Controlled Perturbation for Arrangements of Circles[*]

Dan Halperin Eran Leiserowitz

School of Computer Science
Tel Aviv University
{danha,leiserow}@tau.ac.il

Abstract

Given a collection $\mathcal{C}$ of circles in the plane, we wish to construct the arrangement $\mathcal{A}(\mathcal{C})$ (namely the subdivision of the plane into vertices, edges and faces induced by $\mathcal{C}$) using floating point arithmetic. We present an efficient scheme, controlled perturbation, that perturbs the circles in $\mathcal{C}$ slightly to form a collection $\mathcal{C}'$, so that all the predicates that arise in the construction of $\mathcal{A}(\mathcal{C}')$ are computed accurately and $\mathcal{A}(\mathcal{C}')$ is degeneracy free.

We introduced controlled perturbation several years ago, and already applied it to certain types of arrangements. The major 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 present the scheme, describe how the resolution bound is determined and how it effects the perturbation magnitude. We implemented the perturbation scheme and the construction of the arrangement and we report on experimental results.

Keywords

Arrangements, Robustness, Perturbation

[*]Work reported in this paper has been supported in part by the IST Programme of the EU as Shared-cost RTD (FET Open) Projects under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces) and No IST-2001-39250 (MOVIE - Motion Planning in Virtual Environments), by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.
A preliminary version appeared in Proc. 19th Acm Symp. on Computational Geometry, San Diego, 2003, pp. 264-273.

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.

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.

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 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, 14]. Throughout the years, algorithms for building different kind of arrangements have been proposed. Once an arrangement has been constructed, it can be used to perform many operations, such as *point location*, finding the intersection or union of the 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 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

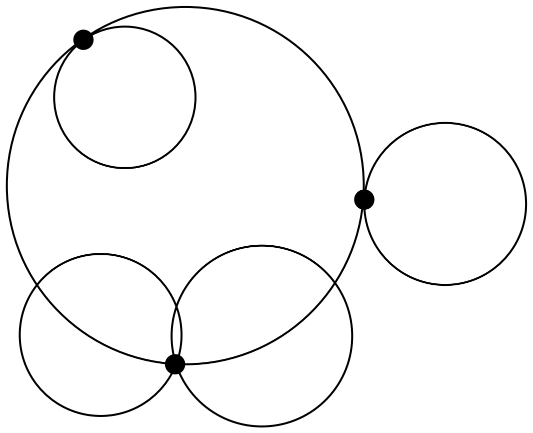

Figure 1: Arrangement of circles with several degeneracies.

be moved by a distance greater than δ. Figure 2 shows a few possible results of our perturbation algorithm, applied to the arrangement shown in Figure 1.

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" [6]). 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" [18]) 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 [16]). 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, 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.

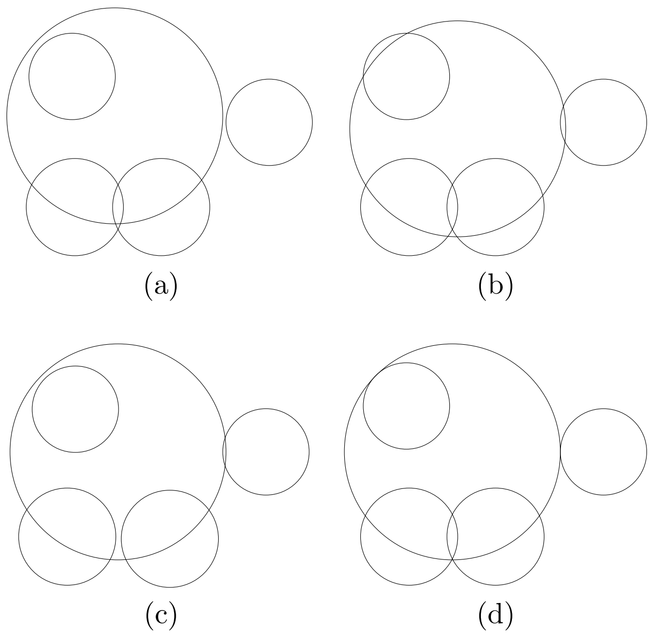

Figure 2: (a) - (c) A few possible results of our perturbation algorithm applied to the arrangement shown in Figure 1. For illustration purposes, the magnitude of the perturbation is rather large. (d) The real result of our perturbation algorithm (as was computed using the software that we developed) — 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.

The perturbation scheme that we follow, controlled perturbation, was first presented in [15] as a method to speed up molecular surface computation. 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 was extended in [23], were it was applied to arrangements of polyhedral surfaces. Those arrangements require complex calculations in order to achieve a good perturbation bound.

In [23] (as in [15]), the *resolution bound* (Section 4) 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, which we anticipate will lead to a better understanding of the method and will open the way to applying the method in other settings.

Related work

Robustness and precision issues have been intensively studied in Computational Geometry in recent years [24], [27].

A prevailing approach to overcoming robustness issues in Computational Geometry is to use exact computation [6], [28]. Such a strategy gives accurate results, and sometimes even allows the input to be degenerate (although degeneracies still need to be handled in the code). When applied naively, exact computation can considerably slow down the performance of a program. One of the possible solutions is to use *filtering* (e.g., [5], [11], [19], [25]). Typically, 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. In [26], high-level filtering is done on arrangements of conic arcs, and an alternative approach is given in [2]. In [8], algebraic methods and arithmetic filtering are used for exact predicates on circle arcs.

An alternative approach aims to compute robustly with limited precision arithmetic, often by approximating or perturbing the geometric objects (e.g., [10], [13], [17], [21]). A variety of methods for handling imprecise geometric computations are surveyed in [24]. Controlled perturbation is a method of this type.

2 Overview of the Scheme

In this section, we give an overview of the controlled perturbation scheme. We describe the main concepts, on which we expand in later sections. 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.

2.1 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 Cartesian 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

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

bound and the perturbation bound. A formal definition of these parameters will be given in the subsequent sections, together with the way we derive them.

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.

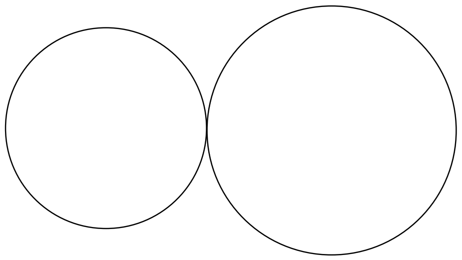
Figure 3: Outer tangency — two circles (bounding interior-disjoint disks) intersect in a single point.

The degeneracies that arise in arrangements of circles have a natural geometric characterization as *incidences*. For example, in *outer tangency* (Figure 3), 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., [15]) 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 [22, 23].

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 ρ 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, $\rho \leq n$ (a formal definition of ρ 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 4). 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 Section 4 where these values 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 Section 5.1.

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.

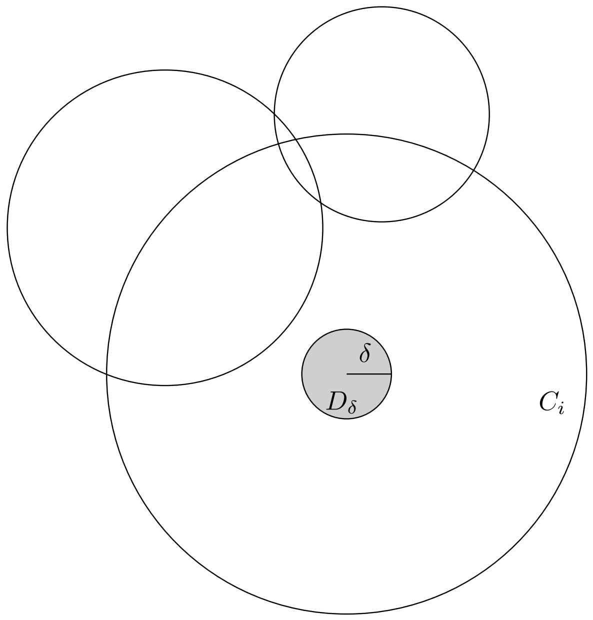

Figure 4: The shaded area is the disk D_δ in which at least half the points will be valid placements for the center of C_i (for clarity, D_δ is shown to be very large).

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 ρ. 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 $\max(0, R_i - \Delta), R_i + \Delta$. We define ρ as the maximum number of such annuli intersecting a single annulus (Figure 5). Notice that in the worst case $\rho = n - 1$.

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 and 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 $M - \Delta$.

Exact vs. Approximate Intersection Point

Intersection points play an important role in our scheme. We will deal with circle-circle intersection points and line-circle intersection points. Sometimes, we will refer to the *exact*

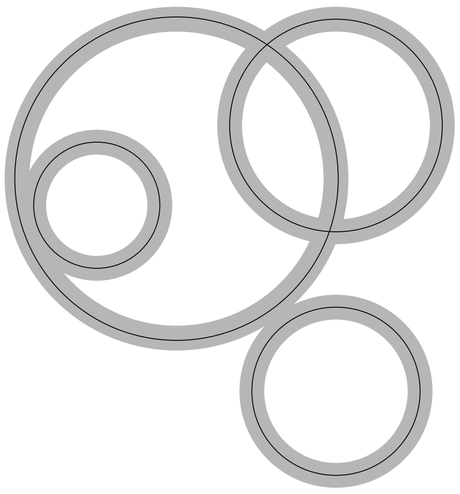
Figure 5: Each $C_i \in \mathcal{C}$ induces an annulus (the shaded area) bounded by two circles, centered at the center of C_i, with radii $\max(0, R_i - \Delta), R_i + \Delta$. We define ρ as the maximum number of such annuli intersecting a single annulus, in this example $\rho = 3$.

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 Section 4) 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 6).

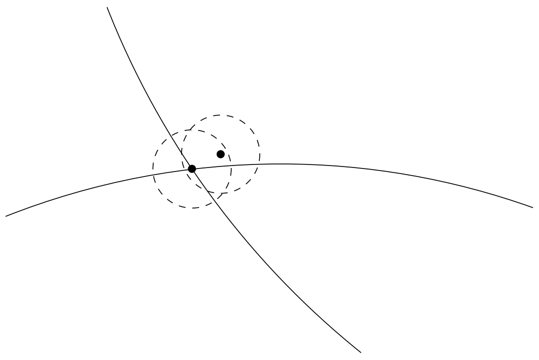
Figure 6: We can inflate a disk of radius ω around the exact (resp. approximate) intersection point that would contain the corresponding approximate (resp. exact) intersection point. These disks are illustrated as dashed circles.

2.2 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 $\varepsilon$, $\delta$ 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 5.1. 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 (Section 3) and ε (Section 4).

3 Computing the Perturbation Bound δ

In this section 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 ρ.

The resolution bound ε is the distance that we need to move the center of a circle in order to avoid a *potential degeneracy.* In the next section we will show how to compute a good bound on the resolution parameter. In this section we show how to determine δ assuming that $\varepsilon > 0$ is given.

3.1 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 section). 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).

3.2 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 7 (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 7 (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 through, or very near to a point in P_{jk} is an annulus (Figure 7 (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 7 (d)). Its area is $\pi\varepsilon^2$.

3.3 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 ρ denote the density parameter of $\mathcal{C}$. There are at most ρ circles defining the regions of

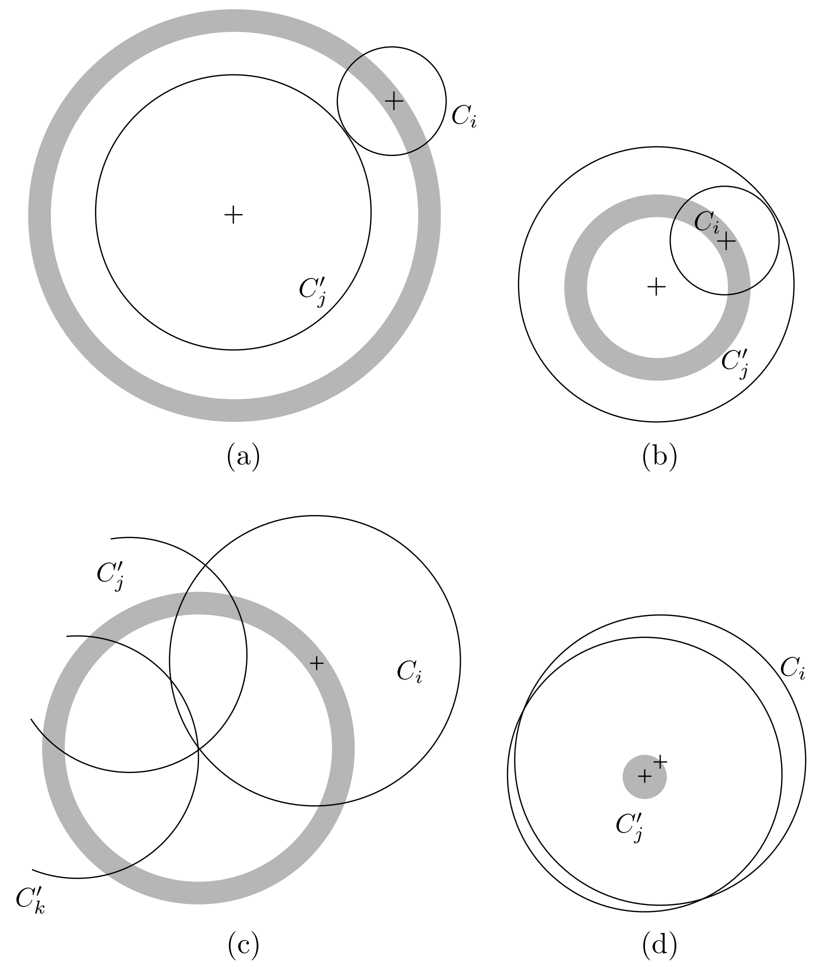

Figure 7: The shaded areas in (a),(b),(c),(d) are the portions of the forbidden regions F_1, F_2, F_3 and F_4 respectively for C_i and C'_j (and also C'_k for F_3).

F_1 and F_2, there are at most $\binom{\rho}{2} \times 2$ points defining the region F_3, and at most ρ points defining the region F_4. Therefore the following bounds on the areas can be obtained.

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

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

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

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

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

If C_i should be perturbed, then δ will define a disk D_δ in which its center can be moved (Figure 4). 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

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

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

$$\delta > \sqrt{2\rho\varepsilon(12R + 4\rho R + \varepsilon)}\,. \tag{1}$$

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 (Section 6). 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 (Section 5). 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.

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

4 Defining the Predicates and Determining a Worst Case ε

In this section we examine the four possible degeneracies that can arise in an arrangement of circles (Section 3.1). Given the precision of the underlying arithmetic, we can find the ε sufficient 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 Section 3. Denote by ε_i the resolution parameter needed to compute the forbidden region F_i.

4.1 Preliminaries

To examine the error induced by a floating-point computation we will use the notation suggested in [5, 12]. 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 recursively according to Table 1 (taken from [12]). Intuitively, $\widetilde{E_{sup}}$ reflects errors resulting from the *operands*, and ind_E reflects errors resulting 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.

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

Table 1: The computation of $\widetilde{E_{sup}}$ and ind_E [12]. In the first row we assume that A is a floating-point number.

4.2 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:

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

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 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. 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 an upper bound on the absolute value of $X_1, X_2, Y_1, Y_2, R_1, R_2$. 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)

$$\varepsilon_1 > \sqrt{10 \odot 2^{-p} \odot 12 \odot M^2}\,. \tag{3}$$

Following [12] 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 3, 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,

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

$$|- 2(R_1 + R_2)\varepsilon + \varepsilon^2| > |\varepsilon^2| \tag{4}$$

If $(R_1 + R_2) > 2\varepsilon$, then the left-hand side of Inequality 4 only increases. Thus, we conclude that Inequality 3 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 subsection 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.*

4.3 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:

$$E = (X_1 - X_2)^2 + (Y_1 - Y_2)^2 - (R_2 - R_1)^2 \,. \tag{5}$$

Following similar arguments to those in the case of outer tangency, we conclude that Inequality 3 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 subsection 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 $R_2 - R_1 > 0$.

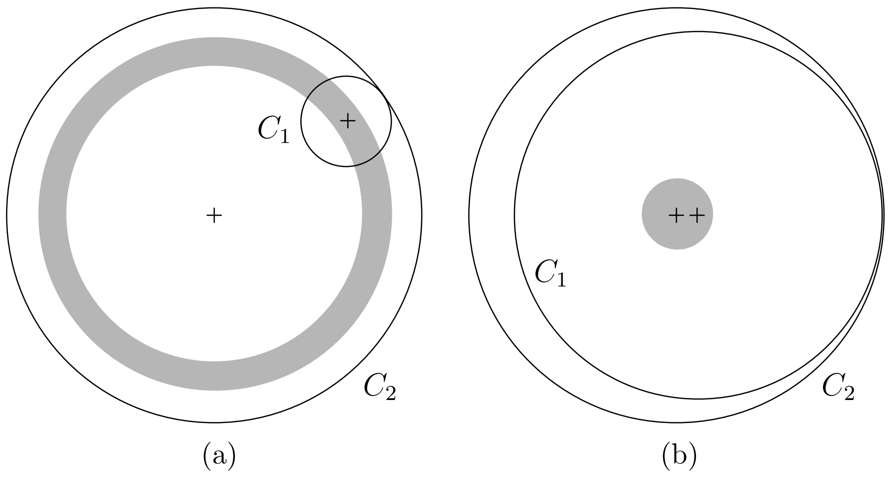

Figure 8: The shaded area is a part of F_2, the forbidden region. (a) The case where $R_2 - R_1 > \varepsilon$. Notice that there are valid placements for the center of C_1 such that C_1 is completely inside C_2. (b) The case where $0 < R_2 - R_1 \leq \varepsilon$. Notice that there is *no* valid placement for the center of C_1 such that C_1 is completely inside C_2.

In the case of the *plus* sign, similar arguments to those of the previous subsection 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 3 is indeed valid. However, if $0 < R_2 - R_1 \leq \varepsilon$, then we cannot certify that Inequality 3 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 8). 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 3 gives a valid ε also for F_2. Notice, that Inequality 3 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.

4.4 Three Circles Intersecting In a Common Point

In this subsection 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 attempt to give a good resolution bound for this type of degeneracy, was to continue with the same approach as in the previous subsections (based on [12]). However, since this is a more complicated situation, the bound that was achieved was very large.

We will compute the intersection points, and η — 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 η. We then make sure that none of the disks overlap.[2]

To compute the intersection point of two circles C_1 and C_2, we use the following formulation [9].

$$s = \frac{1}{2}\,\frac{{R_1}^2 - {R_2}^2}{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} + \frac{1}{2} \tag{6}$$

$$t = \left[\frac{{R_1}^2}{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} - s^2\right]^{\frac{1}{2}}\ . \tag{7}$$

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

$$\begin{aligned}[x, y] \;=\;& [X_1, Y_1] + s[X_2 - X_1, Y_2 - Y_1] \\ & \pm t[Y_2 - Y_1, X_1 - X_2]\,. \end{aligned} \tag{8}$$

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 the worst-case error for such expressions. Finally, we will convert Eq. 8, 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 [3]. 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 $+\infty$.

[2]Notice that throughout this section, we are only concerned with pairs of intersection points originating from three different circles.

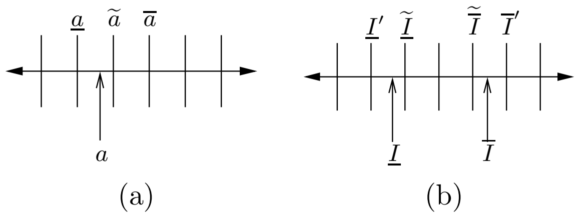

Figure 9: (a) Let a be a real number or the result of an expression involving a single operation on one or two floating-point operands, and let $\widetilde{a}$ be the nearest floating-point number to a. Taking the next representable numbers after $\widetilde{a}$ in both directions as end-points of the 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 $[\underline{I}', \overline{I}']$.

Getting the next representable floating-point number can be done using the function `nextafter()`, which is recommended by the IEEE standard 754, and is available for most platforms. Thus, the interval $[\underline{E_1}, \overline{E_1}]$ contains $\widetilde{E_1}$ (Figure 9 (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 9 (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 (whose proofs we omit here; the proofs are simple and can be found in [20]) 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, $[\underline{E_m}, \overline{E_m}]$.*

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.*

To get a bound on the worst-case error of Eq. 8, 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 1). Also, we will only use the absolute value of the operands (so Lemma 3 would hold).

Yet, in Eq. 8 there are also division operations. The term in the denominators of Eq. 8 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 (Subsection 4.5). 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. 8, 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. 6 and Eq. 7 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. 8 according to the method described above (i.e., regard Eq. 8 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 η denote the bound on the worst-case error for Eq. 8, computed using the method described above and multiplied by $\sqrt{2}$. η 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 η (Figure 10) which contains the *exact* intersection point (recall that the bound that was computed for Eq. 8 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.[3] In other words, two approximate intersection points P_1 and P_2 should be at least 2η apart. Still, in order to be able to apply the efficient perturbation algorithm (as remarked on page 25), 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 6η 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

$$E = (X_{P_2} - X_{P_1})^2 + (Y_{P_2} - Y_{P_1})^2 - (6\eta)^2 , \tag{9}$$

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

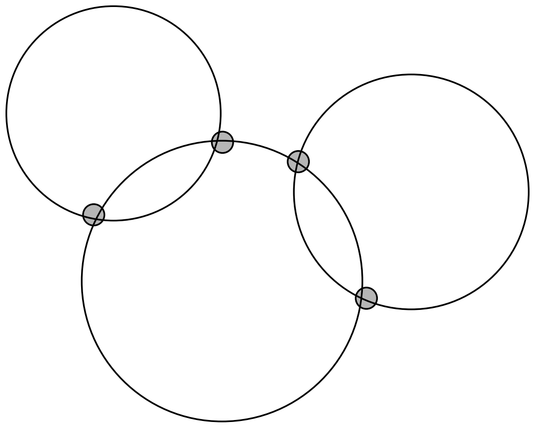

Figure 10: Around each approximate intersection point we inflate a disk of radius η that contains the exact intersection point.

where P_1 and P_2 are intersection points, computed by Eq. 8. As before, since we are using floating-point arithmetic we compute,

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

and according to Table 1, we have:

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

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

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

After squaring both sides, and rearranging terms we get,

$$(X_{P_2} - X_{P_1})^2 + (Y_{P_2} - Y_{P_1})^2 - (6\eta)^2 = 12\eta\alpha + \alpha^2 \,. \tag{10}$$

We notice that the left-hand side of Eq. 10 is exactly E, so we can rewrite our requirement, this time in terms of the right-hand side of Eq. 10 (with the added distance α), that is

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

We can extract a bound on α,

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

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,

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

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

$$\varepsilon_3 > 6 \odot \eta \oplus \alpha = 6 \odot \eta \oplus \sqrt{(10 \odot 2^{-p}(32M^2 \oplus 36\eta^2))} \ . \tag{11}$$

Remark. We use UP rounding mode for all the operations except in the computation of B (recall that according to [12], we compute B in Round To Nearest mode).

4.5 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 (we can check if two circles are intersecting by using the outer and inner tangency tests), is at least ξ, where ξ is a positive floating-point number. Using the same method that we applied for degeneracies of type 1 and 2, we have computed the worst case ε_4 needed to estimate F_4 (we omit the details here).

$$\varepsilon_4 > \xi \oplus \sqrt{(14 \odot 2^{-p} \odot (8 \cdot M^2 \oplus \xi^2))} \ . \tag{12}$$

4.6 Numerical Example

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

- $\eta \leq 0.009$
- $\varepsilon_{1,2} = 0.00016323404237781946$
- $\varepsilon_3 = 0.05426656007499713885$
- $\varepsilon_4 = 0.03162279436525219228$
- $\Rightarrow \varepsilon = 0.05426656007499713885$.

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 (we found the distances above, by experimenting with different values).
(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.

5 Algorithmic Details

5.1 Efficient Perturbation Algorithm

In order to achieve a good running time, we use two type of data structures: a kd-tree [7] 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).

Testing a circle C_i for degeneracies of type 1,2 and 4 can take $\Theta(n)$ or may be $\Omega(n)$. If done in a naive fashion, testing a circle C_i for degeneracy of type 3 can take $\Theta(n^2)$ 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 11). 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 $\{P^j_k, k = 1, \ldots, s | X_j - \frac{R_j}{\sqrt{2}} \leq X_{P^j_k} \leq X_j + \frac{R_j}{\sqrt{2}}, Y_{P^j_k} > Y_j\}$, and use their x coordinate as the key for the binary tree. Analogously, we construct the *lower* binary tree. We construct the *left* binary tree T_{left} of C'_j, such that it will hold all the points $\{P^j_k, k = 1, \ldots, s | Y_j - \frac{R_j}{\sqrt{2}} < Y_{P^j_k} < Y_j + \frac{R_j}{\sqrt{2}}, X_{P^j_k} < X_j\}$, and use their y coordinate as the key for the binary tree. Analogously, we construct the *right* binary tree.

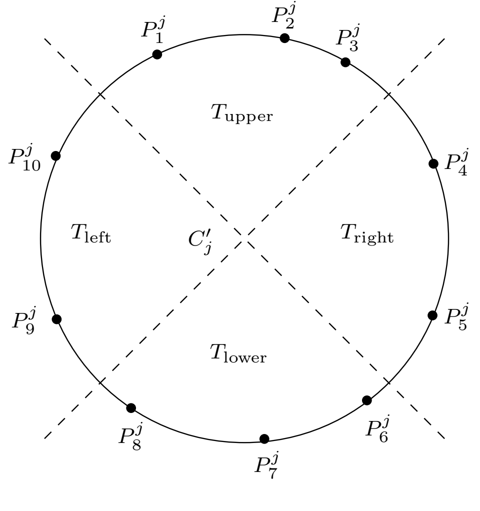

Figure 11: The four binary trees associated with a circle C'_j.

When we come to add the new circle C'_i, we check with 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. 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 11, 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 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). Finding a valid location for the center of C_i requires two attempts on average. There are n circles to be added, and for each circle $O(n \log n)$ time is needed for its insertion (including the update of the relevant binary trees), thus the algorithm runs in overall expected $O(n^2 \log n)$ time. The proof of correctness of the usage of these trees is given in [20].

Remark. In Subsection 4.4 we required that the distance between two approximate intersection points should be at least 6η. This distance allows us to safely decide the order of the *exact* intersection points along the x and y axis. Again, the details are given in [20].

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 an 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). See [7, Chapter 2] for more details on the DCEL structure.

5.2 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 [7], 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.

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 C'_i of $\mathcal{C}'$ (Figure 12). 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 $\Theta(n)$ time (recall that n is the number of circles in $\mathcal{C}'$).

This point location strategy is easy to implement, yet in [20], we point out the robustness-related issues that arise in the implementation of this strategy.

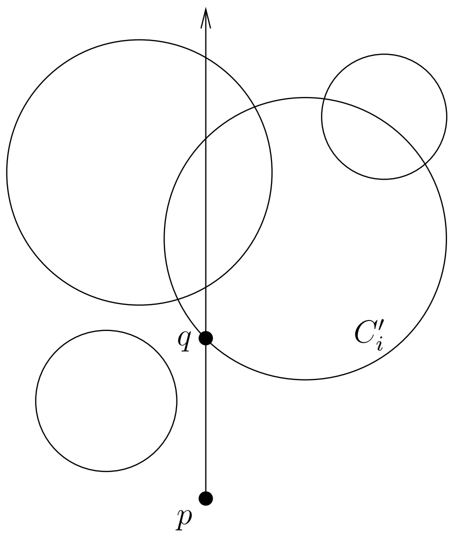

Figure 12: We shoot a vertical ray from p, and find the closest intersection point q.

6 Experimental Results

In this section 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 [7, Chapter 2] for details on this data structure) with a simple point-location mechanism. 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 Subsection 3.2, the bound on δ that we computed in Section 3 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 Section 3 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 13 (a)),
- `flower`, a "flower" composed of 40 circles, all intersecting in a common point (Figure 13 (b)),
- `rand_sparse`, a collection of 40 random circles (Figure 14 (a)),
- `rand_100`, a collection of 100 random circles (Figure 14 (b)),
- `rand_1000`, a collection of 1000 random circles (Figure 15 (a)),
- `rand_2000`, a collection of 2000 random circles (Figure 15 (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 2. The results of the perturbation and running times for those inputs are give in Table 3 (all the given results are from averaging the results of 5 tests for each input), with the IEEE double number type and the bound ε computed using $M - \Delta = 1000$ 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 4 shows the number of near degeneracies that were handled for each input (in a single run of the algorithm). Table 5 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".

Remarks. (1) The fifth column of Table 4 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 is not always the case, as is shown in the simple example, whose data is given in Table 6. However, it appears that in many cases, resolving degeneracy of type 2 (inner tangency) also eliminates degeneracy of type 4.

(2) In order to evaluate the quality of our method, it is interesting to examine how many bits would an exact number type require for the construction of the example arrangements that we have tested. We have carried out this computation for number types that rely on the separation bound theory (e.g., LEDA's "real" [6] or CORE's "Expr" [18]). For LEDA's "real" as described in [4], we have found the separation bound to require 901 bits (notice however, that using such a separation bound will allow us to compute with far greater resolution). That is, the exact number type may require as many as 901 bits, as opposed to the 53 bits of precision that are used by the standard "double" number type.

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

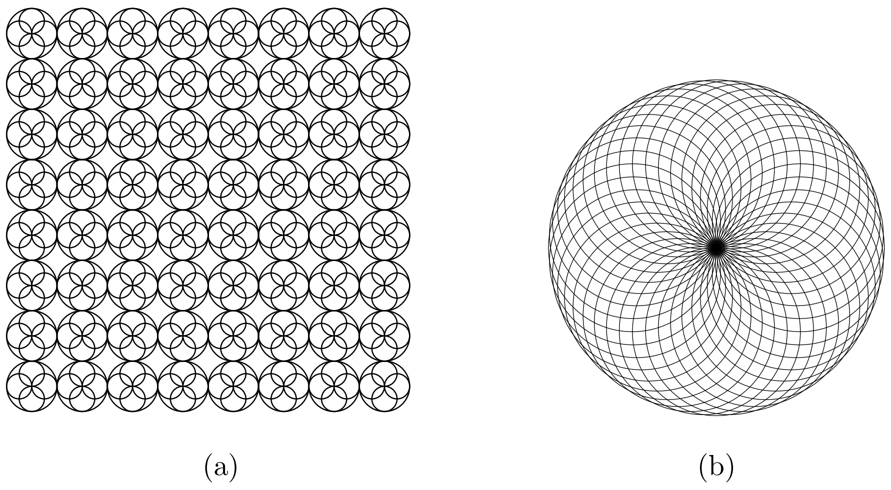

Figure 13: (a) A grid of 320 circles, which involves many inner and outer tangencies. (b) A "flower" composed of 40 circles, all intersecting in a common point (the origin).

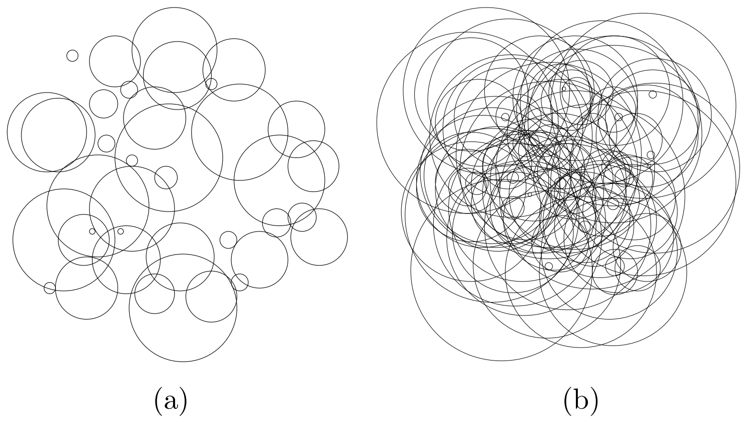

Figure 14: (a) A collection of 40 random circles. (b) A collection of 100 random circles.

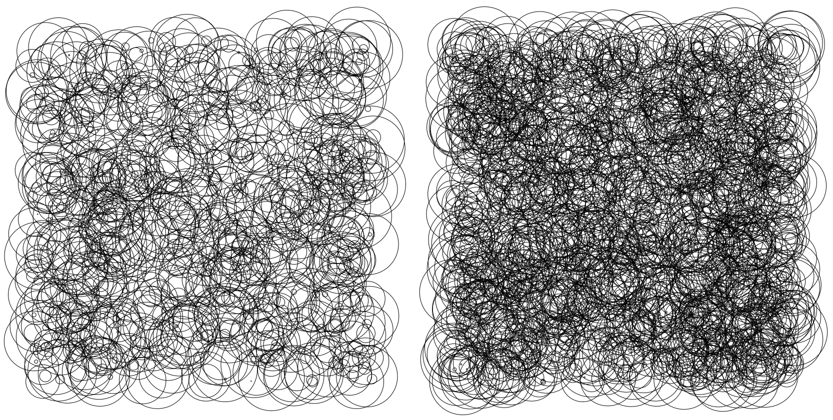
(a) (b)

Figure 15: (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 3: 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 4: 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 5: The properties of the arrangements that were computed for each input (in a single run of the algorithm).

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

Table 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 $\xi = 0.03$.

References

[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] 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.

[3] 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.

[4] 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.

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

[6] C. Burnikel, J. Könemann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig. Exact geometric computation in LEDA. In *Proc. 11th ACM Sympos. Comput. Geom.*, pages 418–419, 1995.

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

[8] 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.

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

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

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

[20] E. Leiserowitz. Controlled perturbation for arrangements of circles. Master's thesis, Dept. Comput. Sci., Tel-Aviv Univ., 2003.

[21] V. J. Milenkovic. Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artif. Intell.*, 37:377–401, 1988.

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

[23] 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.

[24] 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.

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

[26] 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.

[27] C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*, pages 653–668. CRC Press, Inc., 1997.

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