, and the *perturbation bound*. We also give a sketch of the perturbation algorithm. We conclude that chapter by summarizing the resources required by the algorithm.

In Chapter 4 we explain how to compute the perturbation bound, δ. During the perturbation, we can guarantee that an input circle will not be moved by a distance greater than δ. We start by defining the relevant degeneracies in an arrangement of circles, continue by defining the notion of *forbidden regions*, and finally, we show how to bound the total area of those forbidden regions. Notice that in this chapter, we assume that the resolution bound (Chapter 5) is already given.

The main contribution of the thesis is described in Chapter 5. First, we define the predicates that allow us to verify that no degeneracies exist in the arrangement. Next, for each predicate, we derive the resolution bound needed to certify the correctness of the predicate (recall that all the computations are done in fixed precision arithmetic). We conclude this chapter with a numerical example.

Algorithmic details are given in Chapter 6. We present an efficient perturbation algorithm, which runs in expected $O(n^2 \log n)$ time for an arrangement of n circles. Next, we describe how to construct the *doubly-connected edge list* structure (DCEL), which allows us to maintain the *topological* information of the subdivision and enhance it with the *geometric* information (its planar embedding). The DCEL structure is very useful, as a basic data structure upon which many other geometric algorithm can build (e.g., we have implemented an algorithm for finding the union of the circles). Finally, we present a simple *point location* strategy suitable for our DCEL implementation.

Experimental results are reported in Chapter 7, along with some implementation details. The perturbation algorithm and the DCEL construction have been implemented as a set of C++ classes. We have experimented with different types of inputs. Some data sets are highly degenerate, and others contain a huge number of circles (several thousands). We report the running times and the perturbation statistics. We also interpret these results.

In Chapter 8 we discuss and highlight some of the key issues of our work. These issues are of a special interest for those who wish to further use the controlled perturbation scheme. Among the discussed issues, are the importance of the resolution and perturbation bound, and the number type selection.

Chapter 9 gives a brief summary of this work. We also point out possible future research directions.

In appendix A we describe a simple generalization of our perturbation scheme, to the case of arrangements of circular arcs.

In appendix B we further demonstrate our main ideas, by describing how to compute the resolution bound for the `incircle` predicate, which is used in many algorithms, among them the construction of Delaunay triangulations.

Chapter 2

Related Work

Robustness and precision issues have been intensively studied in Computational Geometry in recent years [33, 40]. In this chapter we give a brief survey of the different approaches to dealing with robustness issues. Each approach has its advantages and disadvantages. We focus on three specific topics: (i) exact computation, (ii) filtering methods, and (iii) finite precision approximation methods. The method that we present, controlled perturbation, is a finite precision approximation method.

2.1 Exact Computation

A prevailing approach to overcoming robustness problems in Computational Geometry is to use exact computation [12, 28, 41]. If all the computation is carried out exactly, it is possible to retrieve the true underlying topological structures, thus avoiding the robustness problems. Yet, this approach does not eliminate the "general position" requirement. That is, both the theoretical algorithm and the practical implementation should take degenerate input into account. On the implementation side, this could lead to many special cases, which eventually lead to longer and very complicated source code. On the theory side, the designer of the algorithm should verify that all the geometric theorems and lemmas that he/she is using are also valid in the degenerate case (sometimes, this can be done by symbolically perturbing the input, and proving that the result is also valid for the original input).

In order to allow an exact computation of a certain algorithm, there is a need for a number type that would behave exactly as the mathematical number type that it represents. For many problems in Computational Geometry, a rational number is sufficient to allow for exact solutions. Thus, much effort has been invested to devise an efficient implementation of such a number type (e.g., LEDA's rational number type [28]). Yet, there are also problems where rational numbers are insufficient (e.g., in the construction of arrangements of circles, the coordinates of the intersection points can have irrational values). Hence, the LEDA [28] and CORE [26] libraries both provide algebraic number types. Both implementations are based on the theory of

separation bounds [6] (in Chapter 8 we give a brief description of this number type). In [5], modular arithmetic is used in combination with single precision floating-point arithmetic for sign evaluation of determinants. Sign evaluation of determinants is useful in many algorithms (e.g., the *orientation* and *in_sphere* predicates use such sign evaluation).

The main drawback of the exact computation approach is the computation time. The usage of multiprecision number type can lead to expensive computation cost. The computation can be accelerated by a lazy evaluation scheme, usually referred to as *filtering*. We describe various filtering methods in the next section.

2.2 Filtering

As stated above, when applied naively, exact computation can considerably degrade the performance of a program. One of the possible solutions to this problem is to use *filtering*. In [7, 9, 14, 27, 34], the filtering is done at the level of the number type. That is, a predicate is evaluated using exact computation *only* if it cannot be correctly evaluated using finite precision arithmetic. Thus, for most computations the algorithm will only use finite precision arithmetic, and in the few occasions that the finite precision arithmetic is insufficient, exact computation will be employed. In [38], high level filtering is proposed as a means to speed up exact computation, as described below.

Karasick et al. [27] present a filtering method to evalute the sign of a predicate. Such signs evaluations (namely, in the *in_circle* and *orientation* predicates) are needed to implement the Guibas-Stolfi Delaunay triangulation algorithm [18]. The initial implementation of that algorithm, turned out to be 10,000 times slower when exact rational arithmetic was used instead of the standard floating-point arithmetic. To overcome this huge gap, they use the following method: given a matrix A with integer entries, they create the matrix A' of integer *intervals*, that have lower-precision endpoints. If the interval $|A'|$ does not span zero, then both $|A|$ and $|A'|$ have the same sign; otherwise — increase the precision of the endpoints of A' and evaluate the sign of $|A'|$ again. In the worst case, A' would have endpoints with the same precision as A and no advantage is gained.

Fortune and Van Wyk [14] describe static-analysis techniques that reduce the performance cost of exact *integer arithmetic* used to implement geometric algorithms. They use double-precision floating-point variables to represent a multiprecision integer. They have applied their techniques on a number of examples, such as line-segment intersections in two dimensions, Delaunay triangulations, and a three-dimensional boundary-based polyhedral modeller. In general, their techniques are appropriate for algorithms that use primitives of relatively low algebraic total degree. The techniques have been packaged in a C++ preprocessor. However, adapting a real number RAM version of an algorithm to an integer version can be a difficult task, since it must be possible to represent the geometric data using only integers. In order to minimize the

bit-length required to evaluate the primitives exactly, the programmer must define the primitives carefully.

Shewchuk [34] presents a fast software-level algorithms for exact addition and multiplication of arbitrary precision floating-point values. He then proposes a technique for adaptive-precision arithmetic, that is, speeding those algorithms when multiprecision computation is done only to satisfy some error bounds. He demonstrates this technique by an implementation of several common geometric predicates whose required degree of accuracy depends on their inputs. These algorithms are based on floating-point arithmetic that uses radix two and exact rounding (e.g., the IEEE 754 standard). Publicly available C code is given for the 2D and 3D orientation and incircle tests.

Burnikel, Funke and Seel [7] present an efficient numerical approach to deal with precision inaccuracies arising in the implementation of geometric algorithms. They provide an error analysis strategy for floating-point arithmetic, which divides the computation of the error bound into a static part and a dynamic part — the static part can be computed before runtime without any knowledge of the actual values to be used, and the dynamic part is computed during runtime. They combine their error analysis method with arbitrary precision packages (namely LEDA [28]) to develop an expression compiler EXPCOMP, which supports all common operations $+, -, \cdot, /, \sqrt{}$. The resulting programs show a good runtime behavior.

Wein [38] describes a method to avoid the use of an exact algebraic number type when building an arrangement of conic arcs, by high level filtering. When constructing a geometric object he keeps track of all the steps that led to its construction. That history is used to answer predicates efficiently. Using this representation most of the arrangement vertices are only computed approximately at first. Then, only in the case of ambiguity, the computations are refined using the construction history of the relevant objects. Since those cases are relatively rare, the resulting algorithm is efficient. His ideas have been implemented as part of the CGAL library [11]. An alternative approach for the construction of arrangements of conics is given in [3].

2.3 Finite Precision Approximation

An alternative approach aims to compute robustly with finite precision arithmetic, often by approximating or perturbing the geometric objects [13, 20, 22, 24, 25, 29, 35, 36, 37].

Fortune and Milenkovic [13] analyze the behavior of two algorithms for constructing line arrangements, a sweep algorithm and an incremental algorithm. Each algorithm produces an arrangement realized by a set of pseudo-lines. A bound on the difference between the pseudo-lines and the original lines is given.

Guibas et al. [20] give a general framework, called *Epsilon Geometry*, designed to build robust geometric algorithms out of imprecise geometric primitives. Their

method combines the techniques of interval arithmetic and backward error analysis, along with some geometric reasoning. Their algorithms compute an exact solution for a perturbed version of the input, and return a bound on the size of this implicit perturbation (based on the size of the rounding errors observed during the computation). Notice that in contrast with our perturbation scheme, the actual perturbed version of the input (for which the solution is correct) is not known, and there is only a single number which provides a bound on the distance between the exact and the perturbed version. Also, we use interval arithmetic to bound the floating-point errors, whereas they use it as a reasoning tool, to combine the results of several predicates (notice, that they are using a different approach in the estimation of roundoff errors).

Hobby [24] presents the *snap rounding* paradigm. His algorithm is based on the Bentley-Ottmann sweep line algorithm for finding the intersection of line segments. The plane is tiled with a grid of units squares, called *pixels* (or *tolerance squares*), such that each pixel is centered at a point with integers coordinates. Each vertex in the arrangement is snapped to the center point of the pixel that contains it, thus potentially transforming the segments of the arrangement into polygonal chains. The transformed arrangement preserves certain topological properties of the original arrangement: The rounding can be regarded as a continuous process of deforming the segments into polygonal chains such that no vertex of the arrangement ever crosses through a segment. A variant of this method, *iterated snap rounding*, is given in [22].

Sugihara [36] describes a topology-oriented method, for designing numerically robust geometric algorithms. This method gives higher priority to the consistency of the topological structures, than to the accuracy of the numerical computations. Thus, inconsistency never arises, even when large numerical errors occur. An example for this method is the problem of cutting a convex polyhedron by a plane. Let Π be a convex polyhedron in three dimensions, and H be a plane. For simplicity, assume that no vertex of Π is contained in H. Denote by G the planar graph implied by the vertices and edges of Π. To cut Π by H, we classify the vertices of G in two groups: V_1 — the set of vertices that are above H, and V_2 — the set of vertices that are below H. Since Π is a convex polyhedron the following property should hold: the subgraph of G induced by V_1 and the one induced by V_2 are each connected. This property is then conserved during the algorithm. Numerical values are considered only when they do not contradict this property. In our method, in contrast with the topology-oriented method, we give a bound on the numerical deviation of our structure from the exact structure.

More methods for handling imprecise geometric computations are surveyed in [33]. Controlled perturbation is a method of this type. A general discussion on the properties of floating-point arithmetic is given in [16].

2.4 Previous Work on Controlled Perturbation

The perturbation scheme that we follow, controlled perturbation, was first presented by Halperin and Shelton [23] as a method to speed up molecular surface computation (where atoms of the molecule are modeled by spheres). The use of exact computation turned out to be too slow for real time manipulation, so a finite precision method was needed. Controlled perturbation was devised to handle the robustness issues caused by the use of finite precision arithmetic, and to remove all the degeneracies. It should be noticed, that the molecular models are approximate to begin with, thus the perturbation could be easily justified. The time complexity of their method is linear in the number of spheres in the input.

Raab [32] applied controlled perturbation to arrangements of polyhedral surfaces needed for computing swept volumes. A swept volume is defined as the geometric space occupied by an object moving along a trajectory in a given time interval. The swept volume application computes the boundary of a collection of polyhedra and performs its vertical decomposition. Notice that those arrangements require complex calculations in order to achieve a good *perturbation bound* (Chapter 4).

In [32] (as in [23]), the *resolution bound* (Chapter 5) is assumed to be given. *The resolution bound is a key element in the scheme.* In this work we describe a method for obtaining good resolution bounds (Chapter 5), which we anticipate will lead to a better understanding of the method and will open the way to applying the method in other settings.

Most recently, controlled perturbation was applied to line segments [31]. Again, the resolution bound was assumed to be given.

Chapter 3

Overview of the Scheme

In this chapter, we give an overview of the controlled perturbation scheme. We describe the main concepts, on which we expand in later chapters. Although the ideas that we present here could have been described in a more general setting, we concentrate, for ease of exposition, on arrangements of circles.

3.1 The Main Concepts

For an input circle C_i, our algorithm will output a copy C_i' with the same radius but with its center possibly perturbed. We define $\mathcal{C}_j$ as the collection of circles $\{C_1, \ldots, C_j\}$, and $\mathcal{C}_j'$ as the collection of circles $\{C_1', \ldots, C_j'\}$.

The input to our algorithm is the collection $\mathcal{C} = \mathcal{C}_n$ of n circles, each circle C_i is given by the coordinates of its center X_i, Y_i and its radius R_i; we assume that all the input parameters are representable as floating-point numbers with the given precision. The input consists of three additional parameters: (i) the machine precision p, namely the length of the mantissa in the floating-point representation, (ii) an upper bound on the absolute value of each input number X_i, Y_i and R_i, and (iii) Δ — the maximum perturbation size allowed.[1] The perturbation scheme transforms the set $\mathcal{C}$ into the set $\mathcal{C}' = \mathcal{C}_n'$.

We build the arrangement in an incremental fashion, and if there is a potential degeneracy while adding the current circle, we perturb it, so no degeneracies will occur. Once the j-th step of the procedure is completed, we do not move the circles in $\mathcal{C}_j'$ again. We next describe the two key parameters that govern the perturbation scheme, the resolution bound and the perturbation bound. A formal definition of these parameters will be given in the subsequent chapters, together with the way we derive them.

[1] The exact size of Δ depends on the specific application of the perturbed arrangement. Further details are given below.

Resolution Bound

A degeneracy occurs when a predicate evaluates to zero. The goal of the perturbation is to cause all the values of all the predicate expressions (that arise during the construction of the arrangement of the circles) to become significantly non-zero, namely to be sufficiently far away from zero so that our limited precision arithmetic could enable us to safely determine whether they are positive or negative.

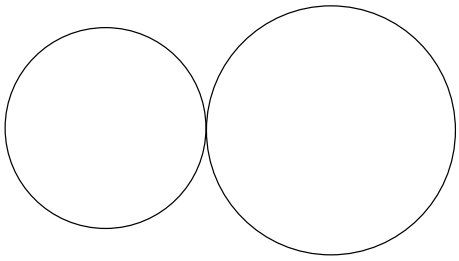


Figure 3.1: Outer tangency — two circles (bounding interior-disjoint disks) intersect in a single point.

The degeneracies that arise in arrangements of circles have a natural geometric characterization as *incidences*. For example, in *outer tangency* (Figure 3.1), two circles intersect in a single point. In our scheme we transform the requirement that the predicates will evaluate to sufficiently-far-from-zero values into a geometric distance requirement.

Outer tangency between C_1 and C_2 occurs when

$$[(X_1 - X_2)^2 + (Y_1 - Y_2)^2]^{\frac{1}{2}} = R_1 + R_2 \,.$$

We will look for a distance $\varepsilon > 0$ such that when we move one circle relative to the other ε away from the degenerate configuration, we could safely determine the sign of the predicate with our limited precision arithmetic, that is we look for a relocation (X_2', Y_2') of the center of C_2 such that

$$|[(X_1 - X_2')^2 + (Y_1 - Y_2')^2]^{\frac{1}{2}} - (R_1 + R_2)| \geq \varepsilon \,.$$

This is a crucial aspect of the scheme: the transformation of the non-degeneracy requirement into a separation distance. We will call the bound on the minimum required separation distance, the *resolution bound* and denote it by ε (it would have been also suitable to call it a separation bound, but we use resolution bound to avoid confusion with separation bounds of exact algebraic computing). If the separation distance is less than ε, then there is a *potential degeneracy* (we use this term, since we do not know if the degeneracy really exists). Deriving a good resolution bound is a central innovation in this work. Previously (e.g., [23]) it was assumed that these bounds were given, and in the experiments crude (high) bounds were used. The bound ε depends on the size of the input numbers (center coordinates and radii) and the machine precision. It is independent of the number n of input circles.

The only modification to the input that our scheme allows is the relocation of the center of the currently inserted circle C_i. This is a choice of convenience which simplifies the analysis and implementation of the scheme. Other choices (in other settings) are described in [31, 32].

Perturbation Bound

Suppose indeed that ε is the resolution bound for all the possible degeneracies in the case of an arrangement of circles for a given machine precision. When we consider the current circle C_i to be added, it could induce many degeneracies with the circles in $\mathcal{C}'_{i-1}$. Just moving it by ε away from one degeneracy may cause it to come closer to other degeneracies. This is why we use a second bound δ, the *perturbation bound* — the maximum distance by which we will perturb the center of any of the circles away from its original placement. The bound δ depends on ε, on the maximum radius of a circle in $\mathcal{C}$, and on a *density* parameter k of the input which bounds the number of circles that are in the neighborhood of any given circle and may effect it during the process, $k \leq n$ (a formal definition of k is given below).

We say that a point q is a valid placement for the center of the currently handled circle C_i, if when moved to q this circle will not induce any degeneracy with the circles in $\mathcal{C}'_{i-1}$. The bound δ is computed such that inside the disk D_δ of radius δ centered at the original center of C_i, at least half the points (constituting half of the area of D_δ) will be valid placements for the circle (Figure 3.2). This means that if we choose a point uniformly at random inside D_δ to relocate the center of the current circle, it will be a valid placement with probability at least $\frac{1}{2}$.

We argue as if the disk from which we sample constitutes a continuous region, whereas it is in fact a discrete set of points (floating-point values inside the disk). However, this gap is easily settled by observing that the forbidden regions are regularly shaped: disks and annuli, whose width (radius in case of a disk) is orders of magnitude larger than the resolution of the floating-point grid—see Chapter 5 where these values (ε_i) are derived. Therefore, the discrepancy between the continuous forbidden regions and their discrete representation is negligible. In any case it is a small constant independent of the input size n. In order for the probability of success (finding a valid placement) to be at least $\frac{1}{2}$ we need to fine-tune the computation of δ slightly. For simplicity of exposition we omit this technical factor. The analysis or results reported below are unaffected by this omission, since we only rely on the fact that there is a large constant probability of success (which might be less than $\frac{1}{2}$ but very close to $\frac{1}{2}$).

After the perturbation, the arrangement $\mathcal{A}(\mathcal{C}')$ is degeneracy free. Moreover, $\mathcal{A}(\mathcal{C}')$ can be robustly constructed with the given machine precision. The perturbation algorithm should not be confused with the actual construction of the arrangement. It is only a preprocessing stage. However, it is convenient to combine the perturbation with an incremental construction of the arrangement, as we describe below in

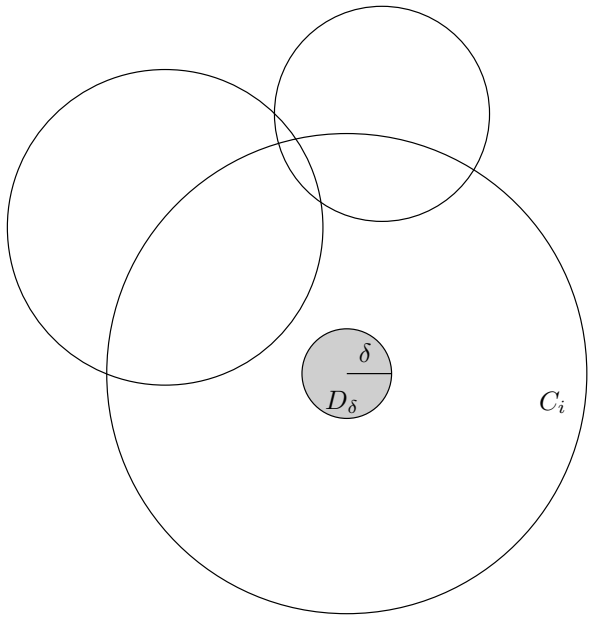


Figure 3.2: The shaded area is the disk D_δ in which at least half the points will be valid placements for the center of C_i (for clarity, D_δ is shown to be very large).

Section 6.1.

An alternative view of our perturbation scheme is as follows. We look to move the centers of the input circles slightly from their original placement such that when constructing the arrangement $\mathcal{A}(\mathcal{C}')$ while using a fixed precision (floating-point) filter, the filter will always succeed and we will never need to resort to higher precision or exact computation.

The additional parameters used in our analysis are described next.

Density Parameter

As stated above, in order to compute the perturbation bound δ, we use a density parameter k. Let Δ be the maximum perturbation that we are willing to allow (the exact size of Δ depends on the specific application of the perturbed arrangement and we assume that it is given to us by the user). If the bound δ that we obtain is greater than Δ then we must resort to higher precision. Each $C_i \in \mathcal{C}$ induces an annulus (i.e., the region sandwiched between two concentric circles), centered at the center of C_i, with radii $max(0, R_i - \Delta)$ and $R_i + \Delta$. We define k as the maximum number of such annuli intersecting a single annulus (Figure 3.3). Notice that in the worst case $k = n - 1$.

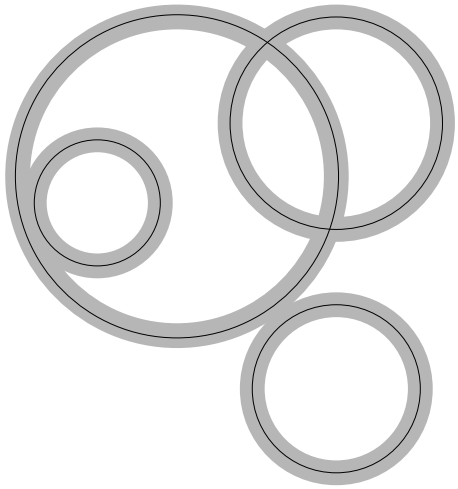


Figure 3.3: Each $C_i \in \mathcal{C}$ induces an annulus (the shaded area) bounded by two circles, centered at the center of C_i, with radii $max(0, R_i - \Delta), R_i + \Delta$. We define k as the maximum number of such annuli intersecting a single annulus, in this example $k = 3$.

Input Bound

In the computation of the bound ε we assume that there is an upper bound M on the size of all the parameters of the circles in $\mathcal{C}$ (center coordinates or radius).

During the perturbation, the center of a circle may move by an amount of at most Δ (again, Δ is the maximum perturbation that we are willing to allow and it is given as part of the input). Therefore the absolute value of the input coordinates for all circles can be at most $M - \Delta$.

Minimum Distance Between the Centers of Intersecting Circles

In Section 5.4 we use the parameter ξ, which defines the minimum distance between the centers of intersecting circles after the perturbation. This parameter simplifies the derivation of the resolution bound. Further details are give in Section 5.4.

Exact vs. Approximate Intersection Point

Intersection points play an important role throughout this thesis. We will deal with circle-circle intersection points and line-circle intersection points. Sometimes, we will refer to the *exact* intersection point, and sometimes to the *approximate* intersection point. As implied by the name, the exact intersection point, is the point that we would have computed, if we were to use exact computation. Since we are not using exact computation, we can only compute the approximate intersection point, which is less

than some precomputed distance ω (see Chapter 5) away from the corresponding exact intersection point. Hence, we can inflate a disk of radius ω around the exact (resp. approximate) intersection point that would contain the corresponding approximate (resp. exact) intersection point (Figure 3.4).

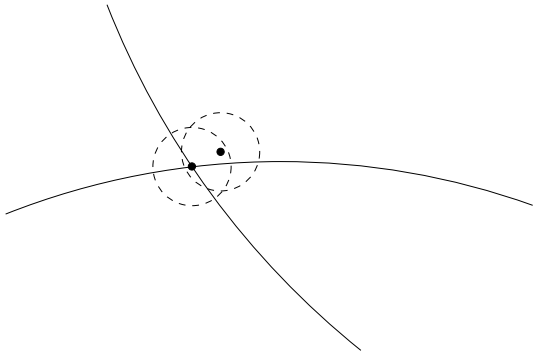


Figure 3.4: We can inflate a disk of radius ω around the exact (resp. approximate) intersection point that would contain the corresponding approximate (resp. exact) intersection point. These disks are illustrated as dashed circles.

3.2 Sketch of the Algorithm

Given a collection $\mathcal{C}$ of n circles $C_1, \ldots, C_n$, the algorithm for perturbing $\mathcal{C}$ proceeds as follows:

1: compute ε,δ and set $\mathcal{C}'_1 = \{C_1\}$.
2: **for all** $C_i, i = 2 \ldots n$ **do**
3: set $C'_i = C_i$.
4: check C'_i against all previously handled circles in $\mathcal{C}'_{i-1}$, and circles' intersections points. If there are no potential degeneracies then go to step 7.
5: set $C'_i = C_i$ (restore the original position).
6: move the center of C'_i randomly, a distance $d \leq \delta$ and go to step 4.
7: $\mathcal{C}'_i := \mathcal{C}'_{i-1} \cup \{C'_i\}$.
8: report the circles in $\mathcal{C}'_n$.

Details regarding an efficient implementation of the algorithm are given in Section 6.1. We quote the result summarizing the resources required by the algorithm.

Theorem 1 *Given a collection $\mathcal{C}$ of n circles, the perturbation algorithm which allows for the robust construction of the degeneracy-free arrangement $\mathcal{A}(\mathcal{C}')$ runs in total expected $O(n^2 \log n)$ time.*

Notice that the worst-case complexity of the arrangement is $\Theta(n^2)$. In the standard "Real RAM" model, computing an arrangement of circles, using the incremental construction algorithm takes $O(n\lambda_4(n))$ time. We next explain how to compute δ (Chapter 4) and ε (Chapter 5).

Chapter 4

The Perturbation Bound

In this chapter we compute an upper bound δ on the maximum necessary perturbation for a single circle. The bound δ depends on the resolution bound ε, the maximum radius of an input circle and the density parameter k.

The resolution bound ε is the distance that we need to separate two circles by, or a circle and an intersection point of two other circles, or the centers of two circles, to avoid a *potential degeneracy* (Figure 4.1). In the next chapter we will show how to compute a good bound on the resolution parameter. In this chapter we show how to determine δ assuming that $\varepsilon > 0$ is given.

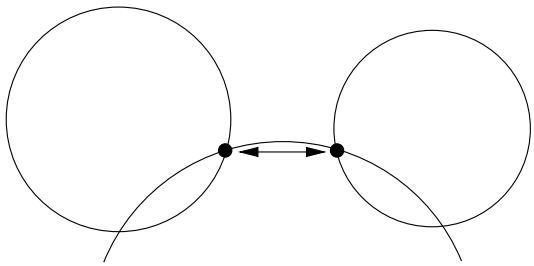


Figure 4.1: An ε separation needed to avoid a potential degeneracy.

4.1 Identifying the Degeneracies

A new circle C_i may induce many degeneracies with circles in $\mathcal{C}'_{i-1} = \{C'_1, \ldots, C'_{i-1}\}$. When adding the i-th circle, we wish to resolve all those potential degeneracies at once. Therefore we may need to perturb C_i by more than ε. We determine an upper bound δ that guarantees that if C_i is randomly perturbed such that its new center is within a circle of radius δ around its original center, then with high probability, all the potential degeneracies involving the i-th circle and the circles in $\mathcal{C}'_{i-1}$ are resolved.

There are four types of degeneracies in an arrangement of circles:

1. An outer tangency between two circles.
2. An inner tangency between two circles.
3. Three circles intersect in the same point.
4. The centers of two intersecting circles are too close.

Notice that we regard two circles with centers too close as a degeneracy (type 4), since it makes the resolution parameter for degeneracy of type 3 too big, thus we regard this case as a degeneracy *only* when the circles intersect. We can check if they are intersecting using the outer and inner tangency tests (further explanation is given in the next chapter). We also require that the size of all the radii will be at least ε (we need this assumption in order to give a good bound on degeneracy of type 1).

4.2 Estimating the Forbidden Regions

The degeneracies described above define a forbidden space for the center of the newly inserted circle, that is, the places where we cannot put the center of a new circle, C_i without incurring a potential degeneracy. Denote the forbidden region induced by the first, second, third and forth types, by F_1, F_2, F_3 and F_4, respectively. Our goal is to compute a worst-case estimate for the area of the forbidden regions. We denote by ρ_{ij} the distance between the centers of C_i and C'_j, for $j < i$. In this subsection, we describe the regions induced by the newly added circle C_i and an existing circle (or a pair of circles in the case of F_3). The forbidden region is the union of all such regions (e.g., F_1 is the union of forbidden regions induced by C_i and a potential outer tangency with each circle in $\mathcal{C}'_{i-1}$).

- **The region** F_1 consists of placements of the center of C_i that induce an outer tangency or near tangency of C_i and another circle. For a circle $C'_j \in \mathcal{C}'_{i-1}$, an exact outer tangency is induced by placing the center of C_i at distance exactly $R_i + R_j$ away from the center of C'_j , namely, $\rho_{ij} - R_i - R_j = 0$. We define the potential degeneracy of this type when using floating-point with resolution parameter $\varepsilon > 0$ as the locus of the center of C_i such that $-\varepsilon \leq \rho_{ij} - R_i - R_j \leq \varepsilon$, which is an annulus centered at the center of C'_j with radii $R_i + R_j + \varepsilon$ and $R_i + R_j - \varepsilon$ (Figure 4.2 (a)). Its area is $\pi[(R_i + R_j + \varepsilon)^2 - (R_i + R_j - \varepsilon)^2] = 4\pi(R_i + R_j)\varepsilon$.
- **The region** F_2 is defined similarly to F_1 for the case of *inner* tangency. Assuming $R_i > R_j$, the area is $\pi[(R_i - R_j + \varepsilon)^2 - (R_i - R_j - \varepsilon)^2] = 4\pi(R_i - R_j)\varepsilon$ (Figure 4.2 (b)).
- **The region** F_3 is defined as follows. Let P_{jk} denote $C'_j \bigcap C'_k$ where $C'_j, C'_k \in \mathcal{C}'_{i-1}$. The locus of placements of the center of C_i that will cause C_i to pass

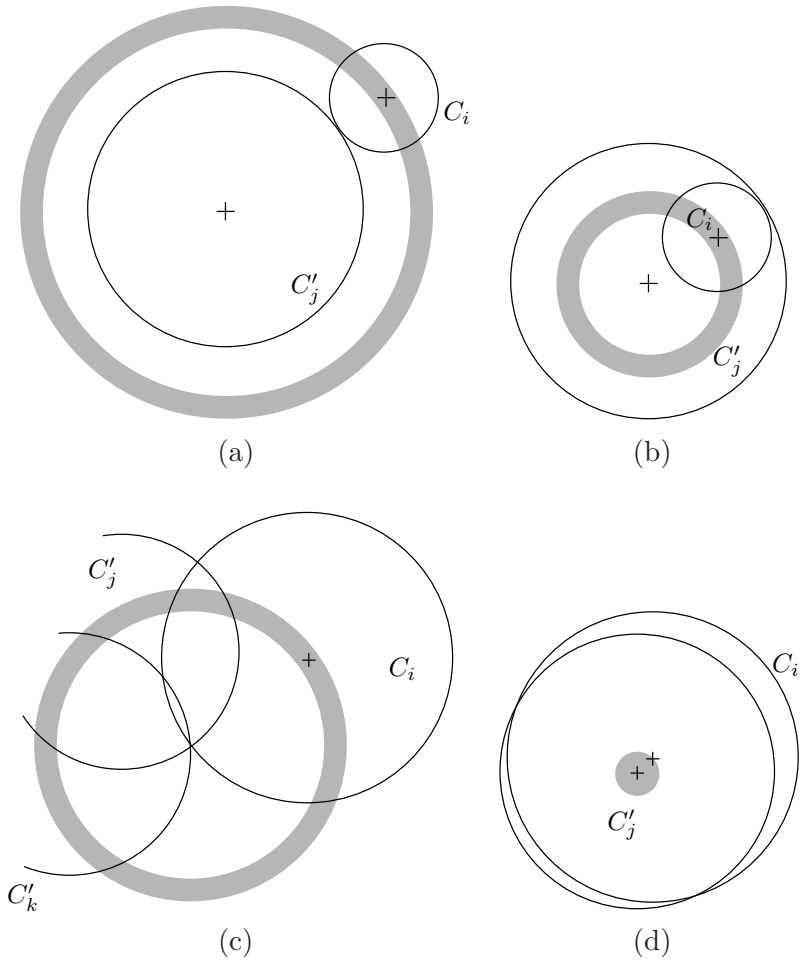


Figure 4.2: The shaded areas in (a),(b),(c),(d) are the portions of the forbidden regions F_1, F_2, F_3 and F_4 respectively for C_i and C'_j (and also C'_k for F_3).

through, or very near to a point in P_{jk} is an annulus (Figure 4.2 (c)). The total forbidden area is $\pi[(R_i+\varepsilon)^2-(R_i-\varepsilon)^2]\cdot card(C'_j \bigcap C'_k) = 4\pi R_i\varepsilon \cdot card(C'_j \bigcap C'_k)$.

- **The region** F_4 consists of placements of the center of C_i such that for an existing circle $C'_j \in \mathcal{C}'_{i-1}$, C_i and C'_j intersect and $\rho_{ij} \leq \varepsilon$ holds (i.e., the centers of the circles are less than ε away — Figure 4.2 (d)). Its area is $\pi\varepsilon^2$.

4.3 Bounding the Area of F_1, F_2, F_3 and F_4

Let $\mathcal{C}$ be a collection of circles as defined above. Also, let $R := max_{i=1}^{n} R_i$, and let k denote the density parameter of $\mathcal{C}$. There are at most k circles defining the regions of F_1 and F_2, there are at most $\binom{k}{2} \times 2$ points defining the region F_3, and at most k points defining the region F_4. Therefore the following bounds on the areas can be obtained.

$$\begin{aligned} A(F_1 \cup F_2) &\leq k[4\pi(R+R)\varepsilon + 4\pi(R-0)\varepsilon] \\ &= k(8\pi R\varepsilon + 4\pi R\varepsilon) \\ &= 12\pi kR\varepsilon. \end{aligned}$$

$$\begin{aligned} A(F_3) &\leq 2\binom{k}{2}4\pi R\varepsilon = \binom{k}{2}8\pi R\varepsilon \\ &\leq \frac{1}{2}k^2 8\pi R\varepsilon = 4\pi k^2 R\varepsilon. \end{aligned}$$

$$A(F_4) \leq k\pi\varepsilon^2.$$

Hence, the bound on the forbidden area $AF = A(F_1 \bigcup F_2 \bigcup F_3 \bigcup F_4)$ is:

$$AF \leq 12\pi kR\varepsilon + 4\pi k^2 R\varepsilon + k\pi\varepsilon^2 = \pi k\varepsilon(12R + 4kR + \varepsilon).$$

If C_i should be perturbed, then δ will define a disk D_δ in which its center can be moved (Figure 3.2). We want the area of this disk to be at least twice the area of the forbidden space. Thus, with probability $\geq \frac{1}{2}$ a point chosen at random inside D_δ constitutes a valid (i.e., a potential degeneracy free) perturbation for C_i.

Therefore we require that

$$\pi\delta^2 > 2AF$$

$$\delta^2 > 2k\varepsilon(12R + 4kR + \varepsilon)$$

$$\delta > \sqrt{2k\varepsilon(12R + 4kR + \varepsilon)}\,. \tag{4.1}$$

It is important to emphasize that at no point of the algorithm, do we compute the intersection of the disk (implied by the center of the circle to be inserted and δ) with the forbidden regions. Instead, we randomly choose a point inside that disk and check that there is no potential degeneracy when setting it as the center of the circle. This is a key point in the practicality of the method — this is what makes the scheme fairly easy to implement.

The perturbation bound δ that was described above is rather crude. Furthermore, in our implementation we do not even use it (Chapter 7). Indeed, the perturbation bound is less important than the resolution bound. The latter is crucial for certifying the validity of the arrangement.

Yet, the perturbation bound is interesting for two main reasons:

- We use δ to establish an upper bound on the running time of the algorithm. It is good to show that there are no huge constants hidden in it.
- If a certain level of accuracy is required by the application at hand, the perturbation bound could be used to predetermine the length of the mantissa needed to achieve that accuracy (further details are given in Chapter 8).

In the next chapter we present the predicates, and derive the resolution bound ε, needed for the computation of δ.

Chapter 5

The Resolution Bound

In this chapter we examine the four possible degeneracies that can arise in an arrangement of circles (Section 4.1). Given the precision of the underlying arithmetic, we can find the ε required to remove them. In other words, we determine for each degeneracy a distance ε such that if one of the circles involved in this degeneracy is moved by at least ε away from the degenerate configuration, then we can safely evaluate the corresponding predicate with the given precision. For each degeneracy we present the appropriate predicate and also compute the worst case ε. Using this ε we then compute the value of δ, the maximum distance of a perturbed circle C_i from its original position, as described in Chapter 4. Denote by ε_i the resolution parameter needed to compute the forbidden region F_i.

5.1 Preliminaries

To examine the error induced by a floating-point computation we will use the notation suggested in [7, 15]. The symbols $\odot$, $\oplus$, $\ominus$, $\oslash$ and $\sqrt{}$ denote the floating-point implementation of multiplication, addition, subtraction, division and square root respectively. We will abbreviate $x \odot x$ by x^2. Denote a predicate which takes m arguments and determines the sign of an expression by $Pr_s = sign(E(x_1, \ldots, x_m))$. Denote by Pr_p the predicate which takes m arguments and returns *true* iff $E(x_1, \ldots, x_m) > 0$. We define a degeneracy when $E = 0$.

Since we are using floating-point arithmetic, we cannot compute E exactly. Instead, we are only computing an approximation $\widetilde{E}$ of E. We also compute a bound $B > 0$ on the maximum difference between $\widetilde{E}$ and the exact value E, namely, $|E - \widetilde{E}| \leq B$ or $\widetilde{E} - B \leq E \leq \widetilde{E} + B$. Consequently, if $\widetilde{E} > B$ then $E > 0$, and if $\widetilde{E} < -B$ then $E < 0$.

The bound B is computed according to the recursive definitions of the index ind_E and the supremum $\widetilde{E_{sup}}$ of an expression E in the following way: $B = 2^{-p} \odot ind_E \odot \widetilde{E_{sup}}$, where p denotes the length of the mantissa. $\widetilde{E_{sup}}$ and ind_E are computed

E	$\widetilde{E}$	$\widetilde{E_{sup}}$	ind_E
A	A	$\lvert A\rvert$	0
$A+B$	$\widetilde{A}\oplus\widetilde{B}$	$\widetilde{A_{sup}}\oplus\widetilde{B_{sup}}$	$1+max(ind_A,ind_B)$
$A-B$	$\widetilde{A}\ominus\widetilde{B}$	$\widetilde{A_{sup}}\oplus\widetilde{B_{sup}}$	$1+max(ind_A,ind_B)$
$A\cdot B$	$\widetilde{A}\odot\widetilde{B}$	$\widetilde{A_{sup}}\odot\widetilde{B_{sup}}$	$1+ind_A+ind_B$
A/B	$\widetilde{A}\oslash\widetilde{B}$	$\frac{(\widetilde{A}\oslash\widetilde{B})\oplus(\widetilde{A_{sup}}\oslash\widetilde{B_{sup}})}{(\lvert\widetilde{B}\rvert\oslash\lvert\widetilde{B_{sup}}\rvert)\ominus(ind_B+1)\cdot 2^{-p}}$	$1+max(ind_A,ind_B+1)$
$A^{\frac{1}{2}}$, $\widetilde{A}>0$	$\sqrt{\widetilde{A}}$	$(\widetilde{A_{sup}}\oslash\widetilde{A})\odot\sqrt{\widetilde{A}}$	$1+ind_A$
$A^{\frac{1}{2}}$, $\widetilde{A}=0$	$\sqrt{\widetilde{A}}$	$\sqrt{\widetilde{A_{sup}}}\odot 2^{\frac{p}{2}}$	$1+ind_A$

Table 5.1: The computation of $\widetilde{E_{sup}}$ and ind_E [15]. In the first row we assume that A is a floating-point number.

recursively according to Table 1 (taken from [15]). Intuitively, $\widetilde{E_{sup}}$ reflects errors resultings from the *operands*, and ind_E reflects errors resultings from the *operators*.

When we add C_i to the collection $\mathcal{C}'_{i-1}$, if *for all* the predicates E involving C_i (regarding all the circles that were already inserted), $|\widetilde{E}|>B$, then C_i is in a valid place, and there is no need to perturb it. If there *exists* a predicate E, for which $|\widetilde{E}|\leq B$, we define such a configuration as a *potential degeneracy*, and we need to perturb C_i. Hence, for each predicate, we need to understand the *geometric* meaning of $|\widetilde{E}|>B$, so it will be reflected in ε and then in δ.

5.2 Outer Tangency

For two circles C_1 and C_2, an outer tangency occurs when the following holds:

$$[(X_1-X_2)^2+(Y_1-Y_2)^2]^{\frac{1}{2}}=R_1+R_2\,.$$

We wish to refrain from using the square-root operation whenever possible (as it leads to coarse bounds on the error). Therefore we take the expression E in the corresponding predicate Pr_s to be:

$$E=(X_1-X_2)^2+(Y_1-Y_2)^2-(R_1+R_2)^2\,. \tag{5.1}$$

We use floating-point arithmetic, so we will compute

$$\widetilde{E}=(X_1\ominus X_2)^2\oplus(Y_1\ominus Y_2)^2\ominus(R_1\oplus R_2)^2\,.$$

According to Table 5.1 we have:

- $\widetilde{E_{sup}}=(|X_1|\oplus|X_2|)^2\oplus(|Y_1|\oplus|Y_2|)^2\oplus(|R_1|\oplus|R_2|)^2$
- $ind_E=5$

- $B = 2^{-p} \odot ind_E \odot \widetilde{E_{sup}}$.

Define a *potential* outer tangency between two circles C_1 and C_2 when

$$|\widetilde{E}| \leq B \, .$$

We call it a potential outer tangency because we do not know for certain, if there is or there is not an outer tangency. Therefore, we require that for all outer tangency tests $|\widetilde{E}| > B$ will hold.

We notice that, it follows from the basic relation $|E - \widetilde{E}| \leq B$, that if $|E| > 2B$ then $|\widetilde{E}| > B$. So, we require that, for all outer tangency tests, $|E| > 2B$. We do so since it is more convenient to analyze the effect of the perturbation using standard arithmetic rather than floating-point arithmetic.

If $|E| = 0$ then the circles are exactly tangent and the distance between their centers is $R_1 + R_2$. Yet, if $|E| > 2B$ (as we wish it to be), then the centers of the circles are $R_1 + R_2 \pm \varepsilon$ distance apart, where $\varepsilon > 0$. The smallest $\varepsilon > 0$ that will cause $|E| > 2B$ to hold, is the resolution bound that we seek. So we have

$$[(X_1 - X_2)^2 + (Y_1 - Y_2)^2]^{\frac{1}{2}} = R_1 + R_2 \pm \varepsilon \, .$$

After squaring both sides, and rearranging terms we get:

$$(X_1 - X_2)^2 + (Y_1 - Y_2)^2 - (R_1 + R_2)^2 = \pm 2(R_1 + R_2)\varepsilon + \varepsilon^2 \, .$$

We notice that the left-hand side is exactly E, so we can rewrite our requirement, this time in terms of ε, that is

$$| \pm 2(R_1 + R_2)\varepsilon + \varepsilon^2| > 2B \, .$$

We first consider the inequality

$$| + 2(R_1 + R_2)\varepsilon + \varepsilon^2| > 2B \, .$$

We notice that the term $(R_1 + R_2)$ can be very small. So for a worst-case estimation of ε we will suppose that $(R_1 + R_2) = 0$. Thus, we rewrite the last inequality as $|\varepsilon^2| > 2B$.

Recall that M is the maximum value for $X_1, X_2, Y_1, Y_2, R_1, R_2$. By setting all the parameters in $\widetilde{E_{sup}}$ to be M, we can now deduce a worst case ε_1 for outer tangency, needed to estimate F_1 (the forbidden region for the placement of the i-th circle, regarding the outer tangency degeneracy)

$$\varepsilon_1 > \sqrt{10 \odot 2^{-p} \odot 12 \odot M^2} \, . \tag{5.2}$$

Following [15] the computation of B should be done in Round To Nearest mode. Since we are interested in a worst-case bound, for the square-root operation in Inequality 5.2, we use UP rounding mode.

Next, we consider the inequality

$$| - 2(R_1 + R_2)\varepsilon + \varepsilon^2| > 2B$$

We assumed that all the radii are at least ε, so $(R_1 + R_2) \geq 2\varepsilon$. Suppose that $(R_1 + R_2) = 2\varepsilon$, then we have,

$$| - 2(R_1 + R_2)\varepsilon + \varepsilon^2| = | - 2(2\varepsilon)\varepsilon + \varepsilon^2| > |\varepsilon^2| \Rightarrow$$
$$| - 2(R_1 + R_2)\varepsilon + \varepsilon^2| > |\varepsilon^2| \quad (5.3)$$

If $(R_1 + R_2) > 2\varepsilon$, then the left-hand side of Inequality 5.3 only increases. Thus, we conclude that Inequality 5.2 also holds for the case when $| - 2(R_1 + R_2)\varepsilon + \varepsilon^2| > 2B$.

Here is the code segment that computes ε_1 (notice that we use the Visual C++ function, `_controlfp()`, for changing the rounding mode; for the gcc compiler, we use the `fesetround()` function).

```
/* NT is the number type (the default is 'double'), machine_eps is
the machine epsilon (for 'double' it is 2^-52) and M is the maximal
input size */
NT temp = 10*machine_eps*12*M*M;
// set UP rounding mode
_controlfp(_RC_UP,_MCW_RC);
// epsilon for F_1 and F_2
NT eps1_2=sqrt(temp);
// restore normal rounding mode
_controlfp( _CW_DEFAULT, 0xfffff );
```

The next code segment illustrates how we implemented the predicate itself.

```
/* test for outer tangency.  temp_x, temp_y and temp_r refer to
an existing circle.  new_x, new_y and new_r refer to the newly
added circle.  */
NT E1 = fabs((temp_x-new_x)*(temp_x-new_x)+
(temp_y-new_y)*(temp_y-new_y)-(temp_r+new_r)*(temp_r+new_r));
if(E1 <= 5*machine_eps*12*M*M)
{
// a potential degeneracy exists
...
}
```

We conclude this section with a lemma that summarizes the discussion above:

Lemma 1 *Given two circles, such that the value of each center coordinate or radius is at most M, and p is the length of the floating-point mantissa — if the absolute difference between the sum of the radii of the two circles and the distance between their centers is greater than $\sqrt{10 \odot 2^{-p} \odot 12 \odot M^2}$, then we can safely determine that no outer tangency exists between the two circles.*

5.3 Inner Tangency

An inner tangency between two circles C_1, C_2 occurs when the following holds (without loss of generality, assume $R_2 > R_1$):

$$[(X_1 - X_2)^2 + (Y_1 - Y_2)^2]^{\frac{1}{2}} = R_2 - R_1 \,.$$

By the same arguments raised earlier for outer tangency, we take the expression E in the predicate Pr_s to be:

$$E = (X_1 - X_2)^2 + (Y_1 - Y_2)^2 - (R_2 - R_1)^2 \,. \tag{5.4}$$

Following similar arguments, to those in the case of outer tangency, we conclude that Inequality 5.2 applies also in the case of inner tangency, that is $\varepsilon_2 = \varepsilon_1$ (the error B is the same for both cases).

Notice that there is a subtle difference between this case and the case of outer tangency. Recall that in the previous section we obtained the inequality $|\pm 2(R_1 + R_2)\varepsilon + \varepsilon^2| > 2B$. The analogous inequality in this case is $|\pm 2(R_2 - R_1)\varepsilon + \varepsilon^2| > 2B$, where $R_2 - R_1 > 0$.

In the case of the *plus* sign, similar arguments to those of the previous section hold (recall that $R_2 - R_1 > 0$). For the case of the minus sign

$$|-2(R_2 - R_1)\varepsilon + \varepsilon^2| > 2B \,,$$

we notice that if $R_2 - R_1 > \varepsilon$ then

$$|-2(R_2 - R_1)\varepsilon + \varepsilon^2| \geq |\varepsilon^2|$$

and Inequality 5.2 is indeed valid. However, if $0 < R_2 - R_1 \leq \varepsilon$, then we cannot certify that Inequality 5.2 holds (e.g., if $R_2 - R_1 = \frac{1}{2}\varepsilon$, then no ε is valid). Yet, recall that our goal in finding ε, is to compute F_2. If $0 < R_2 - R_1 \leq \varepsilon$, then for any ε, there is no valid placement of C_1 so that it is completely inside C_2, so we do not care what is the size of ε that the inequality $|-2(R_2 - R_1)\varepsilon + \varepsilon^2| > 2B$ will yield (Figure 5.1). Intuitively, the inequality $|-2(R_2 - R_1)\varepsilon + \varepsilon^2| > 2B$ means moving the center of C_1 further inside C_2 so there will be no potential inner tangency. However, if $R_2 - R_1 \leq \varepsilon$ then there is no sense in doing so, since there are no valid places to begin with.

In conclusion, Inequality 5.2 gives a valid ε also for F_2. Notice, that Inequality 5.2 gives a tight bound on ε_2 (e.g., construct two circles C_1 and C_2, such that $X_1 = Y_1 = X_2 = Y_2 = R_1 = R_2 = M$). That is, the bound can be achieved, since the term multiplied by ε (in $|\pm 2(R_2 - R_1)\varepsilon + \varepsilon^2| > 2B$) is zero.

5.4 Three Circles Intersecting In a Common Point

In this section we will present an alternative approach to floating-point error analysis, that we shall employ in conjunction with the one that was already given. Our first

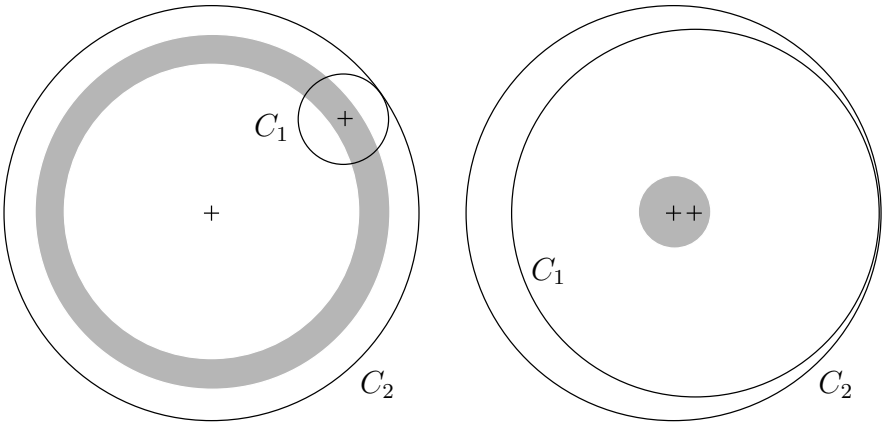


Figure 5.1: The shaded area is a part of F_2, the forbidden region. (a) The case where $R_2 - R_1 > \varepsilon$. Notice that there are valid placements for the center of C_1 such that C_1 is completely inside C_2. (b) The case where $0 < R_2 - R_1 \leq \varepsilon$. Notice that there is *no* valid placement for the center of C_1 such that C_1 is completely inside C_2.

attempt to give a good resolution bound for this type of degeneracy, was to continue with the same approach as in the previous sections (based on [15]). However, since this is a more complicated situation, the bound that was achieved was very large.

We will compute the intersection points, and Err — a bound on the worst case error that can occur during this computation (caused since we are using floating-point arithmetic). Then, around each intersection point we inflate a disk of radius Err. We then make sure that none of the disks overlap.[1]

To compute the intersection point of two circles C_1 and C_2, we use the following formulation [10].

$$s = \frac{1}{2}\frac{{R_1}^2 - {R_2}^2}{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} + \frac{1}{2} \tag{5.5}$$

$$t = [\frac{{R_1}^2}{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} - s^2]^{\frac{1}{2}}\,. \tag{5.6}$$

The intersection point $[x, y]$ is:

$$\begin{aligned}[x, y] &= [X_1, Y_1] + s[X_2 - X_1, Y_2 - Y_1] \\ &\quad \pm t[Y_2 - Y_1, X_1 - X_2]\,.\end{aligned} \tag{5.7}$$

First, we show how to bound the error of an expression that involves only $+, \cdot$ and square-root operations with positive input operands. Then, we will give a bound for

[1] Notice that throughout this section, we are only concerned with pairs of intersection points originating from three different circles. We explain how to handle pairs of intersection points originating from two circles in Section 6.1.

the worst-case error for such expressions. Finally, we will convert Eq. 5.7, such that it will not contain subtraction and division operations, so a bound on the worst case error could be established.

We rewrite the expression as a straight-line program $E_i, i = 1 \ldots m$ such that, each subexpression E_i involves just one arithmetic operation, and takes as its operands the results from previous subexpressions or input parameters (i.e., if $E = ab + cd$, then $E_1 = ab$, $E_2 = cd$ and $E_3 = E_1 + E_2$). The rewriting should be carried out such that it preserves the standard priority of arithmetic operations. By a slight abuse of notation we also denote by E_i the *exact value* of the subexpression E_i.

To evaluate the bound on the error of an expression E, we compute an *interval*, which contains the *exact* value of E, and its length will be the bound on the error. The computation of E is done by the following rules of interval arithmetic [4]. Let $[x]$ denote the interval $[\underline{x}, \overline{x}]$, and $[y]$ the interval $[\underline{y}, \overline{y}]$, the rules for the $+, \cdot$ and square-root operations (with positive operands) are:

$$[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$$

$$[x] \cdot [y] = [\underline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}]$$

$$[x]^{\frac{1}{2}} = \ [\underline{x}^{\frac{1}{2}}, \overline{x}^{\frac{1}{2}}] \quad \underline{x} \geq 0$$

We evaluate E as follows: When we evaluate the first subexpression E_1, all we can compute is $\widetilde{E_1}$ — the floating-point approximation of E_1 (recall that we do *all* our computations using floating-point arithmetic). We will create the interval $[\underline{E_1}, \overline{E_1}]$ where $\underline{E_1}$ is the *next representable floating-point number* after $\widetilde{E_1}$ in the direction of $-\infty$, and $\overline{E_1}$ is the next representable floating-point number after $\widetilde{E_1}$ in the direction of $+\infty$. Getting the next representable floating-point number can be done using the function `nextafter()`, which is recommended by the IEEE standard, and is available for most compilers. Thus, the interval $[\underline{E_1}, \overline{E_1}]$ contains $\widetilde{E_1}$ (Figure 5.2 (a)). Next, we need to compute E_2. If E_2 takes only input parameters as its operands, then $[\underline{E_2}, \overline{E_2}]$ is computed similarly to $[\underline{E_1}, \overline{E_1}]$. If E_2 takes E_1 as at least one of its operands, then we will compute $[\widetilde{\underline{E_2}}, \widetilde{\overline{E_2}}]$ according to the rules of interval arithmetic (for an input parameter a, we take $[\underline{a}, \overline{a}] = [a, a]$), where $\widetilde{\underline{E_2}}$ and $\widetilde{\overline{E_2}}$ are the floating-point approximation of the interval end-points. Since $\widetilde{\underline{E_2}}$ and $\widetilde{\overline{E_2}}$ were computed using floating-point arithmetic and rounding errors may occur, we will create the interval $[\underline{E_2}, \overline{E_2}]$, where $\underline{E_2}$ is the next representable floating-point number after $\widetilde{\underline{E_2}}$ in the direction of $-\infty$, and $\overline{E_2}$ is the next representable floating-point number after $\widetilde{\overline{E_2}}$ in the direction of $+\infty$ (Figure 5.2 (b)). We continue to compute all the subexpressions $E_3 \ldots E_m$ in a similar manner (depending on the origin of the operands of each subexpression). The following two lemmas justify the method and explain how a worst-case error bound is derived.

Lemma 2 *Evaluating an expression E that involves only $+, \cdot$ and square-root operations with positive input operands, in the method described above, yields a bound on the error of the expression, when evaluated using standard floating-point arithmetic. The bound is the length of the last interval, $[\underline{E_m}, \overline{E_m}]$.*

Proof. Our first assumption is that no underflow or overflow occurs, and the rounding mode is `to nearest`. We prove by induction that $E_m \in [\underline{E_m}, \overline{E_m}]$ and $\widetilde{E_m} \in [\underline{E_m}, \overline{E_m}]$. When we evaluate $[\underline{E_i}, \overline{E_i}]$, the induction assumption is that each interval $[\underline{E_j}, \overline{E_j}], j < i$ contains the exact value of E_j.

For each floating-point operation, the error is bounded according to the following inequality [2],

$$|u * v - fl(u * v)| \leq mach\_eps|u * v| \tag{5.8}$$

where $mach\_eps$ is the machine epsilon (that is $mach\_eps = 2^{-p}$), $*$ is one of the operations $+, -, \cdot, /, \sqrt{}$, and $fl()$ is the floating-point result.

It follows from Inequality 5.8, that taking the next representable floating-point numbers after $\widetilde{E_1}$ (the floating-point approximation of E_1) in both directions as endpoints of the interval $[\underline{E_1}, \overline{E_1}]$ assures us that $E_1 \in [\underline{E_1}, \overline{E_1}]$ (Figure 5.2 (a)). It is also obvious that $\widetilde{E_1} \in [\underline{E_1}, \overline{E_1}]$. For all subexpressions that rely only on the input parameters we will use similar arguments to the case of E_1.

Assume that E_i is the first subexpression that relies on previous intervals $[\underline{E_j}, \overline{E_j}]$ and $[\underline{E_k}, \overline{E_k}]$ (the case where E_i relies on an input parameter as one of the operands is easier). According to the induction assumption, $E_j \in [\underline{E_j}, \overline{E_j}]$ and $E_k \in [\underline{E_k}, \overline{E_k}]$. If the i-th interval were computed *exactly*, since it is computed according to interval arithmetic, it would contain E_i. Since we are *not* computing it exactly, we can only compute $[\widetilde{\underline{E_i}}, \widetilde{\overline{E_i}}]$. Again, following Inequality 5.8, taking the next representable floating-point number after $\widetilde{\underline{E_i}}$ in the $-\infty$ direction, as $\underline{E_i}$, and the next representable number after $\widetilde{\overline{E_i}}$ in the $+\infty$ direction, as $\overline{E_i}$ assures us that E_i is indeed in the interval $[\underline{E_i}, \overline{E_i}]$ (Figure 5.2 (b)). Thus, we can conclude that $E \in [\underline{E_m}, \overline{E_m}]$. Again, it is obvious that also $\widetilde{E} \in [\underline{E_m}, \overline{E_m}]$.

Thus, both E and $\widetilde{E}$ are in the final interval $[\underline{E_m}, \overline{E_m}]$, so the distance (hence the error) between E and $\widetilde{E}$ is at most the length of the interval. □

Lemma 3 *Evaluating an expression E, that contains only $+, \cdot$ and square-root operations with positive input operands, in the method described above,* with the maximum values allowed for all its operands*, yields a bound on the* worst-case *error of the expression, when evaluated using standard floating-point arithmetic.*

Proof. Following Inequality 5.8, for the $+, \cdot$ and $\sqrt{}$ operations with positive operands, the size of the bound on the error is in direct relation to the size of the operands. So

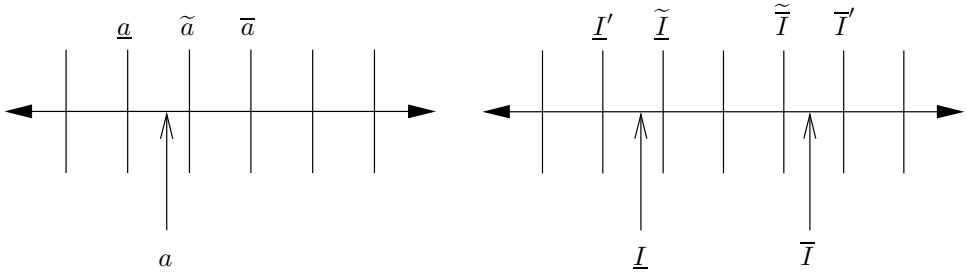


Figure 5.2: (a) Let a be a real number or the result of an expression involving a single operation on one or two floating-point operands, and let $\widetilde{a}$ be the nearest floating-point number to a. Taking the next representable numbers after $\widetilde{a}$ in both directions as end-points of the interval $[\underline{a}, \overline{a}]$ assures us that $a \in [\underline{a}, \overline{a}]$. (b) Let I be the interval $[\underline{I}, \overline{I}]$, and let $\widetilde{\underline{I}}$ and $\widetilde{\overline{I}}$ be the nearest floating-point number to $\underline{I}$ and $\overline{I}$, respectively. Denote the next representable number in the $-\infty$ direction after $\widetilde{\underline{I}}$ as $\underline{I'}$, and the next representable number in the $+\infty$ direction after $\widetilde{\overline{I}}$ as $\overline{I}'$. It follows that $[\underline{I}, \overline{I}] \subseteq [\underline{I'}, \overline{I}']$, thus any number $x \in [\underline{I}, \overline{I}]$ is also contained in $[\underline{I'}, \overline{I}']$.

in order to get the maximum possible error, we need to evaluate E with the maximum values allowed for all its operands.

Each time that we create an interval $[\widetilde{\underline{E_i}}, \widetilde{\overline{E_i}}]$ we shift the end-points to $[\underline{E_i}, \overline{E_i}]$. The fact that the rounding mode is `to nearest`, and the shifting is done by taking the next representable floating-point number in the relevant direction, assures us that the interval $[\underline{E_i}, \widetilde{\underline{E_i}}]$ (resp. $[\widetilde{\overline{E_i}}, \overline{E_i}]$) is larger than the *largest possible error* of $\widetilde{\underline{E_i}}$ (resp. $\widetilde{\overline{E_i}}$).

Simply using `to zero` rounding mode for $\underline{E_i}$, and `to infinity` rounding mode for $\overline{E_i}$ would not suffice. Suppose that for *specific* maximum values (of the operands), no rounding errors have occurred during the computation of E. In this case, using `to zero` and `to infinity` rounding modes, the final interval $[\underline{E_m}, \overline{E_m}]$ would be $[E_m, E_m]$, that is, the interval contains only E_m and we would *falsely* deduce that the bound on the worst-case error is zero. □

To get a bound on the worst-case error of Eq. 5.7, we will change all the *subtraction* operations to *addition* operations, in order to upper-bound the error of the subtraction and all the subsequent operations (as in the computation of the supremum of an expression in Table 5.1). Also, we will only use the absolute value of the operands (so Lemma 3 would hold).

Yet, in Eq. 5.7 there are also division operations. The term in the denominators of Eq. 5.7 is $(X_2 - X_1)^2 + (Y_2 - Y_1)^2$, which is the distance between the centers of the circles. Hence, we will assume that the centers of any two circles are at least some distance ξ apart. If the centers are less then ξ apart, degeneracy of type 4

occurs. We *do not* require from the user to make sure that the centers are ξ apart. This will be taken care of as part of handling degeneracy of type 4 (Section 5.5). In Section 5.6, we explain how we choose ξ, that gives fairly good results when using the IEEE double type. Notice, that choosing a good ξ is a subtle matter, since there is a trade off between the resolution bound induced by degeneracy of type 3 and the resolution bound induced by degeneracy of type 4.

Since we assume that the distance between the centers is at least ξ, then

$$\frac{1}{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} \leq \frac{1}{\xi^2} .$$

As we are looking for a worst-case bound of the error of Eq. 5.7, we can replace $\frac{1}{(X_2-X_1)^2+(Y_2-Y_1)^2}$ with $\frac{1}{\xi^2}$. Let $\chi = \frac{1}{\xi^2}$. We replace Eq. 5.5 and Eq. 5.6 by:

$$\widehat{s} = 0.5(R_1^2 - R_2^2)\chi + 0.5$$

$$\widehat{t} = (R_1^2\chi - \widehat{s}^2)^{\frac{1}{2}} .$$

We can now bound the error of the $[x, y]$ values obtained in Eq. 5.7 according to the method described above (i.e., regard Eq. 5.7 as E, and compute the interval which gives a bound on the worst-case error). Before we evaluate it, we will determine the value of ξ, and then compute $\chi = \frac{1}{\xi^2}$ using UP rounding mode.

Let Err denote the bound on the worst-case error for Eq. 5.7, computed using the method described above and multiplied by $\sqrt{2}$. Err is a positive floating-point number.

We can imagine that around each *approximate* intersection point P that we compute, we inflate a disk of radius Err (Figure 5.3) which contains the *exact* intersection point (recall that the bound that was computed for Eq. 5.7 applies to only one coordinate, either x or y, hence we need to multiply it by $\sqrt{2}$). To prevent three circles from intersecting in a common point, we require that no two such disks will overlap.[2] In other words, two approximate intersection points P_1 and P_2 should be at least $2Err$ apart. Still, in order to be able to apply the efficient perturbation algorithm (Section 6.1), we would like to separate the *exact* intersection points even more, thus we require that two approximate intersection points P_1 and P_2 should be at least $6Err$ apart.

Since we are using floating points arithmetic, we will apply the same method that we used for degeneracies of type 1 and 2, to verify that none of the disks overlap.

Denote by X_P and Y_P the x and y coordinates of the point P. The expression E, for a predicate Pr_p that will check that three circles do not intersect in a common point will be

$$E = (X_{P_2} - X_{P_1})^2 + (Y_{P_2} - Y_{P_1})^2 - (6Err)^2 , \tag{5.9}$$

[2] Again, we are only concerned with pairs of intersection points originating from three different circles.

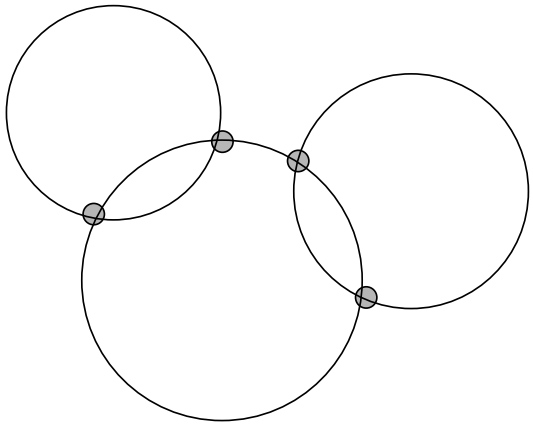


Figure 5.3: Around each approximate intersection point we inflate a disk of radius Err that contains the exact intersection point.

where P_1 and P_2 are intersection points, computed by Eq. 5.7. As before, since we are using floating-point arithmetic we compute,

$$\widetilde{E} = (X_{P_2} \ominus X_{P_1})^2 \oplus (Y_{P_2} \ominus Y_{P_1})^2 \ominus (6Err)^2 \,.$$

and according to Table 5.1, we have:

- $\widetilde{E_{sup}} = (|X_{P_2}| \oplus |X_{P_1}|)^2 \oplus (|Y_{P_2}| \oplus |Y_{P_1}|)^2 \oplus (6Err)^2$
- $ind_E = 5$
- $B = 2^{-p} \odot ind_E \odot \widetilde{E_{sup}}$.

To avoid a potential degeneracy, we require that $\widetilde{E} > B$. Again, it follows that if $|E| > 2B$ then $|\widetilde{E}| > B$. So we now require that $E > 2B$.

If $E = 0$ then the distance between the points is exactly $6Err$. Yet, if $E > 2B$ (as we wish it to be), then the distance between the points is $6Err + \alpha$, where $\alpha > 0$. We seek the smallest $\alpha > 0$ that will cause $E > 2B$ to hold. So we have

$$[(X_{P_2} - X_{P_1})^2 + (Y_{P_2} - Y_{P_1})^2]^{\frac{1}{2}} = 6Err + \alpha \,.$$

After squaring both sides, and rearranging terms we get,

$$(X_{P_2} - X_{P_1})^2 + (Y_{P_2} - Y_{P_1})^2 - (6Err)^2 = 12Err\alpha + \alpha^2 \,. \tag{5.10}$$

We notice that the left-hand side of Eq. 5.10 is exactly E, so we can rewrite our requirement, this time in terms of the right-hand side of Eq. 5.10 (the added distance α), that is

$$12Err\alpha + \alpha^2 > 2B \,.$$

We can extract a bound on α,

$$\alpha > \sqrt{2B} = \sqrt{(10 \odot 2^{-p}((|X_{P_2}| \oplus |X_{P_1}|)^2 \oplus (|Y_{P_2}| \oplus |Y_{P_1}|)^2 \oplus (6Err)^2))} \,.$$

We must now bound the maximum value of an intersection-point coordinate. Construct two circles, such that both have radius M, their center's x coordinate is M, and one circle is slightly above the other; then, the right intersection point has x coordinate $\approx 2M$. Therefore, the bound for the maximum value of an intersection-point coordinate is $2M$, and we can give a worst case bound for α,

$$\alpha > \sqrt{(10 \odot 2^{-p}(32M^2 \oplus 36Err^2))} \, .$$

So we can now deduce the worst case ε_3, needed to estimate F_3 (recall that α is just an added distance to $6Err$, to make sure that the predicate will not fail),

$$\varepsilon_3 > 6 \odot Err \oplus \alpha = 6 \odot Err \oplus \sqrt{(10 \odot 2^{-p}(32M^2 \oplus 36Err^2))} \, . \tag{5.11}$$

Remark. We use UP rounding mode for all the operations except in the computation of B (recall that according to [15], we compute B in Round To Nearest mode).

Since we wish to allow an efficient point location mechanism (Section 6.3), we shall add a test, the *circle - intersection point* test. Around each existing exact intersection point, we shall inflate a disk of radius Err and we shall verify that the the newly added circle C_i does not intersect any of these disks. Because we do not know the exact intersection points, we shall inflate a disk of radius $2Err$ around the approximate intersection points, and verify that C_i doest not intersect any of these bigger disks. Since Err is a bound on the distance between the exact and the approximate point, the disks with radii $2Err$ contain the original disks with radii Err (Figure 5.4).

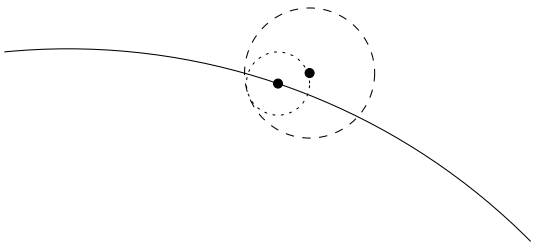


Figure 5.4: Since Err is a bound on the distance between the exact and the approximate point, the disks with radii $2Err$ (the dashed circle) contain the original disks with radii Err (the dotted circle).

Denote by ε_3' the resolution bound needed to compute F_3 with regard to this test. For the newly added circle C_i and an existing approximate intersection point P, we set $C_1 := C_i$ and $X_2 := X_P, Y_2 := Y_P, R_2 := 2Err$. The newly added circle C_i is valid when the following holds:

$$(X_1 - X_2)^2 + (Y_1 - Y_2)^2 - (R_1 + R_2)^2 > 0 \bigvee (X_1 - X_2)^2 + (Y_1 - Y_2)^2 - (R_1 - R_2)^2 < 0 \, . \tag{5.12}$$

If we were to use exact computation, then ε_3' would simply be $2Err$ (recall how we estimate F_3 — see Section 4.2). Yet, since we are using floating-point arithmetic, we need to take into account the possible errors in the computation of Expression 5.12. Expression 5.12 can be regarded as a combination of outer and inner tangency tests (except that we omitted the absolute value on the left-hand side), so the resolution bound ε_3' should be,

$$\varepsilon_3' = 2Err + \varepsilon_1 \,.$$

(ε_1 is the added distance that allows us to evaluate Expression 5.12 using floating-point arithmetic). Because $\varepsilon_3' < \varepsilon_3$, we can still use ε_3 in the computation of F_3.

Remark. Although ε_3 is worse than ε_3', we use it to compute F_3, since the predicate from which it arises, enables us to determine the order of two intersection points along the x or y axis. This ordering allows us to carry out the efficient algorithm described below in Section 6.1. Otherwise, we could have used ε_3'.

5.5 The Centers of Two Intersecting Circles Are Too Close

In handling degeneracy of type 3, we assumed that the distance between the centers of each pair of *intersecting* circles, is at least ξ, where ξ is a positive floating-point number. We can check if two circles are intersecting by using the outer and inner tangency tests.[3] For two circles, C_1 and C_2, the test is

$$(X_1-X_2)^2+(Y_1-Y_2)^2-(R_1+R_2)^2<0 \bigwedge (X_1-X_2)^2+(Y_1-Y_2)^2-(R_1-R_2)^2>0 \,.$$

. As was mentioned in the previous subsection, the task of finding a ξ that will always give good results is not simple. See the numerical example in the next section, for a specific choice of ξ.

We now need a predicate that will assert that the degeneracy does not occur, and also a value for ε_4 (needed for the computation of δ). For two circles, C_1 and C_2, the expression E for the predicate is,

$$E = (X_1 - X_2)^2 + (Y_1 - Y_2)^2 - \xi^2 \,. \tag{5.13}$$

The predicate Pr_p checks if $E > 0$. Again, since we are using floating-point arithmetic we need to compute,

$$\widetilde{E} = (X_1 \ominus X_2)^2 \oplus (Y_1 \ominus Y_2)^2 \ominus \xi^2 \,.$$

and according to Table 5.1, we have:

- $\widetilde{E_{sup}} = (|X_1| \oplus |X_2|)^2 \oplus (|Y_1| \oplus |Y_2|)^2 \oplus \xi^2$

[3] Due to the perturbation scheme, those predicates will yield *correct* results.

- $ind_E = 5$
- $B = 2^{-p} \odot ind_E \odot \widetilde{E_{sup}}$.

To avoid a potential degeneracy, we require that $\widetilde{E} > B$. Again, if $|E| > 2B$ then $|\widetilde{E}| > B$. So we now require that $E > 2B$.

If $E = 0$ then the distance between the centers is exactly ξ. Yet, if $E > 2B$ (as we wish it to be), then the distance between the points is $\xi + \alpha$, where $\alpha > 0$. We seek the smallest $\alpha > 0$ that will cause $E > 2B$ to hold. So we have

$$((X_1 - X_2)^2 + (Y_1 - Y_2)^2)^{\frac{1}{2}} = \xi + \alpha \,.$$

After squaring both sides, and rearranging terms we get,

$$(X_1 - X_2)^2 + (Y_1 - Y_2)^2 - \xi^2 = 2\xi\alpha + \alpha^2 \,. \tag{5.14}$$

We notice that the left-hand side of Equation 5.14 is exactly E, so we can rewrite our requirement, this time in terms of the added distance, α, that is

$$2\xi\alpha + \alpha^2 > 2B \,.$$

We can extract a bound on α,

$$\alpha > \sqrt{(14 \odot 2^{-p} \odot (8 \cdot M^2 \oplus \xi^2))} \,.$$

We can now deduce a worst case ε_4 needed to estimate F_4,

$$\varepsilon_4 > \xi \oplus \alpha = \xi \oplus \sqrt{(14 \odot 2^{-p} \odot (8 \cdot M^2 \oplus \xi^2))} \,. \tag{5.15}$$

Remark. We use UP rounding mode for the square-root operation in α, and the addition of ξ.

5.6 Numerical Example

In the previous sections, we have presented the predicates that we use in our algorithm, and the worst case ε associated with each forbidden region. Still, when we compute δ, we do not distinguish between the different ε's, thus, we will take the maximal one,

$$\varepsilon = max\{\varepsilon_i | i = 1, \ldots, 4\} \,. \tag{5.16}$$

Here is an example of the various ε's we obtain, when we are using the IEEE double type, with $M = 10^3$ and $\xi = 0.03$:

- $Err \leq 0.009$

- $\varepsilon_{1,2} = 0.00016323404237781946$
- $\varepsilon_3 = 0.05426656007499713885$
- $\varepsilon_4 = 0.03162279436525219228$
- $\Rightarrow \varepsilon = 0.05426656007499713885$.

Remarks. **(1)** There is a strong connection between degeneracies of type 3 and 4. In fact, we added degeneracy type 4, to be able to give a good resolution bound for type 3. Yet, we must ensure that degeneracy type 4 by itself will not make ε very big. So, for different values of M, different minimum distance between the centers is required.
(2) It should be clear that all we require from the user of the perturbation is to insert circles such that their coordinates are less than $M - \Delta$ and their radii are less than M. The user should not worry about whether the centers of the circles are less than ξ apart. If this is the case, it will be taken care of when we remove degeneracies of type 4.

Chapter 6

Algorithmic Details

In Section 6.1 we present an efficient perturbation algorithm, which runs in expected $O(n^2 \log n)$ time. A naive implementation of the perturbation scheme presented in previous chapters would run in $O(n^3)$ expected time. In Section 6.2, we describe the *doubly-connected edge list* structure (DCEL), which allows us to maintain the *topological* information of the subdivision and enhance it with the *geometric* information (its planar embedding). Finally, in Section 6.3, we present a *point location* strategy suitable for our DCEL structure.

6.1 Efficient Perturbation Algorithm

In order to achieve a good running time, we use two types of data structures: a kd-tree [8] and binary trees. The kd-tree is used for practical (heuristic) speeding up of the algorithm, whereas the binary trees are also used to achieve the good theoretical bound on the running time.

When adding the circle C'_i, we use a kd-tree to maintain the circles $\mathcal{C}'_{i-1}$ that were already inserted. That is, the kd-tree is constructed by the x and y coordinates of the centers of the circles in $\mathcal{C}'_{i-1}$. When we add the circle C'_i to $\mathcal{C}'_{i-1}$, we check for degeneracies of C'_i regarding all the circles in the kd-tree whose centers are in the range $X_i - 3R_{max} \leq X \leq X_i + 3R_{max}$ and $Y_i - 3R_{max} \leq Y \leq Y_i + 3R_{max}$ where $R_{max} = max(R_j, j = 1 \ldots i)$ (circles whose centers are outside the range cannot be in a degenerate state with respect to C'_i).

If done in a naive fashion, testing a circle C_i for degeneracy of type 3 can take $O(n^2)$ time (there are $O(n^2)$ intersection points), resulting in an algorithm running in expected $O(n^3)$ time.

In order to make the algorithm efficient, we keep four balanced binary trees for each circle in $\mathcal{C}'_{i-1}$ (Figure 6.1). Denote by $P^j_k, k = 1, \ldots, s$ all the intersection points of C'_j with other circles in $\mathcal{C}'_{i-1}$. We construct the *upper* binary tree T_{upper} of C'_j, such

that it will hold all the points $\{P_k^j, k = 1, \ldots, s | X_j - \frac{R_j}{\sqrt{2}} \leq X_{P_k^j} \leq X_j + \frac{R_j}{\sqrt{2}}, Y_{P_k^j} > Y_j\}$, and use their x coordinate as the key for the binary tree. Similarly, we construct the *lower* binary tree T_{lower} of C'_j, such that it will hold all the points $\{P_k^j, k = 1, \ldots, s | X_j - \frac{R_j}{\sqrt{2}} \leq X_{P_k^j} \leq X_j + \frac{R_j}{\sqrt{2}}, Y_{P_k^j} < Y_j\}$, and use their x coordinate as the key for the binary tree. We also construct the *left* binary tree T_{left} of C'_j, such that it will hold all the points $\{P_k^j, k = 1, \ldots, s | Y_j - \frac{R_j}{\sqrt{2}} < Y_{P_k^j} < Y_j + \frac{R_j}{\sqrt{2}}, X_{P_k^j} < X_j\}$, and use their y coordinate as the key for the binary tree. And similarly, we construct the *right* binary tree T_{right} of C'_j, such that it will hold all the points $\{P_k^j, k = 1, \ldots, s | Y_j - \frac{R_j}{\sqrt{2}} < Y_{P_k^j} < Y_j + \frac{R_j}{\sqrt{2}}, X_{P_k^j} > X_j\}$, and use their y coordinate as the key for the binary tree.

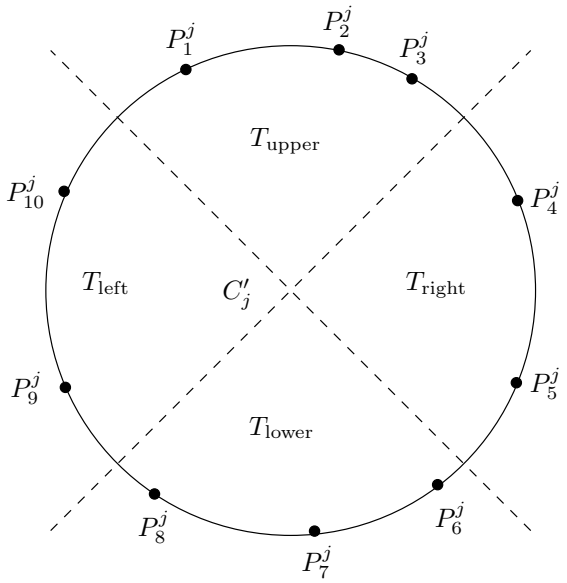


Figure 6.1: The four binary trees associated with a circle C'_j.

When we come to add the new circle C'_i, we check which existing circles it intersects. For an intersection point P, which lies on C'_i and C'_j, we wish to insert it to the appropriate trees of C'_i and C'_j. We first test on which tree T of the four trees associated with C'_j it should be. This test is done using floating-point arithmetic, so errors could occur, yet, we will show later that this is not a problem. Next, we check which are the two neighboring intersection points of P along the tree, if P would be inserted into T. We check if a degeneracy of type 3 occurs with those neighbors. If P would be a leftmost/rightmost leaf in T, we will check it against the rightmost/leftmost leaf in the neighboring tree to T adjacent to P. For example, in Figure 6.1, P_2^j will be checked against P_1^j and P_3^j, and P_3^j will be checked against P_2^j and P_4^j, and so on.

The key observation is that, if the point P is sufficiently far away from of its two neighbors (degeneracy of type 3 does not occur), then it will be sufficiently far away from all other intersection points that belong to the tree containing P. So adding P takes time $O(\log n)$ (the addition of P to the appropriate tree of C'_i is done similarly), and the algorithm would run in overall expected $O(n^2 \log n)$ time.

We still need to prove that the structure of all the binary trees is valid. There is

no ambiguity in the order of two intersection points originating from the same pair of circles. In Eq. 5.7, we first evaluate $[X_1, Y_1]+s[X_2-X_1, Y_2-Y_1]$ and $t[Y_2-Y_1, X_1-X_2]$, and only then, we perform the addition or subtraction of $t[Y_2 - Y_1, X_1 - X_2]$. Thus, the ordering of the intersection points along the x axis or the y axis, depends on a single arithmetic operation. In this case, floating-point arithmetic is reliable (that is, the points cannot switch their order).

We now need to prove the validity of the binary structure for the case where we need to compare two intersection points originating from three different circles. We shall first prove the following lemma:

Lemma 4 *Around each exact intersection point we inflate a disk of radius Err (as was defined in Section 5.4). Denote by $D(P_i)$ such a disk for intersection point P_i. Between each pair of disks $D(P_i)$ and $D(P_j)$ we can put another disk of radius Err, with its center on the common circle of P_i and P_j, and no pair of those three disks will overlap.*

Proof. The distance between the approximate intersection points P_i and P_j is at least $6Err$ (otherwise degeneracy of type 3 occurs). Denote by *seg* the line segment between those approximate intersection points. Without loss of generality, assume that P_i is to the left of P_j. The disk $D(P_i)$ can cover at most the $\frac{1}{3}$ left-hand part of *seg*, and the disk $D(P_j)$ can cover at most the $\frac{1}{3}$ right-hand part of *seg*. Thus, in the middle of *seg* we can put another disk of radius Err, that will not overlap the parts of *seg* covered by $D(P_i)$ or $D(P_j)$ (Figure 6.2 (a)). Moving the center of the third disk, in a direction perpendicular to *seg*, we can place it on the circle on which P_i and P_j lie. Such movement does not cause the third disk to overlap with either $D(P_i)$ or $D(P_j)$ (Figure 6.2 (b)). If a pair of disks would overlap, then there exists no direction in which the projection of the disks will not overlap. Yet, in our case, such a direction does exist (the direction perpendicular to *seg*).

□

We use Lemma 4 to prove the following lemma:

Lemma 5 *Using the predicate for degeneracy of type 3, as was described in Section 5.4, the structure of each binary tree of circle C'_j is valid.*

Proof. Without loss of generality we prove it for the *upper* tree. We show that for each P_i and P_k, which should be on the *upper* tree of C'_j, their order in the tree is correct. To do so, we must prove that the x-ranges of $D(P_i)$ and $D(P_k)$ do not overlap. Denote by Γ the portion of C'_j associated with T_{upper}. For intersection points, that lie near the center of Γ, clearly there can be no overlapping (Figure 6.3 (a)). Thus, we shall check the intersection points that lie near the boundary of Γ (Figure 6.3 (b)). Near the boundary of Γ, we can approximate the circle with a line l tangent to C'_j (Figure 6.3 (c)). The line l has an angle of 135 degrees with the positive direction

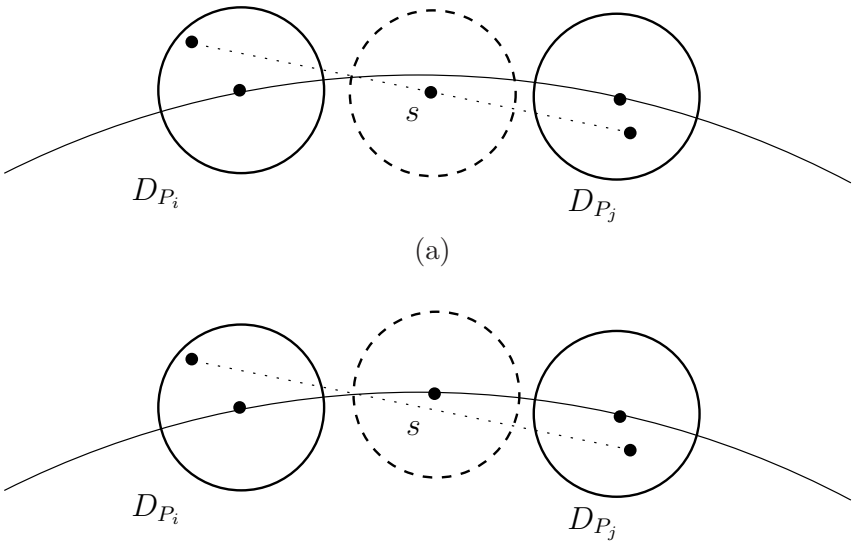


Figure 6.2: (a). Between $D(P_i)$ and $D(P_j)$ we can put another disk of radius Err, and the three disks will not overlap. (b) Moving the center of the third disk, in a direction perpendicular to seg, we can place it on the circle on which P_i and P_j lie.

of the x axis. The worst case configuration, is when the three disks lie on the line l (Figure 6.3 (d)). The distance between the centers of $D(P_i)$ and $D(P_k)$ on the x-axis is $\frac{4Err}{\sqrt{2}}$, which is larger than $2Err$, hence, the x-ranges of $D(P_i)$ and $D(P_k)$ do not overlap. □

Finally, recall that during the creation of the binary trees, floating-point errors can cause us to put intersection points in wrong trees. Yet, the worst case error for the term $X_j \pm R_j/\sqrt{2}$ (or $Y_j \pm R_j/\sqrt{2}$) is significantly smaller than Err, so an intersection point can be mistakenly inserted only as a leftmost/rightmost leaf of a neighboring tree. So all that was said above is still valid.

Recall that in Section 5.4 we also defined the *Circle - Intersection point* test, which verifies that the center of the newly added circle C_i is at least $R_i + Err$ away from all existing approximate intersection points. This test is required for allowing the point location mechanism. If the center of C_i is less than $R_i + Err$ away from an intersection point that lies on C_j, and C_i and C_j do not intersect, then a near tangency will occur (recall the Err is smaller than ε). Thus, C_i and C_j must intersect. Again, we will use the four binary trees associated with each circle to perform this test efficiently.

We next prove the following lemma:

Lemma 6 *Let C_i' be the newly added circle and let C_j' be one of the circles that C_i' intersects. Denote by P_{ij}^1 and P_{ij}^2 the two approximate intersection points of C_i' and C_j'. If the center of C_i' is at least $R_i + 2Err$ away from any of the neighboring approximate intersection points of P_{ij}^1 and P_{ij}^2 on C_j' (for brevity we will refer to those*

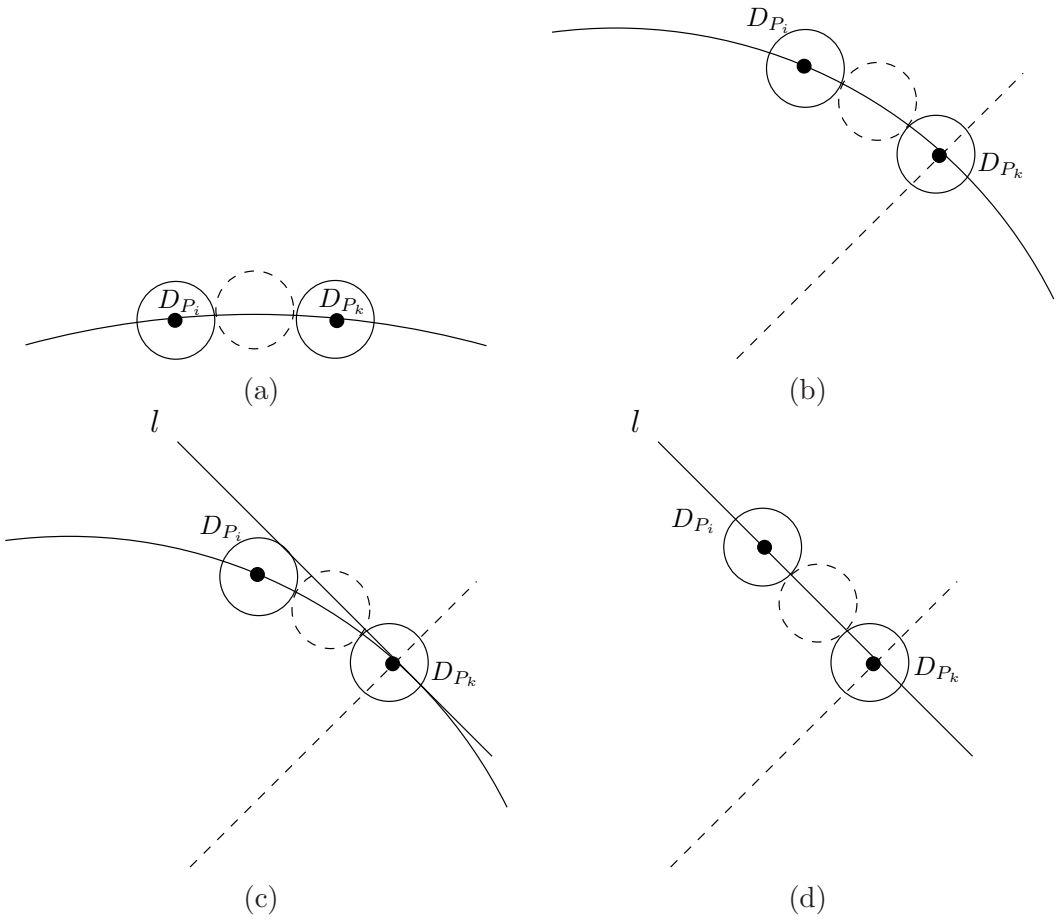


Figure 6.3: (a) P_i and P_k lie near the center of Γ. (b) P_i and P_k lie near the boundary of Γ. (c) Near the boundary of Γ, we can approximate the circle with a line l tangent to the circle at the boundary of Γ. (d) P_i and P_k lie on the line l.

points as the neighbors of P_{ij}^1 and P_{ij}^2), then the center of C_i' is at least $R_i + Err$ away from all the exact *intersection points on C_j' (except P_{ij}^1 and P_{ij}^2) — Figure 6.4. Consequently, it would be sufficient to check C_i' only against the four neighbors of P_{ij}^1 and P_{ij}^2, which could be done in $O(1)$ time (assuming that we already found the intersection points). Therefore, the* Circle - Intersection point *test for the circle C_i' would take overall $O(n)$ time.*

Proof. Since we require the center of C_i' to be at least $R_i + 2Err$ away from any of the neighboring *approximate* intersection points of P_{ij}^1 and P_{ij}^2, then the center of C_i' is at least $R_i + Err$ away from any of the neighboring *exact* intersection points of P_{ij}^1 and P_{ij}^2 (this issue is explained in Section 5.4). Thus we can inflate a disk of radius Err around each of the neighboring *exact* intersection points of P_{ij}^1 and P_{ij}^2 on C_j', and those disks will not intersect with C_i'. Denote this set of disks by $\mathcal{D}$.

Next, inflate a disk of radius Err around each other exact intersection points on C_j' (except P_{ij}^1 and P_{ij}^2) and denote this set of disks by $\mathcal{D}'$. Then we need to prove

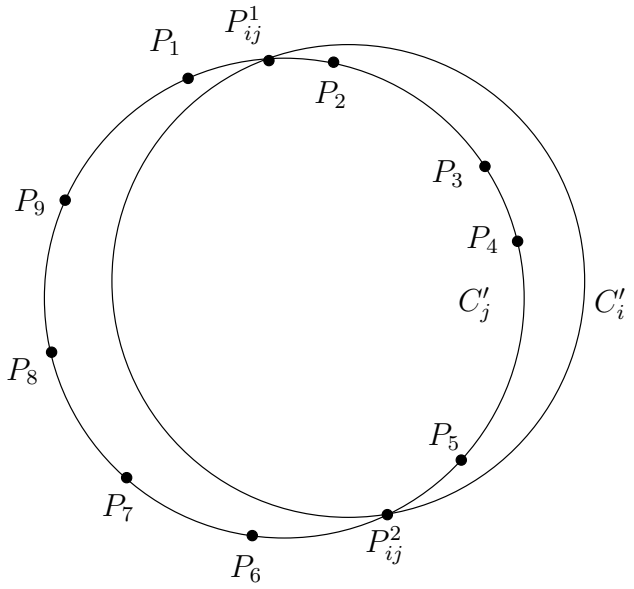


Figure 6.4: If the center of C'_i is at least $R_i + 2Err$ away from any of the neighbors of P_{ij}^1 and P_{ij}^2 (in this case, P_1, P_2, P_5 and P_6), then the center of C'_i is at least $R_i + Err$ away from all the *exact* intersection points on C'_j (except P_{ij}^1 and P_{ij}^2). Note that in the figure, we do not distinguish between the exact and approximate intersection points.

that C'_i cannot intersect the disks in $\mathcal{D}'$ without intersecting at least one of the disks in $\mathcal{D}$.

Let us assume that C'_i intersects a disk $D_k \in \mathcal{D}'$. Denote by Q_1 and Q_2 the intersection points of C'_i and D_k, such that Q_1 is the intersection point closer to P_{ij}^1, and Q_2 is the intersection point closer to P_{ij}^2. Denote by γ_1 the smaller portion of C'_i that lies between Q_1 and P_{ij}^1, and by γ_2 the smaller portion of C'_i that lies between Q_2 and P_{ij}^2 (Figure 6.5). At least one of the two following cases must hold: (i) Q_1 is the farthest point on γ_1 from C'_j (ii) Q_2 is the farthest point on γ_2 from C'_j (by "farthest" we mean that the shortest distance is the longest). If case (i) (resp. (ii)) holds, then γ_1 (resp. γ_2) intersects the disk of $\mathcal{D}$ whose center lies on γ_1 (resp. γ_2). Notice that such a disk must exist. □

Thus, we conclude that the *Circle - Intersection point* test for the newly added circle C_i can be done in $O(n)$ time (without searching for the intersection points), and the asymptotic expected total running time does not change.

6.2 The DCEL Structure

The collection of circles $\mathcal{C}'$ induces a planar subdivision. This subdivision consists of several type of elements

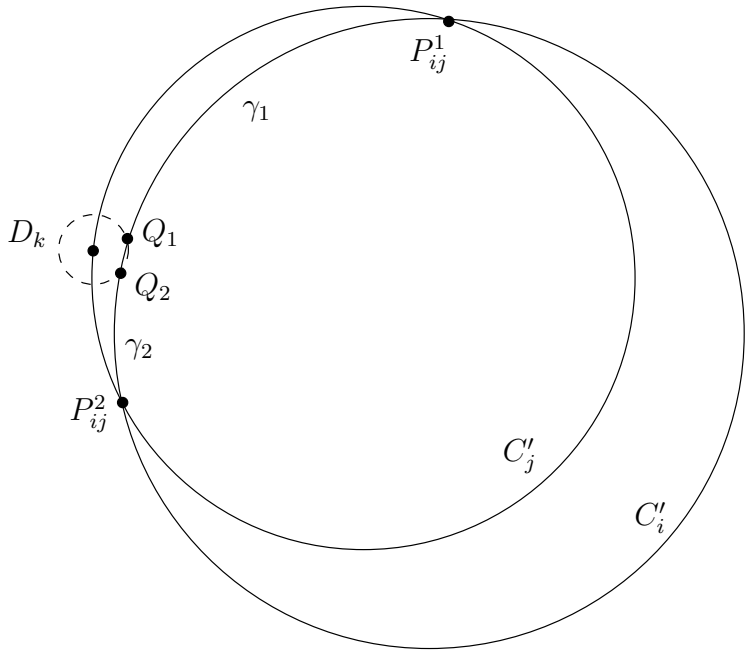


Figure 6.5: We denote two portions of C'_i as γ_1 and γ_2. If C'_i intersects a disk in $\mathcal{D}'$ then it intersects a disk in $\mathcal{D}$.

- 0-dimensional elements — the intersection points (vertices),
- 1-dimensional elements — the maximal circular arcs belonging to a single circle, not intersecting other circles (edges), and
- 2-dimensional elements — the maximal connected regions of $\mathbb{R}^2 \setminus \bigcup_{C' \in \mathcal{C}'}$ (faces).

We use the *doubly-connected edge list* structure (DCEL) to maintain the *topological* information of the subdivision and enhance it with *geometric* information (its planar embedding). An *island* is a connected set of circles. Every edge is represented by two *half-edges*, with opposite orientations. Two half-edges originating from the same edge are said to be twins. Each half-edge has pointers to its *twin* half-edge, *source* vertex, *target* vertex and to its incident face. Each face has pointers to its incident half-edge and to the list of *islands* which it contain (in the list we store pointers to incident half-edges of the islands). The DCEL is illustrated in Figure 6.6. See [8, Chapter 2] for more details.

In our program, there are several stages in the construction of the DCEL:

- half-edge creation,
- half-edge pointers setting,
- face creation, and

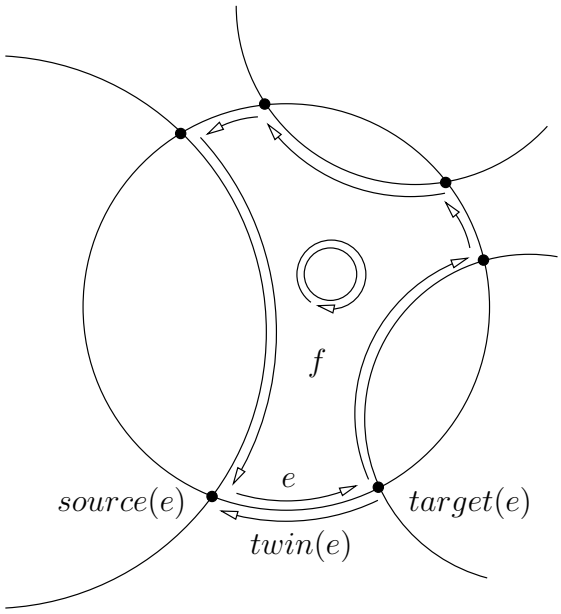


Figure 6.6: The DCEL structure. The face f contains one island.

- island adjustment.

We give detailed description of each stage:

- *Half-edge creation.* For each circle, we sort all the intersection points that lie on it, in clockwise order. For each pair of subsequent intersection points we create a half-edge and its twin. For each newly created half-edge we set the `twin, source` and `target` pointers. We defer the setting of the `next` pointer to the next stage. The overall complexity of this stage is $O(n^2)$ (there are $O(n^2)$ halfedges).

- *Half-edge pointers setting.* In this stage, we set the `next` pointer of each half-edge. First, since we performed the perturbation, *we know that each vertex will have a degree 4 exactly*, so each vertex involves half-edges originating from exactly two circles. Thus, we classify the intersection points into two types, as follows: given two circles C_1 and C_2, we use Eq. 5.7 to compute the intersection points. Define the intersection point given by $[x, y] = [X_1, Y_1] + s[X_2 - X_1, Y_2 - Y_1] + t[Y_2 - Y_1, X_1 - X_2]$ as type 1, and $[x, y] = [X_1, Y_1] + s[X_2 - X_1, Y_2 - Y_1] - t[Y_2 - Y_1, X_1 - X_2]$ as type 2 (Figure 6.7). An intersection point of type 1 is always to the left of the oriented line l passing through the centers of C_1 and C_2, in that order. This classification gives us sufficient information for setting the `next` pointers of the edges incident to the vertices induced by those intersection points. Figure 6.8 shows the `next` pointers setting for an intersection point of type 1. The complexity of this stage is $O(n^2)$ (there are $O(n^2)$ halfedges, and we handle each half-edge once and in constant time).

Remark. Notice that the intersection points classification is robust. The classification is such that, when we *add* the $t[Y_2-Y_1, X_1-X_2]$ term the point is classified as type 1 and when we *subtract* it, the point is classified as type 2. A possible mistake could occur if due to computation errors, the term $t[Y_2 - Y_1, X_1 - X_2]$ would have changed its sign, so the addition would in reality be a subtraction and vice versa. Yet, the term $t[Y_2 - Y_1, X_1 - X_2]$ cannot change it sign, since t is the result of a square-root operation, hence it is always non-negative. The terms $Y_2 - Y_1$ or $X_1 - X_2$ involves a single floating-point operation, thus they cannot change sign when Round To Nearest rounding mode is used. Therefore, we can conclude that the floating-point implementation of $t[Y_2 - Y_1, X_1 - X_2]$ is of the same sign as the exact one, and the classification is still valid.

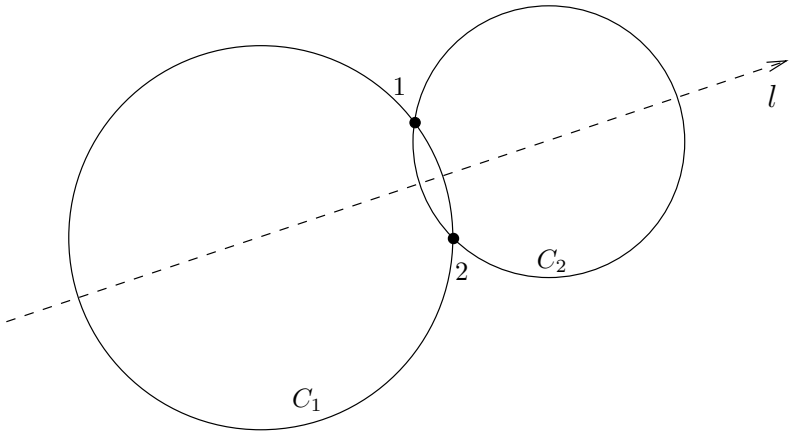


Figure 6.7: The classification to intersection points of type 1 and 2, and the oriented line l. Intersection point of type 1 is always to the left of l.

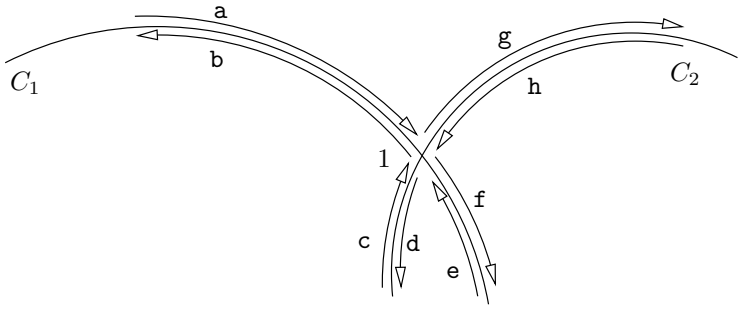


Figure 6.8: `next` pointers setting for an intersection point of type 1 — `a.next:=g`, `c.next:=b`, `e.next:=d`, `h.next:=f`.

- *Face creation.* We make another pass on all the half-edges, and for every half-edge `h`, if `h.incident_face` is NULL, we create a new face `f`, and set `f.incident_half-edge:=h` and `h.incident_face:=f`. We traverse the half-edges, using the `next` pointer, starting from `h.next`, and set their `h.incident_face` pointer to point to `f`. We do so until we return to `h`. We repeat this process until there is no half-edge that has an `incident_face` with a NULL pointer.

If a face `f` has the half-edges $h_1, \ldots, h_j$ pointing at it, and they are all in clockwise orientation then the connected set of circles that includes the underlying circles of $h_1, \ldots, h_j$ *could* be an *island*. In order to decide if `f` is an island, we do the following additional test: find the leftmost vertex `p` of `f`, and find its underlying circles, C_1 and C_2. Find the second intersection point of C_1 and C_2, and check if it is to the left of `p`, if so, then f is not an island (Figure 6.9). Following the remark on page 55, this step is robust.

During the face creation stage, we will also create a priority queue `Q`, that will hold pointers to all the islands. The key of the queue will be the topmost point in an island. Finding the topmost point in an island is done by adding the radius of each circle (in the island) to its center's y coordinate. Since the error of a single operation is minimal, and the fact that during the perturbation we eliminated all near tangencies, lead us to the conclusion that this stage of the algorithm is robust (the elimination of near tangencies has sufficiently "separated" the circles from one another). The complexity of this stage is $O(n^2)$ (there are $O(n^2)$ halfedges, and we handle each half-edge once).

- *Islands adjustment*. The final stage of the DCEL construction is to adjust the island lists. Recall that, at the previous stage, we built an islands priority queue `Q` (the island with the topmost point is given the highest priority). For each island `i` of `Q`, we perform *point location* (Section 6.3) using the topmost point of `i`. We find the face which contains the point, and we add the island i into its islands list. Notice that islands can be nested within other islands. The islands which are not contained in other islands are called *continents* and they are islands of the *unbounded face*. The point location mechanism is described below. The complexity of a point location query is $O(n)$, and there are at most n islands, thus the complexity of this stage is $O(n^2)$.

Since the complexity of each stage is $O(n^2)$, we conclude that the DCEL construction is done in $O(n^2)$ time which is worst-case optimal since the complexity of the arrangement can be $\Omega(n^2)$.

6.3 Point Location

A basic requirement from a subdivision data structure is to support point location. That is, given a query point p, we wish to locate the face f which contains p.

In [8], an efficient point location strategy is presented, which can answer queries in expected $O(\log n)$ time. However, the implementation of such a point location-strategy is rather intricate. Here, we use a very simple point location strategy. It is easy to implement and we bring it here, to point out the robustness-related issues in answering point location queries.

Given a query point p, we shoot a vertical ray from p. That is, we find the closest intersection point q of an upward directed vertical ray emanating from p and a circle

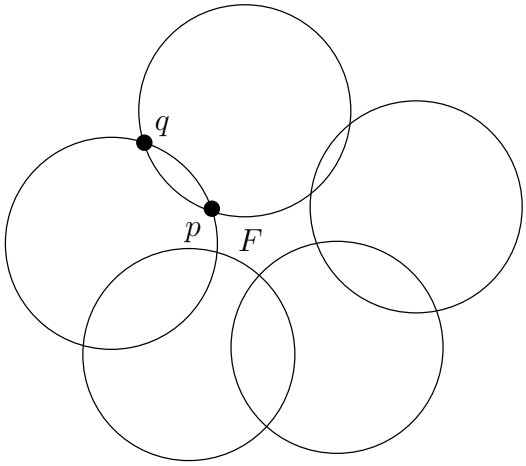


Figure 6.9: We want to check if F is an island. We find the leftmost vertex p and its underlying circles, and compute the second intersection point q and compare their x coordinates. Since q lies to the left of p, F is not an island.

C'_i of $\mathcal{C}'$ (Figure 6.10). The answer to the query, is the incident halfedge of q, which points to the face that contains p. If there is no circle above p, then p belongs to the unbounded face. We can find q by computing all the intersection points of the vertical ray emanating from p with circles in $\mathcal{C}'$, while maintaining the intersection point with the minimum y coordinate. This step might take $O(n)$ time (recall that n is the number of circles in $\mathcal{C}'$).

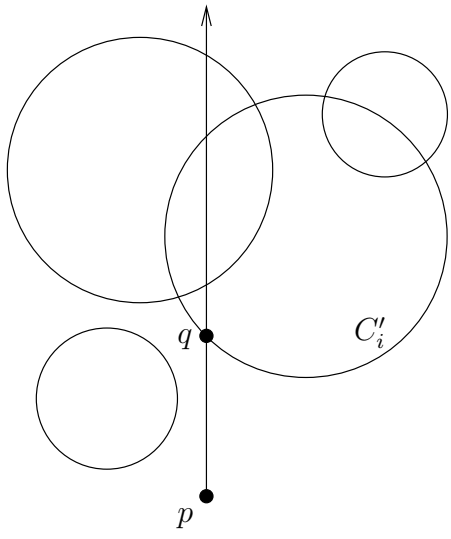


Figure 6.10: We shoot a vertical ray from p, and find the closest intersection point q.

Since the vertical ray shooting is carried out using floating-point arithmetic, errors may occur, and we could get a false result. Hence, using the same method that we applied in the computation of Err (Section 5.4), we will compute the error of the *line-circle* intersection point, denoted by Err' (in this case, there is no need to multiply by

$\sqrt{2}$). We will first check that p is not nearly tangent to any circle in the arrangement, if this is the case — we will reject this query (because the perturbation eliminates near tangencies, this could never be the case for the "islands adjustment" stage of the DCEL construction). In the "islands adjustment" stage, we ignore the circles that belong to the island being adjusted.

Let $q_1 \ldots q_m$ be the intersection points that we computed during vertical ray shooting (namely the intersection points of circles in $\mathcal{C}'$ with the upward vertical ray emanating from p), sorted by their y coordinate, such that q_1 is the closest point to p. If the distance between q_1 and q_2 is at least $2Err'$, then we can certify that q_1 is indeed the intersection point that we seek.

If the distance between q_1 and q_2 is less than $2Err'$, and the distance between q_1 and q_3 is at least $2Err'$, then the floating-point errors may cause us to switch the order of the *exact* intersection points q_1 and q_2 (note that q_1 and q_2 are the only candidates to be the point q that we seek). We can resolve this possible mis-ordering of q_1 and q_2, by the following additional test. Let C'_j be the circle associated with q_1, and let C'_k be the circle associated with q_2. C'_j and C'_k partition the plane into at most four distinct regions:

- Λ_1 — The region which is outside both circles.
- Λ_2 — The region which is inside both circles.
- Λ_3 — The region which is inside C'_k and outside C'_j.
- Λ_4 — The region which is inside C'_j and outside C'_k.

Hence, we can check to which region the query point p belongs, and verify that the result is consistent with the ordering of q_1 and q_2 (recall that, we first check that p is not nearly tangent to any circle in the arrangement). Figure 6.11 exemplifies this case.

We shall now describe a method that will allow us to verify that for a specific floating-point precision and specific M and ξ values (M and ξ were described in previous chapters), there can be no configuration in which the distance between q_1 and q_3 is less than $2Err'$ (thus, only the case where q_1 and q_2 switch order may occur, which we already know how to resolve).

The idea is to transform each circle $C'_i \in \mathcal{C}'$ into an annulus A_i with the same center as C'_i and radii $max(0, R_i - Err')$ and $R_i + Err'$. We need to prove that three annuli cannot intersect in a common point. Recall that during the perturbation process, we make sure that around each exact intersection point, we can inflate a disk of radius Err, that no other circle is allowed to intersect (this is the *Circle - Intersection point* test, Section 5.4).

For two circles, C'_k and C'_j, denote by p_1 and p_2 their two exact intersection points, and by D_1 and D_2 the disks of radius $Err - Err'$ inflated around p_1 and p_2.

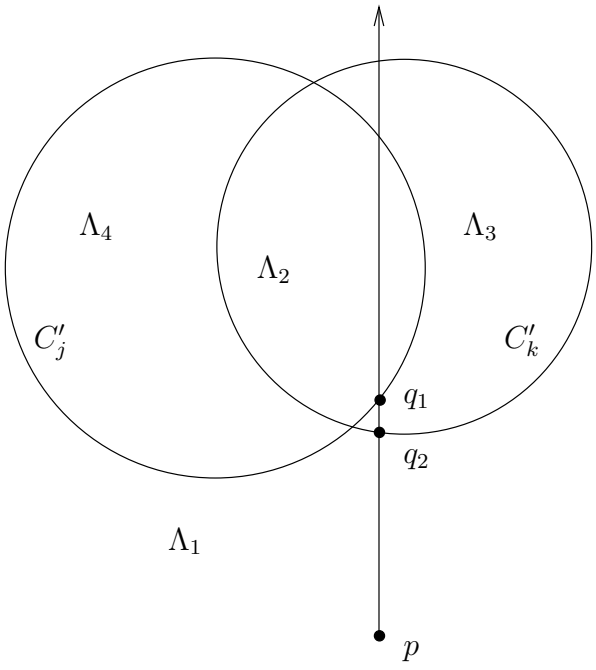
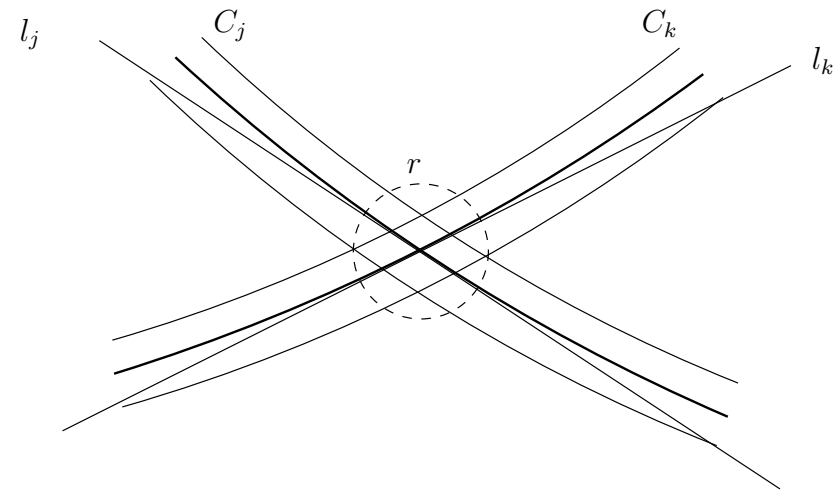


Figure 6.11: Due to the mis-ordering of q_1 and q_2, p could be falsely judged to be in the face that lies in the region Λ_3. Yet, testing p would reveal that it is in fact in the region Λ_1, and the ordering of q_1 and q_2 can be corrected.

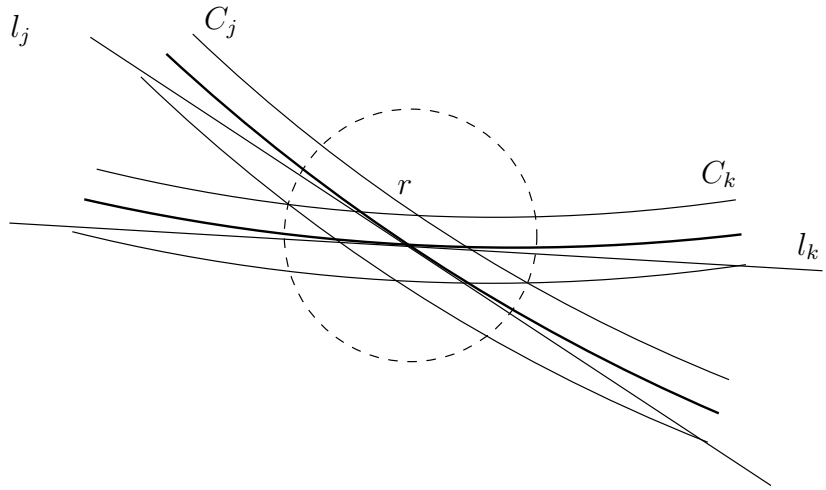
We need to prove that the intersection of A_k and A_j lies completely in D_1 and D_2. In the following lemma, we shall describe a construction that yields a bound on the diameter of the circumcircle that contains the intersection of A_k and A_j (around one of the intersection points of C'_j and C'_k).

Lemma 7 *For a given floating-point precision and specific M and ξ values, let C_1 and C_2 be circles with $X_1 := -\frac{\xi}{2}, X_2 := \frac{\xi}{2}, Y_1 := 0, Y_2 := 0, R_1 := M, R_2 := M$. The diameter of the circumcircle that contains the intersection of A_1 and A_2 (around one of the intersection points of C_1 and C_2) is a bound on all such diameters.*

Proof. Given two circles, C_j and C_k, without loss of generality we shall only concern ourselves with one of their intersection points, denoted by r. Denote by l_j (resp. l_k) the tangent line to C_j (resp. C_k) at r (Figure 6.12 (a)). We observe the following: The diameter of the circumcircle that contains the intersection of A_k and A_j around r increases as the angle between l_k and l_j decreases (Figure 6.12 (b)). If the centers of two equal radii circles are an infinitesimal distance apart, then the angle between the tangent lines is infinitesimal small. If we move the centers such that they are ξ apart, then the angle will increase. Any further separation will only increase the angle. On the contrary, increasing the radii of the circles will decrease the angle. If the circles are not of equal radii, then increasing the smaller radius will decrease the angle. Hence, we conclude that the construction given in the lemma, causes the angle between the tangent lines l_k and l_j to be the smallest, thus maximizing the diameter of the circumcircle. □



(a)



(b)

Figure 6.12: (a) The dashed circle is the circumcircle that contains the intersection of A_j and A_k around the point r. (b) The diameter of the circumcircle that contains the intersection of A_j and A_k increases as the angle between l_k and l_j decreases (C_k has been rotated around r).

Thus, after we are given specific values for the floating-point precision and specific M and ξ values, we can compute the bound Err'' on the circumcircle's diameter, and we check that the following holds:

$$Err'' \leq Err - Err' \tag{6.1}$$

(the circumcircle whose diameter we have bounded, must contain r).

If Inequality 6.1 fails to hold, then we should either decrease the value of M, or increase the floating-point point precision or the value of ξ. If Inequality 6.1 does hold, then the point-location mechanism that was described above could be used.

Remark. The computation of Err'' should be done using exact computation (or

by using one of the floating-point error bounding methods that we have described). For a specific precision and M and ξ values, this computation needs be done just *once*, as an *offline processes*. Mathematical software (such as *Mathematica* or *Maple*) can be used to perform it.

Chapter 7

Experimental Results

In this chapter we report on experimental results with our implementation of the perturbation scheme that was described above. We implemented the perturbation scheme as a set of C++ classes. We also implemented the DCEL construction (Doubly Connected Edge List, see [8, Chapter 2] for details on this data structure) with the simple point-location mechanism described in Chapter 6. After the perturbation, our program assumes that the circles are in general position, thus it avoids handling the different special cases, that would have been needed to handle degenerate inputs.

As was already stated in Section 4.2, the bound on δ that we computed in Chapter 4 is crude. As a heuristic, in our implementation, we first set δ to be 2ε. After a constant number of failed attempts to find a valid placement for the currently inserted circle, we set $\delta := 2\delta$ and again, after a constant number of failed attempts, we set $\delta := 2\delta$, until we find a valid location for the current circle. Thus, we may end up at the bound that was computed in Chapter 4 after $\lceil \log_2 \frac{\delta}{\varepsilon} \rceil$ attempts. So, the running time may increase by a multiplicative factor of $O(\log \delta)$ (notice that ε is independent of the input size n).

We have tested our program on several inputs :

- `grid`, a grid of 320 circles, which involves many inner and outer tangencies (Figure 7.1 (a)),
- `flower`, a "flower" composed of 40 circles, all intersecting in a common point (Figure 7.1 (b)),
- `rand_sparse`, a collection of 40 random circles (Figure 7.2 (a)),
- `rand_100`, a collection of 100 random circles (Figure 7.2 (b)),
- `rand_1000`, a collection of 1000 random circles (Figure 7.3 (a)),
- `rand_2000`, a collection of 2000 random circles (Figure 7.3 (b)), and
- `rand_10000`, a collection of 10000 random circles.

The first two data sets, `grid` and `flower` are highly degenerate, `rand_sparse` and `rand_100` are two types of random data sets (the parameters of each circle were chosen randomly). The last three inputs consist of huge (several thousands circles) random data sets (again, the parameters of each circle were chosen randomly).

For the random data sets, all the input parameters are given as integers (to "promote" degeneracies). The properties of each input data set are given in Table 7.1. The results of the perturbation and running times for those inputs are give in Table 7.2, with the IEEE double number type and the bound ε computed using $M = 1000$ and $\xi = 0.03$. The tests have been performed on an Intel Pentium III 1 GHz machine with 2 GB RAM, operating under Linux Redhat 7.3 using gcc 2.95.3. Table 7.3 shows the number of near degeneracies that were handled for each input (in a single run of the algorithm). Table 7.4 shows the properties of the DCEL structures that were computed for each input (in a single run of the algorithm).

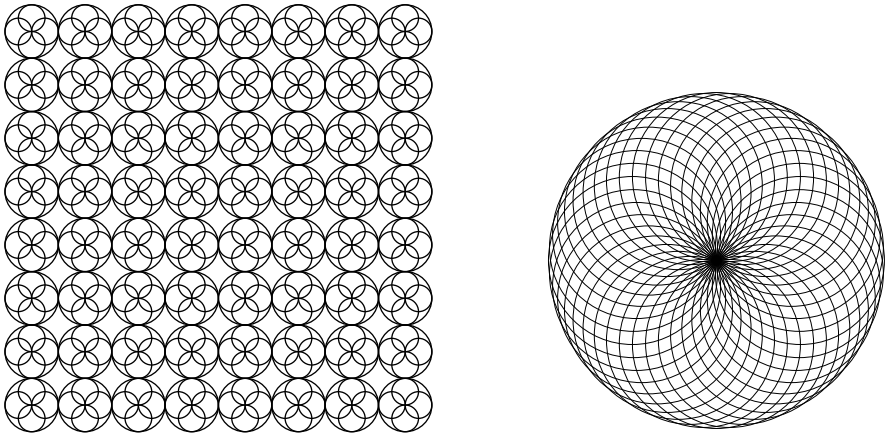
Notice that for the `flower` input, the largest perturbation has occurred although the input contains only 40 circles. The reason lies in the fact that circle C_i adds $2(i-1)$ new intersection points many of them very close to the center of the "flower". For the last circles there are already ≈ 1000 existing intersection points, which forces the newly added intersection points (induced by those last circles) to be rather far from the center of the "flower".

We have also constructed an additional data set, `mini_example`, which contains four circles (Figure 7.4). The parameters of those circles are given in Table 7.5. Although the circles are in general position even without the perturbation, the DCEL construction fails, due to an error in the point-location mechanism (it falsely finds an intersection point tha lies *below* the query point). When the perturbation is applied, the DCEL is constructed correctly. This example illustrates the importance of handling the robustness issues, that arise from the use of finite precision arithmetic.

Remark. The fifth column of Table 7.3 shows that for all the examples, degeneracy of type 4 (the centers of two intersecting circles are too close) was not detected. Notice that this not always the case, as is shown in the simple example, whose data is given in Table 7.6. However, it appears that in many cases, resolving degeneracy of type 2 (inner tangency) also eliminates degeneracy of type 4.

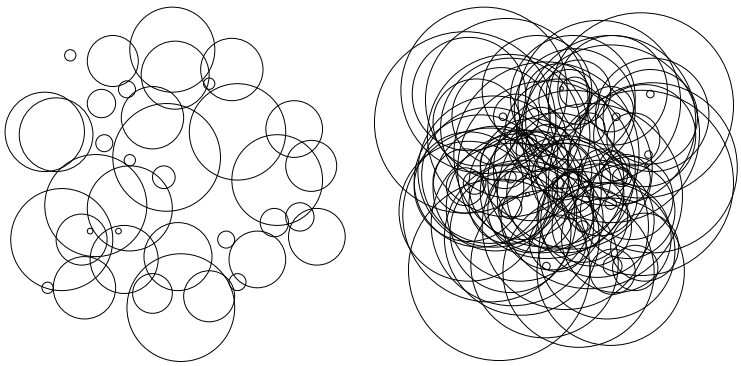
Name	n	R	$M - \Delta$
`grid`	320	10	140
`flower`	40	100	100
`rand_sparse`	40	20	100
`rand_100`	100	49	100
`rand_1000`	1000	100	1000
`rand_2000`	2000	100	1000
`rand_10000`	10000	35	1000

Table 7.1: n denotes the number of circles, R denotes the maximum radius and $M-\Delta$ is the maximum input size minus Δ.



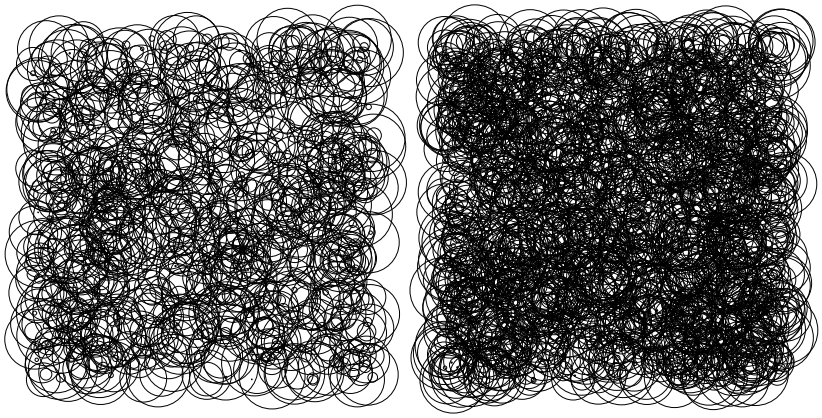
(a) (b)

Figure 7.1: (a) A grid of 320 circles, which involves many inner and outer tangencies. (b) A "flower" composed of 40 circles, all intersecting in a common point (the origin).



(a) (b)

Figure 7.2: (a) A collection of 40 random circles. (b) A collection of 100 random circles.



(a) (b)

Figure 7.3: (a) A collection of 1000 random circles. (b) A collection of 2000 random circles.

name	avg.	max.	var.	p_time	t_time
grid	0.1122	0.6320	0.0101	0.1060	0.1140
flower	1.0359	4.2529	0.9586	0.1280	0.1360
rand_sparse	0.0424	0.0493	0.0000	0.0000	0.0020
rand_100	0.0597	0.4017	0.0044	0.1100	0.1300
rand_1000	0.0497	0.3994	0.0015	0.4000	0.5560
rand_2000	0.1815	1.0856	0.0070	2.1540	2.8040
rand_10000	0.3412	1.4527	0.0172	6.3560	9.4780

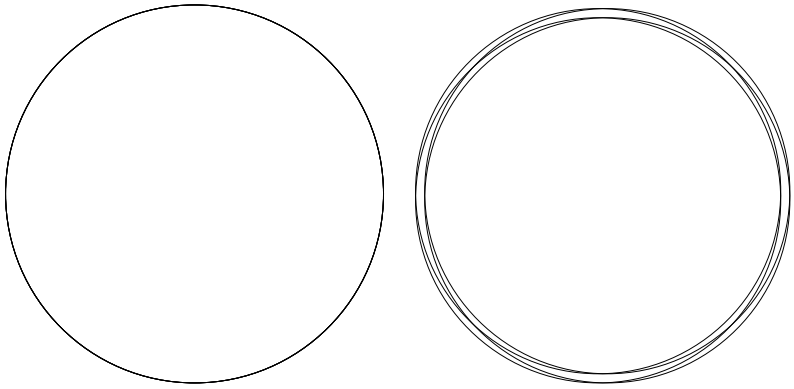
Table 7.2: Avg. denotes the average perturbation size, max. denotes the maximum perturbation size, var. denotes the perturbation variance, p_time denotes the time of the perturbation (in seconds) and t_time denotes the total (perturbation and DCEL construction) time (in seconds). All the given results are from averaging the results of 5 tests for each input.

Name	type 1	type 2	type 3	type 4	total
grid	137	31	94	0	262
flower	0	0	4701	0	4701
rand_sparse	2	0	0	0	2
rand_100	1	5	80	0	86
rand_1000	6	2	169	0	177
rand_2000	7	4	2222	0	2233
rand_10000	229	150	14850	0	15229

Table 7.3: The number of near degeneracies that were handled for each input (in a single run of the algorithm).

Name	#vertices	#halfedges	#faces
`grid`	1324	5296	1326
`flower`	1490	5960	1492
`rand_sparse`	110	458	121
`rand_100`	3566	14266	3569
`rand_1000`	28342	113404	28362
`rand_2000`	110790	443220	110822
`rand_10000`	346954	1388506	347301

Table 7.4: The properties of the arrangements that were computed for each input (in a single run of the algorithm).



(a) (b)

Figure 7.4: (a) A collection of four circles, whose parameters are given in Table 7.5. (b) A similar arrangement, yet with different parameters, such that the structure of the arrangement is more evident.

i	X_i	Y_i	R_i
1	1000.000000	1000.000000	1000
2	1000.000001	1000.000000	1000
3	1000.000000	1000.000001	1000
4	1000.000001	1000.000001	1000

Table 7.5: The parameters of the circles in `mini_example`, the example in Figure 7.4.

i	X_i	Y_i	R_i
1	0.0	0.0	1000
2	0.02	0.0	1000

Table 7.6: The parameters of the circles that will cause a degeneracy of type 4 to arise, when using the IEEE double number type and the bound ε computed using $M = 1000$ and $\xi = 0.03$.

Chapter 8

Discussion

In this chapter we discuss certain key aspects of the scheme and insights that we gained throughout the work.

The Resolution Bound. As was already stated, there are two key parameters that govern the perturbation scheme — the resolution bound and the perturbation bound. The perturbation bound is important (as described in the next item), yet the resolution bound is crucial, since it certifies the *correctness* of the arrangement, thus allowing us to create a robust algorithm, which is our main goal in this work.

For a given predicate, it should be clear that the resolution bound is not simply the size of the maximal error when evaluating this predicate with floating-point arithmetic. In fact, computing the resolution bound requires a careful understanding of the geometry behind the predicate. In some cases, finding a geometric interpretation that will suit our perturbation scheme, could force us to use a more complicated predicate with a possibly larger maximal error.

For example, we could use the following method to devise a predicate for detecting degeneracy of type 3: Given the circles C_i,C_j and C_k, denote by L_{ij} (resp. L_{ik}) the radical axis of C_i and C_j (resp. C_i and C_k). If the intersection point p of L_{ij} and L_{ik} lies on C_i, then the degeneracy occurs (Figure 8.1). The advantage of such a predicate, is that it is fairly easy to compute the radical axis of two circles, thus the error of such an expression would be relatively small. Yet, deriving the resolution bound in this case seems highly non-trivial.

Using the Perturbation Bound. Recall that, in our implementation, we start by setting $\delta := 2\varepsilon$, and increase it until we succeed in finding a valid placement for the currently inserted circle. Thus, we did not really use the perturbation bound, as it was described in Chapter 4. Yet, the perturbation bound is important for the following reasons:

- Since it is a key parameter in the controlled perturbation scheme, it is interesting from a theoretical point of view.

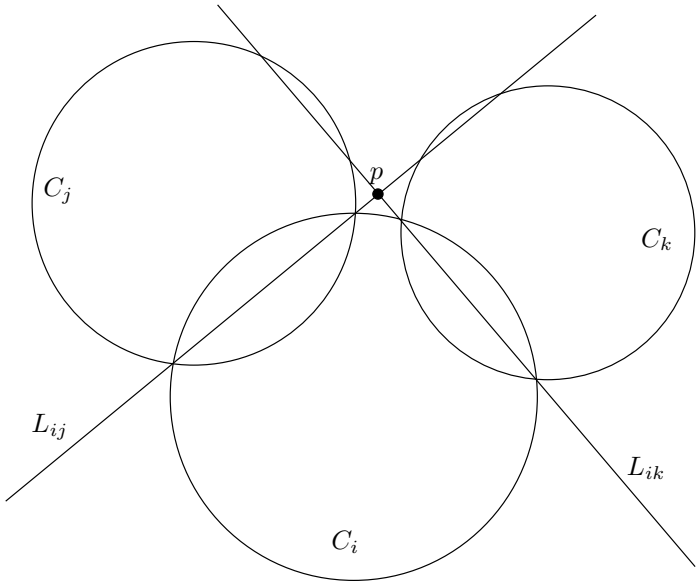


Figure 8.1: If the intersection point p of L_{ij} and L_{ik} lies on C_i, then degeneracy of type 3 occurs.

- If an application requires a certain level of accuracy, the perturbation bound can be used to predetermine the length of the mantissa needed to achieve that accuracy.
- It is used to establish the expected running time of the algorithm.

Number Type Selection. In this work, we have handled arrangements of circles. The handling of this type of arrangements using finite precision arithmetic is considered difficult, and no "complete" solution is known. Our algorithm with the standard IEEE double, still yields reasonable results. If it does not, then given a required accuracy, we can predetermine the length of the mantissa needed to achieve that accuracy. There are several available libraries that provide number types with a specified length of mantissa (for example, GMP [17], CORE [26] and LEDA [28]). Notice, that there is no need to increase the length of the mantissa during run time. It appears, that for arrangements of more complicated objects (i.e., conics or algebraic curves) such libraries should be used.

Separation Bounds. A number $\alpha \in \mathbb{R}$ is called an *algebraic number* if there exist integers $\alpha_0, \ldots \alpha_d$ such that α is a root of the polynomial $p(x) = \sum_{k=0}^{d} \alpha_k x^k$. The LEDA [28] and CORE [26] libraries both allow the use of an algebraic number type. The number is represented internally using an expression DAG whose inner nodes represent binary or unary operations, and whose leaves represent constant integers (the input operands). Each $\sqrt[k]{\square}$ node is given a *weight* of k, and all other nodes are given a weight of 1 (Figure 8.2).

To devise a robust algorithm using the exact computation paradigm, it is crucial that all comparisons between two algebraic numbers are carried out accurately. Given

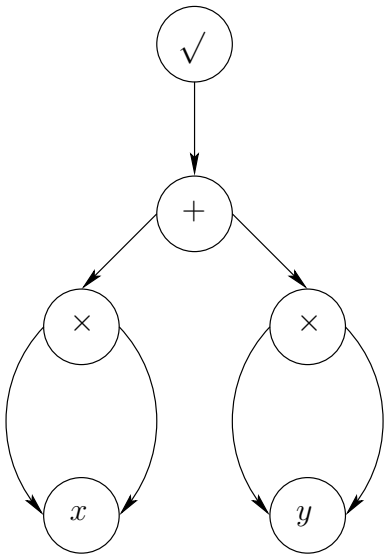


Figure 8.2: An expression DAG representing the expression $\sqrt{x^2 + y^2}$. In this DAG the root has weight 2, whereas all the other nodes have weight 1.

two expressions E_1 and E_2 we can determine if $E_1 > E_2, E_1 < E_2$ or $E_1 = E_2$ by evaluating the sign of the expression $E_1 - E_2$. Hence, we require an accurate sign evaluating operation. Such accurate sign evaluation of DAG expressions can be done using the *separation bound* theory. The separation bound is an easily computable function such that any non-zero expression E is lower bounded by it. A detailed explanation of this theory is given in [6].

It is interesting to investigate what would be the separation bound needed in the construction of arrangements of circles. The most elementary operation, needed in a naive construction of such arrangements, it to compute the intersection points of two circles, and perform some comparison operations with those points later on. Using Eq. 5.7 to compute those intersection points, a comparison operation which involves such points would yield a separation bound that requires $17 \log N$ bits, where N is the maximal *integer* number allowed ($\log N$ is the number of bits required to represent N). In our implementation we use a floating-point number with mantissa of length p, thus the separation bound would be at least $17p$ (notice however, that using such a separation bound will allow us to compute with far greater resolution). As a concrete example, for the standard double number type, p is 53, so the separation bound would require 901 bits.

Chapter 9

Conclusions

Controlled perturbation was presented several years ago, and was already applied to certain types of arrangements. The main contribution of the current work is the derivation of a good (small) resolution bound, that is, a bound on the minimum separation of features of the arrangement that is required to guarantee that the predicates involved in the construction can be safely computed with the given (limited) precision arithmetic. A smaller resolution bound leads to smaller perturbation of the original input. We implemented the perturbation scheme and the construction of the arrangement.

There are still many further research directions, that could shed further light on the controlled perturbation scheme. Among those directions:

- **Optimal insertion ordering.** In the controlled perturbation scheme, we insert the objects (in our case, circles) one by one. Is there an optimal insertion order? optimal in the sense of minimizing the perturbation size, or in the sense of reducing the running time.
- **Arrangements of unbounded objects.** In the work that was done so far on controlled perturbation (including the current work), the arrangements that were handled were of bounded objects (i.e., spheres, circles, segments and polyhedra). The handling of arrangements of unbounded objects (such as lines), raises some new difficulties in the computation of the resolution bound and the perturbation bound. For example, in arrangements of lines, the intersection points can be arbitrarily far from the origin, hence, their coordinate values can be very large.
- **Arrangements of algebraic curves (Figure 9.1).** Some of the ideas presented in this work, could be a starting point for applying controlled perturbation to arrangements of more complicated algebraic curves. Algebraic curves of high degree are much more difficult to handle than circles and circular arcs (e.g., computing the intersection points), so advanced techniques would be needed in the computation of the resolution bound and the perturbation bound.

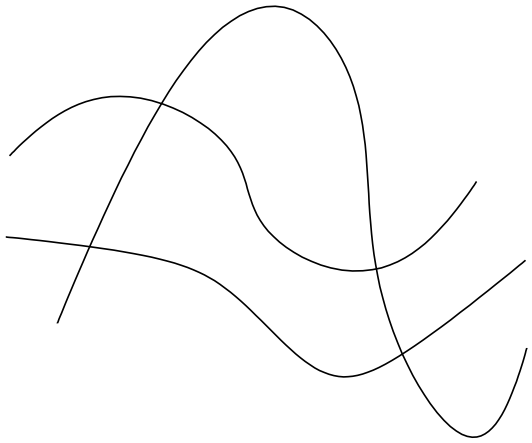


Figure 9.1: An arrangements of algebraic curves.

- **Applying controlled perturbation to other types of geometric algorithm.** Thus far, controlled perturbation was applied only for algorithms that construct arrangements. It appears that controlled perturbation could also be applied to other types of geometric algorithms such as convex hull computation, triangulations and Voronoi diagrams construction (see Appendix B for an example).

Appendix A

Extension I: Arrangements of Circular Arcs

In this appendix, we describe how our perturbation scheme can be extended to allow the construction of arrangements of circular arcs (Figure A.1). Such arrangements are a generalization of the arrangements that we dealt with in the thesis. We propose an "easy" extension (requiring only minor modifications in the algorithm that was presented for full circles) which has some drawbacks, that we will mention below.

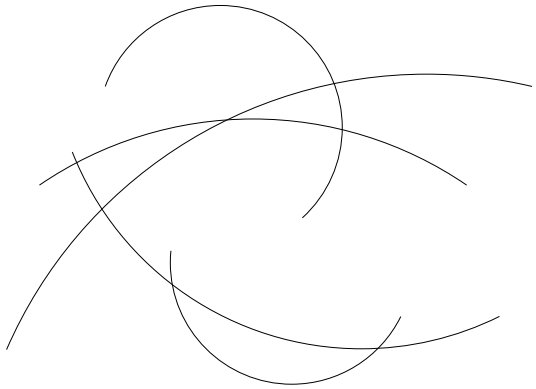


Figure A.1: An arrangement of circular arcs.

We start by examining the possible degeneracies. There are four[1] types of degeneracies in an arrangement of circular arcs (the first three also arise in the case of full circles):

1. An outer tangency between two arcs.

2. An inner tangency between two arcs.

[1]We are not including the degeneracy that occurs when the centers of the underlying circles are too close, although we *do* take it into consideration in our solution (through degeneracy of type 3).

3. Three arcs intersect in a common point.

4. The endpoint of an arc lies on another arc (Figure A.2).

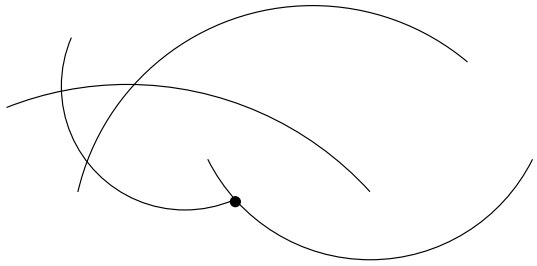


Figure A.2: An arrangement of circular arcs with degeneracy of type 4.

Define an arc A_i by its underlying circle C_i and its two endpoints P_i^1 and P_i^2. We assume that the arc is oriented clockwise, going from P_i^1 to P_i^2. We specify an endpoint by its x coordinate, and then we compute the y coordinate (similarly, one can start from the y coordinate). That is, we compute the y coordinates using the x coordinates.

In order to build the arrangement $\mathcal{A}'$ of the perturbed arcs $A'_i, i = 1 \ldots n$, we will use the efficient circles perturbation algorithm that was given in Section 6.1 (we will refer to it as the circles perturbation algorithm), as follows. For each arc A_i that we add, we first insert its end-points P_i^1 to P_i^2 to the appropriate binary trees (the upper/lower/left/right tree) of its underlying circle C_i. We then add C_i to the arrangement, while keeping the information of the arc A_i. This arrangement can be regarded as the one that we described for full circles, enhanced with the information on the arcs (e.g., for each arc we need to know which of the points in its binary trees indicate the first and second endpoints). If an arc A_i needs to be perturbed, we will translate it by relocating the center of its underlying circle C_i, and re-computing the endpoints P_i^1 and P_i^2 (Figure A.3). Notice however, that when we apply the circles perturbation algorithm, we also account for the degeneracies that occur, when the centers of the underlying circles are too close.

After inserting all the arcs, we proceed with the construction of the DCEL structure, in a similar manner to the one that was describe in Section 6.2. Notice, that the DCEL construction process should be adjusted, such that it will build the arrangement of arcs (and not the underlying circles). We omit the straightforward details of this modification here.

Remark. We would also need to slightly modify the computation of δ, to take into account the forbidden regions implied by the arcs end-points.

We now prove the following lemma:

Lemma 8 *Using the perturbation algorithm that was described in Section 6.1, with the modifications described above, allows us to robustly construct arrangements of circular arcs.*

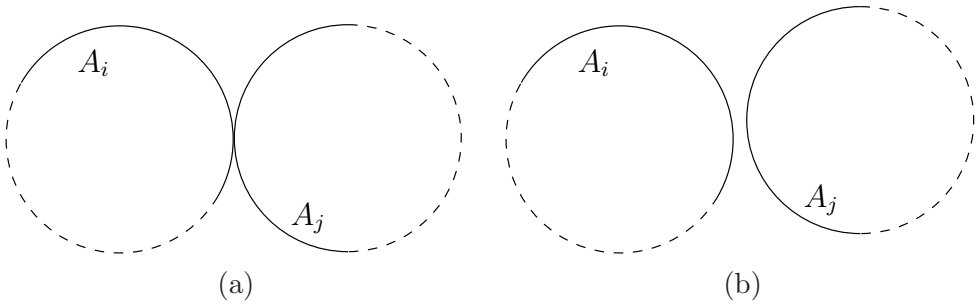


Figure A.3: (a) The arcs A_i and A_j are in a state of outer tangency (the underlying circles are dashed). (b) The arc A_j is perturbed to avoid the degeneracy.

Proof. We shall prove the lemma by showing that each degeneracy is handled correctly by the circle perturbation algorithm.

For two arcs A'_i and A'_j, outer and inner tangencies would imply that their underlying circles C_i and C_j are in a tangent state. The circle perturbation algorithm assures us that there will be no such tangencies.

Similarly, if three perturbed arcs A'_i,A'_j and A'_k intersect in a common point, then so do their underlying circles C_i,C_j and C_k. Again, The circle perturbation algorithm assures us that no three circles intersect in a common point.

In order for degeneracy of type 4 to occur (where the endpoint of an arc lies on another arc) for two arcs A'_i and A'_j, the involved endpoint must lie on the intersection point of the underlying circles C_i and C_j. The error of the computation of the endpoints P_i^1 and P_i^2 (also after possible perturbation of the underlying circle) is at a much smaller scale than that of the circle-circle intersection point error Err that was computed in Section 5.4 (computing the x coordinates involves just one addition or subtraction operation, and the extraction of the y coordinate from the circle equation is also easier than the circle-circle intersection). Thus, we can imagine that we inflate a disk of radius Err around all the arcs end-points, and the same analysis that we employed in Section 5.4 could be applied here. In other words, we regard P_i^1 and P_i^2 as circle-circle intersection points, so the circles perturbation algorithm will verify that they will not lie too close to any arc-arc intersection points (their disks will not overlap). □

There are obvious drawbacks to this simple approach. First, there can be unnecessary perturbation. Since we base our tests on the underlying circles, we might falsely deduce that there is a degeneracy between the arcs. Such a false degeneracy is illustrated in Figure A.4. However, the measure of the perturbation is still bounded by Δ. The second drawback is that the arrangement of the underlying circles can have a larger complexity than the arrangement of arcs (Figure A.5). Yet, both arrangements have the same worst-case asymptotic complexity.

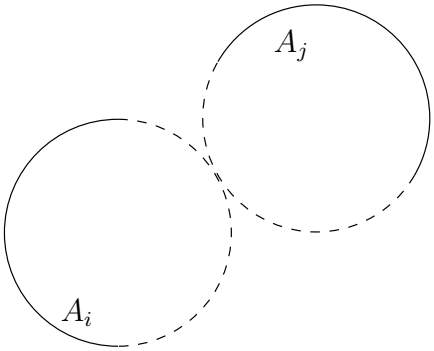


Figure A.4: Since we base our tests on the underlying circles, we might falsely deduce that there is a degeneracy between the arcs.

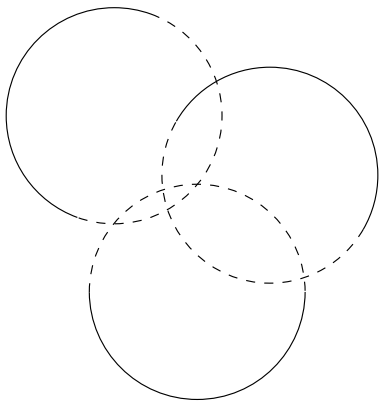


Figure A.5: The arrangement of the underlying circles can have a larger complexity than the arrangement of the arcs.

Appendix B

Extension II: Delaunay Triangulation and the Incircle Predicate

Given a set of points $\mathcal{P}$ in the plane, the unique triangulation $\mathcal{T}$ that satisfies the property that for each triangle, its circumcircle contains no other points of $\mathcal{P}$, is called a *Delaunay triangulation* (Figure B.1). The Delaunay triangulation satisfies several important properties:

1. It is the dual of the *Voronoi diagram* of $\mathcal{P}$.
2. It is related to the three dimensional convex hull of a lifted copy of $\mathcal{P}$.
3. It maximizes the minimum angle among all the possible triangulations of $\mathcal{P}$.

Delaunay triangulations play a fundamental role in many applications in computer graphics, computational geometry, CAD/CAM and more. Much work has been done to find efficient algorithms to compute the Delaunay triangulation. An experimental comparison of a number of different algorithms for computing the Delaunay triangulation is given in [39].

In this appendix we will focus on the randomized incremental construction algorithm [19] (the algorithm is also described in [8, Chapter 8]). This algorithm takes expected $O(n \log n)$ time. This algorithm (like numerous other algorithms) uses the `incircle` predicate. Given three points p_1, p_2 and p_3 and a query point q, the `incircle` predicate tells if the point q is inside of the circle defined by p_1, p_2 and p_3 (Figure B.2). A degeneracy occurs when q lies on the circumcircle. To avoid such a degeneracy we shall move q to be at least ε away from the circumcircle. Notice that applying controlled perturbation to the randomized incremental construction algorithm is not a trivial matter. In fact, doing the perturbation while maintaining the expected $O(n \log n)$ time (without any assumptions on the input) remains an open

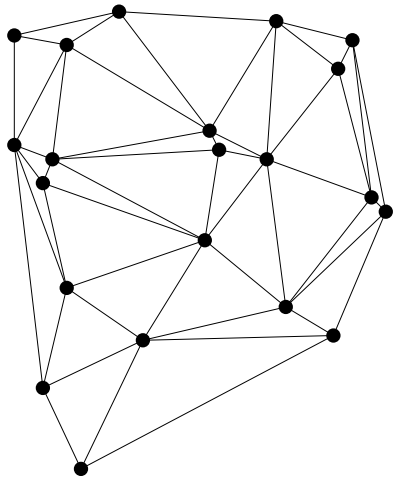


Figure B.1: A Delaunay triangulation.

question for future research. Here we will only show how to derive a resolution bound for the `incircle` predicate.

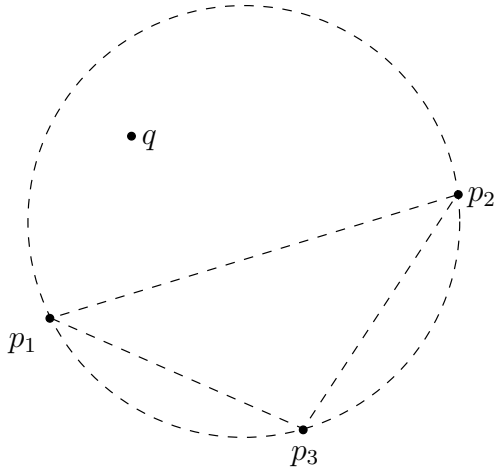


Figure B.2: Given three points p_1, p_2 and p_3 and a query point q, the `incircle` predicate tells if the point q is inside of the circle defined by p_1, p_2 and p_3 (which is the circumcircle of the triangle defined by p_1, p_2 and p_3).

The Resolution Bound for the Incircle Predicate

For a point p_i, denote by X_i and Y_i its x and y coordinates. Given three input points p_1,p_2,p_3 and a query point p_i, the expression E that we use in evaluating the `incircle` predicate is defined as follows [30]:

$$E = \begin{vmatrix} 1 & X_i & Y_i & X_i^2+Y_i^2 \\ 1 & X_1 & Y_1 & X_1^2+Y_1^2 \\ 1 & X_2 & Y_2 & X_2^2+Y_2^2 \\ 1 & X_3 & Y_3 & X_3^2+Y_3^2 \end{vmatrix} . \tag{B.1}$$

We will use the method described in [15] to compute a bound B on the worst case error of E (the bound on the worst case error is achieved when we use the maximum size allowed for all the operands). An ε separation is needed when

$$|\widetilde{E}| \leq B\,.$$

where $\widetilde{E}$ is the floating-point approximation of E.

In that case, we will move the query point by ε_x on the x axis and by an ε_y on the y axis. We define $E(\varepsilon_x, \varepsilon_y)$ as

$$E(\varepsilon_x, \varepsilon_y) = \begin{vmatrix} 1 & X_i+\varepsilon_x & Y_i+\varepsilon_y & (X_i+\varepsilon_x)^2+(Y_i+\varepsilon_y)^2 \\ 1 & X_1 & Y_1 & X_1^2+Y_1^2 \\ 1 & X_2 & Y_2 & X_2^2+Y_2^2 \\ 1 & X_3 & Y_3 & X_3^2+Y_3^2 \end{vmatrix} .$$

As in the case of the predicates for arrangements of circles, we wish that $|E(\varepsilon_x, \varepsilon_y)| > 2B$ will hold (instead of $|\widetilde{E(\varepsilon_x, \varepsilon_y)} > B|$, where $\widetilde{E(\varepsilon_x, \varepsilon_y)}$ is the floating-point approximation of $E(\varepsilon_x, \varepsilon_y)$). Expanding $E(\varepsilon_x, \varepsilon_y)$ we can restate the last requirement

$$|E - \varepsilon_x \begin{vmatrix} 1 & Y_1 & X_1^2+Y_1^2 \\ 1 & Y_2 & X_2^2+Y_2^2 \\ 1 & Y_3 & X_3^2+Y_3^2 \end{vmatrix} + \varepsilon_y \begin{vmatrix} 1 & X_1 & X_1^2+Y_1^2 \\ 1 & X_2 & X_2^2+Y_2^2 \\ 1 & X_3 & X_3^2+Y_3^2 \end{vmatrix} - (2X_i\varepsilon_x + \varepsilon_x^2 + 2Y_i\varepsilon_y + \varepsilon_y^2) \begin{vmatrix} 1 & X_1 & Y_1 \\ 1 & X_2 & Y_2 \\ 1 & X_3 & Y_3 \end{vmatrix} | > 2B\,. \tag{B.2}$$

Define $D = \begin{vmatrix} 1 & X_1 & Y_1 \\ 1 & X_2 & Y_2 \\ 1 & X_3 & Y_3 \end{vmatrix}$, $G = \begin{vmatrix} 1 & Y_1 & X_1^2+Y_1^2 \\ 1 & Y_2 & X_2^2+Y_2^2 \\ 1 & Y_3 & X_3^2+Y_3^2 \end{vmatrix}$ and $G' = \begin{vmatrix} 1 & X_1 & X_1^2+Y_1^2 \\ 1 & X_2 & X_2^2+Y_2^2 \\ 1 & X_3 & X_3^2+Y_3^2 \end{vmatrix}$.

After rearranging terms we get

$$|-(\varepsilon_x^2+\varepsilon_y^2)D - \varepsilon_x(G+2X_iD) + \varepsilon_y(G'-2Y_iD) + E| > 2B\,. \tag{B.3}$$

Lemma 9 *Let $\varepsilon = \sqrt{\varepsilon_x^2+\varepsilon_y^2}$. The ε needed so that Inequality B.3 will hold is*

$$\varepsilon > (5BD^{-1})^{\frac{1}{2}}\,. \tag{B.4}$$

Proof. First, we can assume that $D^{-1} > 0$ (if not then we will swap X_1 and Y_1 with X_2 and Y_2). We plug (B.4) into the $\varepsilon_x^2+\varepsilon_y^2$ term of (B.3)

$$|-((5BD^{-1})^{\frac{1}{2}})^2 D - \varepsilon_x(G+2X_iD) + \varepsilon_y(G'-2Y_iD) + E| > 2B\,.$$

We will now prove that the following holds

$$|-5B-\varepsilon_x(G+2X_iD)+\varepsilon_y(G'-2Y_iD)+E|>2B\,.$$

We can take ε_x and ε_y with the appropriate signs such that the following will hold:

$$|-(5B+C_1+C_2)+E|>2B\,. \tag{B.5}$$

where $C_1 \geq 0$ and $C_2 \geq 0$. C_1 is due to the multiplication involving ε_x and C_2 is due to the multiplication involving ε_y. Removing the absolute value from the left-hand side of Inequality B.5, we get:

$$-(5B+C_1+C_2)+E<-2B\,. \tag{B.6}$$

Multiplying both sides of Inequality B.6 by -1, we get:

$$5B+C_1+C_2-E>2B\,. \tag{B.7}$$

Since we assumed that $|E| \leq 2B$, and $C_1 \geq 0$ and $C_2 \geq 0$, we conclude that Inequality B.7 holds, which completes our proof. □

Notice that D is twice the area of the triangle t implied by the points p_1,p_2 and p_3. The fact that ε depends on D^{-1} means that we should set a bound on the minimal area of t. If such a bound is not known in advance, we should certify it using additional predicates. In fact, there is a resemblance to the case of degeneracy of type 3 in the case of arrangements of circles (three circles intersecting in a common point), for which we had to define an additional predicate, which assured that degeneracy of type 4 (the centers of two intersecting circles are too close) does not occur.

Remark. The ε that we found is proved for just one quarter of the disk which δ defines around the original point p (since we need to take the appropriate signs of ε_x and ε_y). That is, there is just one quarter of the disk in which moving the point by an amount of ε will indeed cause the predicate to be evaluated correctly.

Bibliography

[1] P. K. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[2] American National Standards Institute and Institute of Electrical and Electronic Engineers. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard, Std 754-1985,* New York, 1985.

[3] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, and E. Schömer. A computational basis for conic arcs and Boolean operations on conic polygons. In *Proc. ESA 2002*, volume 2461 of *Lecture Notes in Computer Science*, pages 174–186. Springer-Verlag, 2002.

[4] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109(1–2):25–47, 2001.

[5] H. Brönnimann, I. Emiris, V. Pan, and S. Pion. Sign determination in residue number systems. *Theoretical Computer Science*, 210(1):173–197, 1999.

[6] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. In *Proc. ESA 2001*, pages 254–265, 2001.

[7] C. Burnikel, S. Funke, and M. Seel. Exact geometric computation using cascading. *Comput. Geom. Theory Appl.*, 11(3):245–266, 2001.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[9] O. Devillers, A. Fronville, B. Mourrain, and M. Teillaud. Algebraic methods and arithmetic filtering for exact predicates on circle arcs. *Comput. Geom. Theory Appl.*, 22:119–142, 2002.

[10] D. Eberly. Intersection of linear and circular components in 2D. http://www.magic-software.com/, 2000.

[11] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL a computational geometry algorithms library. *Softw. – Pract. Exp.*, 30(11):1167–1202, 2000.

[12] E. Flato. Robust and efficient construction of planar Minkowski sums. Master's thesis, Dept. Comput. Sci., Tel-Aviv Univ., 2000. http://www.cs.tau.ac.il/∼flato.

[13] S. Fortune and V. Milenkovic. Numerical stability of algorithms for line arrangements. In *Proc. 7th ACM Sympos. Comput. Geom.*, pages 334–341, 1991.

[14] S. Fortune and C. J. Van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Trans. Graph.*, 15(3):223–248, 1996.

[15] S. Funke. Exact arithmetic using cascaded computation. Master's thesis, Dept. Comput. Sci., Saarland University, 1997.

[16] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[17] T. Granlund. GNU multiple precision arithmetic library (GMP). http://www.swox.com/gmp/, 2000.

[18] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams. *ACM Trans. Graph.*, 4(2):74–123, 1985.

[19] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[20] L. J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. In *Proc. 5th ACM Sympos. Comput. Geom.*, pages 208–217, 1989.

[21] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.

[22] D. Halperin and E. Packer. Iterated snap rounding. *Comput. Geom. Theory Appl.*, 23:209–225, 2002.

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

[24] J. D. Hobby. Practical segment intersection with finite precision output. *Comput. Geom. Theory Appl.*, 13(4):199–214, 1999.

[25] C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick. Towards implementing robust geometric computations. In *Proc. 4th ACM Sympos. Comput. Geom.*, pages 106–117, 1988.

[26] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A core library for robust numeric and geometric computation. In *Proc. 4th ACM Sympos. Comput. Geom.*, pages 351–359. ACM Press, 1999.

[27] M. Karasick, D. Lieber, and L. R. Nackman. Efficient Delaunay triangulations using rational arithmetic. *ACM Trans. Graph.*, 10(1):71–91, 1991.

[28] K. Melhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[29] V. J. Milenkovic. Shortest path geometric rounding. *Algorithmica*, 1997.

[30] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK, 1992.

[31] E. Packer. Finite precision approximation techniques for planar arrangements of line segments. Master's thesis, Dept. Comput. Sci., Tel-Aviv Univ., 2002.

[32] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *Proc. 15th ACM Sympos. Comput. Geom.*, pages 163–172, 1999.

[33] S. Schirra. Robustness and precision issues in geometric computation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 597–632. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[34] J. Shewchuk. Adaptive robust floating-point arithmetic and fast robust geometric predicates. *Discrete Comput. Geom.*, 18:305–363, 1997.

[35] K. Sugihara. A robust and consistent algorithm for intersecting convex polyhedra. *Comput. Graph. Forum (Proc. EUROGRAPHICS '94)*, 13(3):45–54, 1994.

[36] K. Sugihara. Robust geometric computation based on topological consistency. In V. Alexandrov, J. Dongarra, B. Juliano, R. Renner, and C. K. Tan, editors, *Computational Science - ICCS 2001*, volume 2073 of *Lecture Notes in Computer Science*, pages 12–26. Springer-Verlag, Berlin, Germany, 2001.

[37] K. Sugihara and M. Iri. A robust topology-oriented incremental algorithm for Voronoi diagrams. *Comput. Geom. Theory Appl.*, 4(2):179–228, 1994.

[38] R. Wein. High level filtering for arrangements of conic arcs. In *Proc. ESA 2002*, volume 2461 of *Lecture Notes in Computer Science*, pages 884–895. Springer-Verlag, 2002.

[39] E. Welzl, P. Su, and R. L. Drysdale III. A comparison of sequential delaunay triangulation algorithms. *Comput. Geom. Theory Appl.*, 7:361–385, 1997.

[40] C. K. Yap. Robust geometric computation. In *Handbook of discrete and computational geometry*, pages 653–668. CRC Press, Inc., 1997.

[41] C. K. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997.