Controlled Perturbation for Delaunay Triangulations

Stefan Funke[*] Christian Klein[*] Kurt Mehlhorn[*] Susanne Schmitt[*]

18. Oktober 2004

Keywords: *Randomized incremental algorithm, randomized incremental construction, controlled perturbation, floating point computation, Delaunay triangulations, convex hulls*

Zusammenfassung

Most geometric algorithms are idealistic in the sense that they are designed for the Real-RAM model of computation and for inputs in general position. Real inputs may be degenerate and floating point arithmetic is only an approximation of real arithmetic. Perturbation replaces an input by a nearby input which is (hopefully) in general position and on which the algorithm can be run with floating point arithmetic. Controlled perturbation as proposed by Halperin et al. calls for more: control over the amount of perturbation needed for a given precision of the floating point system. Or conversely, a control over the precision needed for a given amount of perturbation. Halperin et al. gave controlled perturbation schemes for arrangements of polyhedral surfaces, spheres, and circles. We extend their work and point out that controlled perturbation is a general scheme for converting idealistic algorithms into algorithms which can be executed with floating point arithmetic. We also show how to use controlled perturbation in the context of randomized geometric algorithms without deteriorating the running time. Finally, we give concrete schemes for planar Delaunay triangulations and convex hulls and Delaunay triangulations in arbitrary dimensions. We analyze the relation between the perturbation amount and the precision of the floating point system. We also report about experiments with a planar Delaunay diagram algorithm.

1 Introduction

Most algorithms of computational geometry are designed under two simplifying assumptions: the availability of a Real-RAM and non-degeneracy of the input. A Real-RAM computes with real numbers in the sense of mathematics, i.e., it stores real numbers in its registers and performs exact arithmetic (basic arithmetic, roots of polynomials, trigonometric functions, ...) on them. The exact notion of degeneracy depends on the problem; examples are collinear or cocircular points or three lines with a common point. We call an algorithm designed under the two simplifying assumptions an *idealistic algorithm*. Implementations have to deal with

[*]Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, {funke,cklein,mehlhorn,sschmitt}@mpi-sb.mpg.de

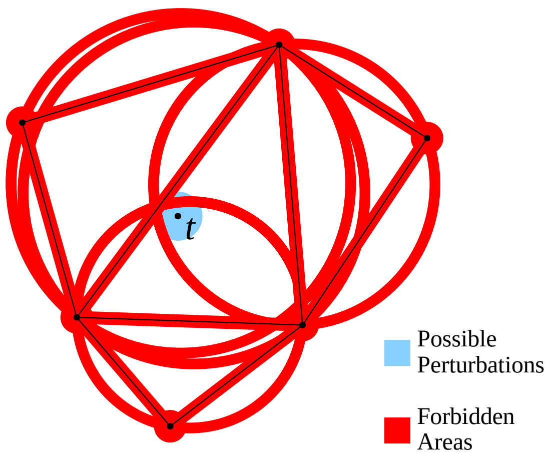

Abbildung 1: The forbidden areas (light gray) for a new point t induced by a Delaunay triangulation. Each forbidden area is induced by a vertex (a disc centered at it), an edge (a strip around it) or a triangle (an annulus around its circumcircle) of the current diagram.

the precision problem (= the Real-RAM assumption) and the degeneracy problem (= the non-degeneracy assumption).

The *exact computation paradigm* [KLN91, JRZ91, FvW93, Yap93, MN94, MN99] addresses the precision problem. It proposes to implement a Real-RAM tuned to geometric computations. The degeneracy problem is addressed by reformulating the algorithms so that they can handle all inputs. This may require non-trivial changes. The approach is followed in systems like LEDA [LED] and CGAL [CGA].

Halperin et al. [HS98, HR, HL03] proposed controlled perturbation to overcome both problems. The idea is to solve the problem at hand not on the input given but on a nearby input. The perturbed input is carefully chosen, hence the name *controlled perturbation*, so that it is non-degenerate and can be handled with approximate arithmetic. They applied the idea to three problems (computing polyhedral arrangements, spherical arrangements, or arrangements of circles) and showed that variants of the idealistic algorithms can be made to work. In this paper we extend their work in several directions:

1. we point out that controlled perturbation and guarded

tests are a general conversion strategy[1] for idealistic algorithms, see Section 2,

2. we show how to use controlled perturbation in the context of randomized algorithms, in particular randomized incremental constructions, see Section 3,
3. we give specific schemes for planar Delaunay triangulations and convex hulls and Delaunay triangulations in arbitrary dimensions, see Sections 4, 5, and 6, and
4. we show how to generalize the analysis from the previous item to all surprise-free randomized incremental constructions, see Section 7.

Note: We want to emphasize that while at first sight the concept of controlled perturbation might look similar to the approach of smoothed analysis as pioneered by Spielman/Teng [ST04], the goals and effects are quite different. The purpose of controlled perturbation is to actually perturb the input, thereby reducing the required precision of the underlying arithmetic and getting rid of explicit treatment of degenerate cases. In contrast, the goal of [ST04] is more analytical in a sense that they examine the combinatorial complexity of an algorithm when moving from a problem instance randomly to another 'nearby' instance. That should be seen rather as a trade-off between worst-case and average-case analysis than something to actually implement.

2 Guarded Tests and Controlled Perturbation

Geometric algorithms branch on geometric predicates. A basic predicate for two-dimensional geometry is orientation. Given three points decide whether they lie on a common line or form a left turn or form a right turn. Typically, geometric predicates can be expressed as the sign of an arithmetic formula E. For example, the *orientation predicate* for $d+1$ points $(p_0,\dots,p_d)$ in $\mathbb{R}^d$ is given by the sign of a $(d+1)\times(d+1)$ determinant:

$$(2.1)\quad orient(p_0,\dots,p_d) := \operatorname{sign}\begin{vmatrix} p_{01} & \dots & p_{0d} & 1 \\ \vdots & \dots & \vdots & \vdots \\ p_{d1} & \dots & p_{dd} & 1 \end{vmatrix}.$$

The determinant evaluates to zero if and only if the $d+1$ points lie in a common hyperplane. In many algorithms this is considered a degeneracy.

When evaluating an arithmetic formula E using floating-point arithmetic, round-off errors occur which might result in the wrong sign being reported. If this stays undetected, the program may enter an illegal state and disasters might happen, see [KMP+] for some instructive examples. In order to guard against round-off errors, we postulate the availability of a predicate $\mathcal{G}_E$ with the following *guard property: If $\mathcal{G}_E$ evaluates to true when evaluated with floating point arithmetic, the evaluation of E with floating point arithmetic yields the correct sign.* In an idealistic algorithm A we now guard every sign test by first testing whether the corresponding guard evaluates to true. If not, we abort. We call the resulting algorithm a guarded algorithm and use A_g to denote it. On an input x, A_g will either follow the same execution path as A or abort after an initial segment of it. In the former case, we will say that A_g succeeds on x. When A_g succeeds on x, the combinatorial part of the output will be correct and the numerical part will be a floating point approximation of the exact result. In all applications in this paper, the numerical part of the output will be identical to the input. Also the running time of A_g on x will be at most the running time of A on input x; this assumes that the cost of evaluating a guard is bounded by the cost of evaluating the corresponding expression and ignores constant factors.

The controlled perturbation version of idealistic algorithm A is as follows: Let δ be a positive real. On input x, we first choose a δ-perturbation $\tilde{x}$ of x and then run the guarded algorithm A_g on $\tilde{x}$. If it succeeds, fine. If not, repeat. What is a δ-perturbation? If the input is a set of points, the following definition is natural. A δ-perturbation of a point is a random point in the δ-ball (or δ-cube) centered at the point and for a set of points a δ-perturbation is simply a δ-perturbation of each point in the set. For more complex objects, alternative definitions come to mind, e.g., for a a circle one may want to perturb the center or the center and the radius.

The goal is now to show experimentally and/or theoretically that A_g has a good chance of working on a δ-perturbation of each input and a small value of δ. More generally, one wants to derive a relation between the precision p of the floating point system (= length of the mantissa), a characteristic of the input set, e.g., the number of points in the set and an upper bound on the maximal coordinate of any point in the input, and δ. Halperin et al. have done so for arrangements of polyhedral surfaces, arrangements of spheres, and arrangements of circles.

We want to stress that a guarded algorithm can be used without any analysis. Suppose we want to use it with a certain δ. We execute it with a certain precision p. If it does not succeed, we double p and repeat. We elaborate on this scheme at the end of Section 4.

Guard predicates must be safe and should be effective, i.e., if a guard does not fire, the approximate sign computation must be correct, and guards should not fire too often unnecessarily. It is usually difficult to analyze the floating point evaluation of $\mathcal{G}_E$ directly. For the purpose of the analysis, we therefore postulate the existence of a *bound predicate* $\mathcal{B}_E$ with the property: *If $\mathcal{B}_E$ holds, $\mathcal{G}_E$ evaluates to true when*

[1] This observation is already implicit in the paper of Halperin and Leiserowitz [HL03]. They write in Section 2.1: We look to move the centers of the circles slightly ... such that when constructing the arrangement (of the perturbed circles) while using a fixed precision floating point filter, the filter will always succeed and we will never have to resort to higher precision or exact arithmetic.

E	$\widetilde{E}$	$\widetilde{E_{\text{sup}}}$	ind_E
$c = const$	c	$\lvert c\rvert$	0
$x+y$	$x \oplus y$	$\widetilde{x_{\text{sup}}} \oplus \widetilde{y_{\text{sup}}}$	$1+\max(\text{ind}_x, \text{ind}_y)$
$x-y$	$x \ominus y$	$\widetilde{x_{\text{sup}}} \oplus \widetilde{y_{\text{sup}}}$	$1+\max(\text{ind}_x, \text{ind}_y)$
$x \cdot y$	$x \odot y$	$\widetilde{x_{\text{sup}}} \odot \widetilde{y_{\text{sup}}}$	$1+\text{ind}_x+\text{ind}_y$

Tabelle 1: Rules for calculating error bounds. $\oplus$, $\ominus$, and $\odot$ stand for floating point addition, subtraction, and multiplication, respectively. .

evaluated with floating point arithmetic. We next give concrete examples for guard and bound predicates:

Straight-Line Evaluation: When E is evaluated by a straight-line program, it is easy to come up with suitable predicates $\mathcal{G}_E$ and $\mathcal{B}_E$ using forward error analysis. For example, the rules in Table 1 ([MN99]) recursively define two quantities $\widetilde{E_{\text{sup}}}$ and ind_E for every arithmetic expression E such that

$$|E - \widetilde{E}| \leq B_E := \widetilde{E_{\text{sup}}} \cdot \text{ind}_E \cdot 2^{-p}$$

where $\widetilde{E}$ denote the value of E computed with floating point arithmetic and p denotes the mantissa length of the floating-point system. (i.e. $p = 52$ for IEEE doubles). We also write ε instead of 2^{-p}. We can then use

$$\text{(2.2)} \qquad \mathcal{G}_E \equiv \left(|\widetilde{E}| > B_E\right) \quad \text{and} \quad \mathcal{B}_E \equiv (|E| > 2B_E)\,,$$

where $\mathcal{B}_E$ is valid since it guarantees that $|\widetilde{E}| = |E| - |E - \widetilde{E}| > 2B_E - B_E = B_E$ by the inverse triangle inequality. Orientation tests and insphere tests in low dimensions, certainly dimensions 2 and 3, are usually implemented through expressions and hence straight-line computation.

General Evaluation: For a more complex expression E, we will evaluate the sign with a program involving branching. For example, we might compute the sign of the determinant of a $d \times d$ matrix A by computing an LU-decomposition of the matrix and then determining the signs of the determinants of L and U (which is simply the parity of the number of negative elements on the diagonal). Gaussian elimination [DH91, Section 2.4.2] yields matrices L' and U' such that every entry of $\Delta = L'U' - A$ is bounded in absolute value by $f(d)M\varepsilon$, where M is the maximal absolute value of an entry of A and $f(d)$ depends on the pivoting strategy. For example, $f(d) = d2^d$ for partial pivoting.

Let $A' = L'U'$. Then $A' = A + \Delta = A(I + A^{-1}\Delta)$. We have $\det(I - A^{-1}\Delta) = (1-\gamma_1)\cdots(1-\gamma_d)$ where the γ_i are the eigenvalues of $A^{-1}\Delta$. Assume $|\gamma_i| \leq 1/(100d)$ for all i. Then $|1 - \prod_i(1-\gamma_i)| \leq \sum_i |\gamma_i| \leq 1/100$ and hence $\det A' = \det A(1+\delta)$ with $|\delta| < 1/100$. Let $\lambda_1,\ldots,\lambda_n$ be the eigenvalues of A. Then $\det A = \lambda_1 \cdots \lambda_d$ and $|\lambda_i| \leq M$ for all i since every entry of A is bounded by M. Thus $|\lambda_i| \geq |\det A|/M^{d-1}$ for all eigenvalues of A and hence all eigenvalues of A^{-1} are bounded by $M^{d-1}/|\det A|$ in absolute value. Thus the largest eigenvalue of $A^{-1}\Delta$ is bounded by $f(d)M^d\varepsilon/|\det A|$ in absolute value. This is less than $1/(100d)$ if $|\det A| > 100d\,f(d)M^d\varepsilon$. We may thus use

$$\mathcal{G}_d \equiv \left(|\det L'U'| > 1.01 \cdot 100d\,f(d)M^d\varepsilon\right)$$
$$\mathcal{B}_d \equiv \left(|\det A| > B_d := 1.01^2 \cdot 100d\,f(d)M^d\varepsilon\right)\,.$$

Clarkson [Cla92] gave a method for computing the exact sign of a $d \times d$ integer matrix with entries bounded by M with floating point arithmetic of precision $1.5\log d + 2\log M$. It is not clear to us how to exploit his method in the context of controlled perturbation.

3 Randomized Algorithms and Controlled Perturbation

Randomized algorithms are abundant in computational geometry; we are particularly interested in randomized incremental constructions. We use A to denote our randomized algorithm and assume that it uses at most $m = f(n)$ random bits on any input of length n. We use x to denote the input and $\pi \in \{0,1\}^m$ to denote the random bits used by the algorithm. We also use the following notation:

$T(x,\pi)$ denotes the running time of A on input x and random bits π.

$T(x) = \mathrm{E}_\pi[T(x,\pi)] = 2^{-m}\sum_\pi T(x,\pi)$ denotes the expected running time of A on x. The expectation is taken with respect to the random bits.

$U_\delta(x)$ denotes the δ-neighborhood of x. We use x' to denote a random element in $U_\delta(x)$.

$T_\delta(x) = \mathrm{E}_{x' \in U_\delta(x)}[T(x')] = \frac{1}{|U_\delta(x)|}\sum_{x' \in U_\delta(x)} T(x')$ denotes the δ-smoothed running time at x, i.e., the average expected running time on a random instance in a δ-neighborhood of x. For simplicity, we use summation instead of integration for the averaging over $U_\delta(x)$. The δ-smoothed running time at x may be larger or smaller than the running time at x.

The guarded version A_g of A satisfies: For every input x and random bits π, the execution of $A_g(x,\pi)$ is a prefix of the execution of $A(x,\pi)$ and $T_g(x,\pi) = O(T(x,\pi))$ for all x and π, where $T_g(x,\pi)$ is the running time of A_g on input x with random bits π. Let $\chi(x,\pi)$ be the indicator variable which is 1 if $A_g(x,\pi)$ aborts and let

$$p_\delta(x) = \sum_\pi \sum_{x' \in U_\delta(x)} \frac{\chi(x',\pi)}{2^m \cdot |U_\delta(x)|}$$

be the probability that A_g fails on a random δ-perturbation x' of x and random bits π, (figure 2). The controlled perturbation algorithm CP is as follows:

```
repeat
  choose a random δ-perturbation $x'$ and random bits $\pi$;
until $A_g(x',\pi)$ succeeds.
```

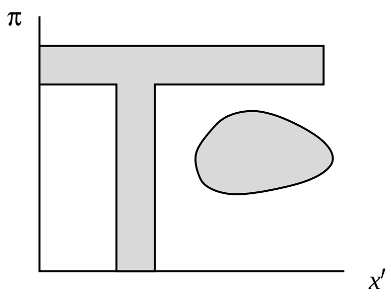

Abbildung 2: The x-axis spans the instances in the δ-neighborhood of x and the y-axis spans the space of random bits. The shaded area indicates the pairs (x',π) for which A_g aborts. Observe that there may be instances for which no random bits work and and that there may be random bits which work for no instance.

THEOREM 3.1. *The expected running time of the controlled perturbation scheme on input x is bounded by* $\frac{1}{1-p_\delta(x)}$ *times the smoothed complexity of A on input x, i.e.,*

$$\mathrm{E}[T_{CP}(x)] \le \frac{1}{1-p_\delta(x)} T_\delta(x) \ .$$

Proof: *CP* runs A_g on a random δ-perturbation x' of x with random bits π. This has cost bounded by $T(x',\pi)$. If the computation succeeds, we are done. Otherwise, we repeat. The probability of failing is $p_\delta(x)$. Thus

$$T_{CP}(x) \le \sum_{\pi} \sum_{x' \in U_\delta(x)} \frac{T(x',\pi) + \chi(x',\pi) T_{CP}(x)}{2^m \cdot |U_\delta(x)|}$$

and hence $T_{CP}(x) \le T_\delta(x) + p_\delta(x) \cdot T_{CP}(x)$. □

In the case of randomized incremental constructions the input x is a set S of n objects and the random bits determine a permutation of S. The objects in S are considered in the order of this permutation. The execution of the algorithm naturally splits into phases, one phase per point. If the probability of failure in any particular phase is less than c/n, the total probability of failure is less than c.

4 Planar Delaunay Triangulations

A triangulation of a point set S is called *Delaunay triangulation* ($\mathcal{DT}(S)$) if the interior of the circumcircle of any triangle in the triangulation contains no point of S. If S contains no four cocircular points, $\mathcal{DT}(S)$ is unique.

There are many algorithms for computing the Delaunay triangulation of a point set following the well-known paradigms for geometric algorithms like divide and conquer, sweep-line, and also RIC. For all these algorithms the only geometric predicates required are the orientation and incircle tests[2] and the comparison of coordinates.

The maybe most elegant algorithm is the RIC developed by Guibas, Knuth and Sharir in [GKS92]. In this algorithm the points are inserted in random order into the triangulation. The starting triangulation consists of a triangle with endpoints at infinity and hence containing all sites.

When a new point p is inserted, the triangle $\Delta(qrs)$ containing p is located and split into three new triangles by connecting its vertices to p. Then for each new triangle $\Delta(pab)$ and its neighboring old triangle $\Delta(abc)$ it is checked if $incircle(a,b,c,p)$ holds. If not, the edge a,b is replaced by the edge p,c. This generates two new triangles which are also checked. The expected number of edges generated by the algorithm is less than $6n$ and the number of triangles generated is at most twice the number of edges generated[3] and hence overall the update step takes expected time $O(n)$.

To locate the triangle containing p, a simple acyclic point location graph is build, in which each node represents a triangle that existed at some time during the algorithm. If a triangle is destroyed by splitting it or flipping an edge, its children in the graph will be the triangles generated by this operation. To locate a point, we start at the infinite triangle and check which child triangle contains the point, and then again check the children of this triangle, until we reach the bottom of the search graph, and thus have found the triangle of the current triangulation containing p. In each search step an orientation test (p,q,r) is performed where (q,r) is the common edge of the children triangles.

The expected time of the algorithm is $O(n \log n)$, which is the optimal asymptotic running time for the construction of Delaunay triangulations.

4.1 Controlled Perturbation

We derive and analyze the controlled perturbation version of the algorithm in three steps. We first derive the guards and bound predicates. Then we give a quantitative version of the statement: if the input is in sufficiently general position the guarded algorithms succeeds. Finally, we show that a δ-perturbation with sufficiently large δ transforms any input into a sufficiently general input with constant probability.

In the following we assume that all coordinates are bounded by M after the δ-perturbation. We use *Orient* and *Incircle* to denote the absolute values of the orientation and incircle determinants. Floating point comparisons of

[2]The insphere test in d-dimensions decides for a sequence of $d+2$ points $p_0,\dots,p_d,p$ in $\mathbb{R}^d$ whether the point p lies outside, on, or inside the oriented sphere defined by the first $d+1$ points. We have $insphere(p_0,\dots,p_d,p) = orient(l(p_0),\dots,l(p_d),l(p))$ where for a point $q=(q_1,\dots,q_d) \in \mathbb{R}^d$, $l(q) = (q_1,\dots,q_d,q_1^2+\dots+q_d^2) \in \mathbb{R}^{d+1}$ is the lifting of q onto the paraboloid of revolution in R^{d+1}. For $d=2$, the insphere test is called incircle test.

[3]The average degree of a node in a planar graph is less than six and hence backwards analysis [CMS93] gives a $6n$ bound on the expected number of edges constructed. Let m be the number of edges constructed. When a point of degree k is inserted, we generate $2(k-3)+3$ new triangles (three for the first three edges and two for each additional edge) and hence the total number of generated triangles is $2m-3n$.

coordinates are exact and need no guards. Using table 1 we obtain the error bounds

$$B_{Orient} = 24 \cdot M^2 2^{-p}, \quad B_{Incircle} = 432 \cdot M^4 2^{-p}\,.$$

By plugging them into (2.2) we get guard predicates $\mathcal{G}_{Orient}$, $\mathcal{G}_{Incircle}$ and bound predicates $\mathcal{B}_{Orient}$, $\mathcal{B}_{Incircle}$ for the incircle and orientation test.

LEMMA 4.1. *If $\mathcal{G}_{Orient}$ and $\mathcal{G}_{Incircle}$ hold for all orientation and incircle tests performed, the guarded algorithm succeeds.*

Let us emphasize one more time that at runtime, only the guard predicates are evaluated. But since analyzing the behavior of the guard predicates seems hard, we will show that under certain conditions, the probability of a bound predicate failing is not too large. As a bound predicate not failing implies the successful evaluation of the respective guard predicate, we can obtain a lower bound on the success probability for the guarded execution of our algorithm. We next give geometric interpretations of these conditions.

LEMMA 4.2. *Let p, q, and r be three points, Δ the triangle defined by them and v and R the center and radius of the circumcircle of Δ. Then $Orient(p,q,r) = 2 \cdot \mathrm{area}(\Delta)$ and $Incircle(p,q,r,t) \geq 2 \cdot \mathrm{area}(\Delta) R |\mathrm{dist}(v,t) - R| \geq \mathrm{area}(\Delta)^{3/2} |\mathrm{dist}(v,t) - R|$.*

Proof: The first equality is standard. For the second inequality, we start with the observation that the determinant $Incircle(p,q,r,t)$ evaluates to six times the signed volume $\mathrm{vol}(L)$ of the simplex L defined by the four points $l(p), l(q), l(r)$ and $l(t)$. Here $l(p)$ denotes the projection of p onto the paraboloid $P : x^2 + y^2 - z = 0$.

Let $l(\Delta)$ be the triangle defined by the lifted points $l(p)$, $l(q)$, $l(r)$ and $\mathrm{area}(l(\Delta))$ be its area, and let E be the plane defined by $l(p)$, $l(q)$, $l(r)$. The equation of E can be obtained as follows. Let $v = (a/2, b/2)$. The points (x,y) on the circumcircle satisfy $(x - a/2)^2 + (y - b/2)^2 - R^2 = 0$ and their liftings have z-coordinate $z = x^2 + y^2 = ax + by + R^2 - a^2/4 - b^2/4$. Thus these liftings lie in the plane with equation $z = ax + by + c$ and $c = R^2 - a^2/4 - b^2/4$. This is the equation of E. Finally, let α denote the angle between E and the (x,y)-plane. Then $\mathrm{vol}(L) = \mathrm{area}(l(\Delta))h/3$ where h is the distance of $l(t)$ from the plane E. Also, $\mathrm{area}(l(\Delta)) = \mathrm{area}(\Delta)/\cos(\alpha)$ and $h = \cos(\alpha) h_v$, where h_v is the vertical distance of t from E. Thus

$$\mathrm{vol}(L) = \mathrm{area}(l(\Delta))h/3 = \mathrm{area}(\Delta)h_v/3.$$

To compute h_v we compare the z-coordinates of $l(t)$ and the projection of t onto E. We have

$$\begin{aligned} h_v &= \left|(t_x^2 + t_y^2) - (at_x + bt_y + c)\right| \\ &= \left|\left(t_x - \frac{a}{2}\right)^2 + \left(t_y - \frac{b}{2}\right)^2 - \left(\frac{a^2}{4} + \frac{b^2}{4} + c\right)\right| \\ &= \left|\mathrm{dist}^2(v,t) - R^2\right| \geq R \cdot |\mathrm{dist}(v,t) - R|\,. \end{aligned}$$

Combining our inequalities we obtain $Incircle(p,q,r,t) \geq 2 \cdot \mathrm{area}(\Delta) R |\mathrm{dist}(v,t) - R|$. For fixed circumradius, the equilateral triangle maximizes the area and hence $\mathrm{area}(\Delta) \leq 6 \cdot R\cos(\pi/6) \cdot R\sin(\pi/6) = 4R^2$. Conversely $R \geq \sqrt{\mathrm{area}(\Delta)}/2$ and hence $Incircle(p,q,r,t) \geq \mathrm{area}(\Delta)^{3/2} |\mathrm{dist}(v,t) - R|$. □

We have now arrived at geometric conditions for success. Whenever the orientation of three points is tested, they must form a triangle of area at least $B_{Orient}/2$, and whenever the incircle property is tested for four points, the fourth point must lie outside an annulus of half-width $B_{Incircle}/\mathrm{area}(\Delta)^{3/2}$ around the circumcircle of the triangle Δ formed by the first three points. The latter condition is the more stringent one, as we will see below.

We come to the analysis of the failure probability of the guarded Delaunay algorithm. For that it is convenient to require additional properties – let us call them *assertions* –, namely that any two perturbed points have distance at least ξ and that any triangle formed in the course of the algorithm has area at least ξ_Δ with $\xi_\Delta \geq B_{Orient}/2$. We will fix these constants later. These assertions – as the bound predicates – do not have to be checked at runtime, but we will argue about the probability of them being fulfilled.

The expected number of edges constructed by the algorithm is bounded by $6n$ and hence, by Markov's inequality, the probability that more than $24n$ edges are constructed is at most $1/4$. For the purpose of the analysis, we consider a run of the guarded algorithm constructing more than $24n$ edges a failure and restrict attention to runs constructing at most $24n$ edges and hence at most $48n$ triangles.

For the analysis, we may assume that the perturbation is chosen in on-line fashion. When we perturb the i-th point, the positions of the previously inserted points are already fixed and the new point is perturbed to a random point in a disc of area $\pi\delta^2$. We will choose δ such that at most a $1/(4n)$ fraction of the points in the disc are forbidden by one of our conditions. We use p to denote the i-th point.

We want that p has distance at least ξ from all previous points. This excludes a region of size at most $n\pi\xi^2$.

During the insertion of p, we perform a number of orientation tests $orient(q,r,p)$ and we construct a number of new triangles $\Delta(q,r,p)$. In each case, (q,r) is a previously constructed edge. We always want that $\mathrm{area}(\Delta(q,r,p)) \geq \xi_\Delta$. There are at most $24n$ such pairs (q,r) to consider[4]. Since

[4]Observe that we do not have to consider all pairs (q,r), but only those which formed an edge in some triangulation.

q and r where inserted earlier, we know that $\mathrm{dist}(q,r) \geq \xi$. Hence, if p is placed outside a strip of half-width $2\xi_\Delta/\xi$ about the line $\ell(q,r)$, the triangle has the desired size. The area of the intersection of such a strip with a circle of radius δ is at most $2\delta \cdot 4\xi_\Delta/\xi$ and hence the total size of the forbidden region is at most $24n \cdot 8\delta\xi_\Delta/\xi$.

We also perform a number of incircle tests $incircle(q,r,s,p)$. In each such case, the first three points form a triangle of the current Delaunay triangulation. There are at most $2n$ such triangles[5] and each has area at least ξ_Δ. The forbidden region is an annulus of half-width at most $B_{Incircle}/\xi_\Delta^{3/2}$ and the area of the intersection of this annulus with a disk of radius δ is at most $2\pi\delta \cdot 2B_{Incircle}/\xi_\Delta^{3/2}$. The total size of the forbidden region is thus bounded by $8n\pi\delta B_{Incircle}/\xi_\Delta^{3/2}$. We summarize:

LEMMA 4.3. *Let ξ and ξ_Δ be positive constants with $\xi_\Delta \geq B_{Orient}/2$. If $\pi\delta^2 \geq 4n \cdot (n\pi\xi^2 + 192n\delta\xi_\Delta/\xi + 8n\pi\delta B_{Incircle}/\xi_\Delta^{3/2})$, the success probability of the guarded algorithm is at least $1/2$.*

Proof: The guarded algorithm fails if during its execution too many objects are created or some insertion fails. An insertion fails if the chosen perturbation leaves the point in a forbidden region. The probability of the former is at most $1/4$, the probability of the latter is at most $1/(4n)$. Hence the probability of the guarded algorithm failing is at most $1/4 + n \cdot 1/(4n) = 1/2$. □

We are aiming for a solution to the inequalities from the lemma with minimal δ, Recall that $B_{Incircle}$ and B_{Orient} are functions of M and p. An optimal solution is easily obtained by numerical methods. For example, assume we have $n = 2^6 = 64$ points each with coordinates in the range $[-127,\ldots,127]$ (i.e. $M = 2^7$) and we are running our implementation on a SUN Sparc station which provides long doubles of $p = 112$ bit precision. For $\xi = 0.4 \cdot 10^{-3}$ and $\xi_\Delta = 0.2 \cdot 10^{-10}$, the above theorem tells us that if we choose $\delta = 0.10724$, the probability of $\mathcal{A}_m$ succeeding is $p_{\mathrm{succ}} \geq 1/2$ and hence the expected running time of $\mathcal{CP}$ is $O(n\log n)$.

We now derive an approximate solution to obtain a feeling for the quantities involved. In doing so, we will ignore constant factors. If we choose δ such that $\pi\delta^2$ is at least three times the value of each term on the right hand side we are on the safe side, i.e., ignoring constant factors $\delta^2 \geq n^2\xi^2$, $\delta^2 \geq n^2\delta\xi_\Delta/\xi$, and $\delta^2 \geq n^2\delta B_{Incircle}/\xi_\Delta^{3/2}$. Only one of the right hand sides increases in ξ and hence we may assume that the first constraint is tight, i.e., $\xi = \delta/n$. Analogously we may assume that the second constraint is tight, i.e., $\xi_\Delta = \delta\xi/n^2 = \delta^2/n^3$ Thus (from $\xi_\Delta \geq B_{Orient}/2$), we get $\delta^2 \geq n^3 B_{Orient} = n^3M^2 2^{-p}$ and $\delta^2 \geq n^2\delta B_{Incircle}/(\delta^2/n^3)^{3/2}$ or $\delta^4 \geq B_{Incircle}n^{13/2} = M^4 2^{-p} n^{13/2}$. In other words, $2^p \geq \max((M/\delta)^2 n^3, (M/\delta)^4 n^{13/2}) = (M/\delta)^4 n^{13/2}$.

THEOREM 4.1. *If the guarded algorithm is executed with precision p, where $p \geq C(\log M - \log\delta + \log n + 1)$ for a suitable constant C, it succeeds with probability at least 1/2.*

4.2 Determining the Optimal Precision

Our estimates in the preceding section are extremely pessimistic and hence one should not use them in a real implementation. Assume δ is given. We advise to start with a small precision p_0 and to double it in case of k repeated failures, for some constant k. Let l_0 be the smallest non-negative integer such that execution with precision $2^{l_0}p_0$ has error bound less than $1/2$. The scheme will terminate after an expected number of $l_0 + O(1)$ rounds. Moreover, if the running time of our algorithm is $p^aT(n)$ for some $a > 1$ (for example, $a = 2$, if the algorithm uses multiplication and multiplication is implemented by the school method), the expected running time is $O((\sum_{0\leq l<l_0}(2^lp_0)^a + \sum_{l\geq l_0} 2^{-k(l-l_0)}(2^lp_0)^a)T(n)) = O(2^{l_0}p_0T(n))$ if $k > a$. A similar scheme can be used to find the optimal value of δ for fixed precision.

4.3 Lazy Perturbations

Another perturbation approach would be to perturb sites only if during their insertion some predicates could not be certified and then repeat the insertion. This seems to work well in practice, see Section 8. There are however two drawbacks. First, in this case the perturbation depends on the insertion order and hence the probabilistic analysis based on configuration spaces does not seem to carry over (at least we were unable to carry it over). Second, it is necessary to explicitly check the auxiliary assertions that every edge constructed has length at least ξ and that every triangle constructed has area at least ξ_Δ. This requires a change of the algorithm, see [Kl04] for more details. We refer to the two approaches as *standard* and *lazy*.

5 Convex Hulls in Arbitrary Dimensions

The Idealistic Algorithm: We use the randomized incremental algorithm analyzed in Clarkson, Mehlhorn, and Seidel [CMS93]. We use d to denote the dimension of the underlying space and for a set R use $conv R$ to denote its convex hull. We assume our input points to be in general position and denote them $x_1, x_2, \ldots, x_n$ in the order of insertion. Let $S = \{x_1,\ldots,x_n\}$ be our set of points and let f_r be the expected number of facets of $conv R$ for a random subset $R \subseteq S$ of size r. We use CH_i to denote the convex hull of the first i points. The algorithm maintains a triangulation T of the current hull CH. The triangulation is initialized to the simplex spanned by the first $d+1$ points. When a point x is added, the triangulation is updated as follows: If $x \in CH$, we leave T as it was. If $x \notin CH$, for every facet F of CH visible from x, we add to T the simplex $S(F,x) = conv(F \cup \{x\})$. We call

[5] Observe that we do not have to consider all triples (q,r,s), but only those which form a triangle in the current triangulation.

F the *base* of this simplex and x its peak. A facet F is visible to x when $S(F,x)$ meets the hull only at F. Let A_i (A for additional) be the facets of CH_i which were not facets of CH_{i-1}.

We need the following fact (shown in [CMS93]):

1. The cost of adding point x_j is bounded by the number of facets in $\cup_{j<i} A_j$ visible from x_i.
2. The expected number of facets in $\cup_{j\le i} A_j$ is $C_i := \sum_{j\le i} d f_j / j$.
3. The expected running time of the algorithm is $O(d^5)\sum_{j\le n} \frac{nf_j}{j(j-1)}$.

The Guarded Algorithm: The algorithm uses only the orientation test given by determinant (2.1). When this test is applied during the insertion of x_i, the first d points define a facet of CH_j for some $j < i$ and the last point is x_i. We therefore write the test as $orient(F,x_i)$ with facet F and point x_i. We use the guard $\mathcal{G}_d$ defined in section 2.

The Analysis: The perturbation must guarantee $Orient(F,x_i) \ge B_d$ for every orientation test performed by the algorithm, where B_d is as defined in section 2. As in the case of Delaunay triangulations, we guarantee more: namely a minimum relative volume of all faces of all dimensions. The details are as follows.

Let h_0, h_1, $\dots h_{d-1}$ be a sequence of positive reals. For an l-face f define forbidden(f,h_l) to be the set of points with distance less than h_l from the l-flat containing f. So for a 0-face f (= a vertex) forbidden(f,h_0) is an open ball with radius h_0 centered at f. For a 1-face f (= an edge) forbidden(f,h_1) is an open hyper-cylinder of radius h_1 whose axis is the line containing f.

The relative volume rvol(f) of a face f is the volume of f when viewed as a subset of its affine hull. Define s_0, s_1, $\dots$, s_d by $s_0 = 1$ and

$$s_l = \frac{h_{l-1}\cdots h_0}{l!} = \frac{h_{l-1}s_{l-1}}{l} \quad \text{for } l \ge 1 .$$

LEMMA 5.1. *Let $CH = CH_i$ be the current hull and let $p = x_{i+1}$ be the point to be inserted. If*

1. *$d!\cdot s_d \ge B_d$ where B_d is as defined in section 2,*
2. *rvol$(f) \ge s_l$ for every l-face f of CH,*
3. *$p \notin$ forbidden(F,h_{d-1}) for any facet F of any previous hull, and*
4. *$p \notin$ forbidden(f,h_l) of any l-face f of CH*

then the insertion of p is succeeds and 2. holds for all faces of $CH' = CH_{i+1}$.

Proof: When we insert p, we perform orientation tests $orient(F,p)$ where F is a facet of some previous hull. The value of $orient(F,p)$ is $d!$ times the signed volume of the simplex $conv(F,p)$. Let h be the distance of p from the hyperplane containing F. Then

$$\text{vol}(conv(F,p)) = h\cdot \text{rvol}(F)/d \ge h_{d-1}\cdot s_{d-1}/d = s_d$$

and hence $Orient(F,p)$ is at least $d!\cdot s_d$, which in turn is at least B_d. Thus we can conclude that the insertion is f-safe, which proves the first part of the theorem.

For the second part observe that an l-face f' of CH' is either an l-face of CH (in which case 2. already holds for f') or has the form $conv(f,p)$ with f an $l-1$-face of CH. In this case let h be the distance of p from the affine hull of f. Then

$$\text{rvol}(conv(f,p)) = h\cdot \text{rvol}(f)/l \ge h_{l-1}\cdot s_{l-1}/l = s_l,$$

which concludes the proof of the theorem. □

The Lemma gives us a geometric condition for success. When we perturb a new point, we must avoid the forbidden regions. We next estimate their size. Analogously to the Delaunay case, we only need to consider the intersection with the δ-ball around a point. Recall that the volume of an l-dimensional ball of radius δ is $f_l\delta^l$ for some constant f_l.

LEMMA 5.2. *Let U^δ be a d-dimensional ball of radius δ and let f be an l-face. Then*

$$\frac{\text{vol}(U^\delta \cap \text{forbidden}(f,h))}{\text{vol}(U^\delta)} \le \frac{c_l c_{d-l}}{c_d}\left(\frac{h}{\delta}\right)^{d-l} .$$

Proof: The ratio is maximized if the l-flat containing f passes through the center of U^δ. Hence assume w.l.o.g. that U^δ is centered at the origin and that the l-flat corresponds to the flat spanned by the first l coordinate vectors. A point $(x_1,\dots,x_d) \in U^\delta \cap \text{forbidden}(f,h)$ must satisfy both

$$x_1^2 + \dots x_d^2 \le \delta^2 \quad \text{and} \quad x_{l+1}^2 + \dots + x_d^2 \le h^2$$

and hence is contained in the set defined by

$$x_1^2 + \dots x_l^2 \le \delta^2 \quad \text{and} \quad x_{l+1}^2 + \dots + x_d^2 \le h^2 .$$

But this set has volume $c_l\delta^l\cdot c_{d-l}h^{d-l}$. □

We need to guarantee 3. and 4. of lemma 5.1 for every insertion. Recall that the expected number of facets constructed by the algorithm is C_n. The probability that more than $4C_n$ facets are constructed is at most $1/4$. We consider any run which constructs more than $4C_n$ facets a failure and proceed under the assumption that at most $4C_n$ facets are constructed. Every face is a subset of some facet and hence we may assume that the number of faces of any dimension constructed by the algorithm is at most $N := 4\cdot 2^d C_n$. Thus the position of a new point is constrained by at most N forbidden regions.

It remains to choose the h_l's. Let $c = \max c_l c_{d-l}/c_d$.We fix them so as to make all forbidden regions the same size, i.e., for $l < d-1$ we choose h_l such that

$$c(\frac{h_l}{\delta})^{d-l} = \frac{h_l}{\delta} .$$

Set $h = h_{d-1}$. Then

$$h_l = \delta(\frac{h}{\delta})^{1/(d-l)} \quad \text{or} \quad h_{d-i} = \delta(\frac{h}{\delta})^{1/i}$$

for $0 \le l \le d-1$ or $1 \le i \le d$. From $B_d = h_0 h_1 \cdots h_{d-1}$ we obtain

$$\delta^d \prod_{1 \le i \le d} (\frac{h}{\delta})^{1/i} = B_d \quad \text{or} \quad (\frac{h}{\delta})^{H_d} = \frac{B}{\delta^d}$$

or

$$\frac{h}{\delta} = (\frac{B}{\delta^d})^{1/H_d} \ge \bigl(C(\frac{M}{\delta})^d \varepsilon\bigr)^{1/H_d}$$

for some constant C and $H_d := \sum_{i=1}^n 1/n$ being the n-th harmonic number. Every forbidden region uses at most a fraction ch/δ of U^δ and hence we need

$$cN\bigl(C(\frac{M}{\delta})^d \varepsilon\bigr)^{1/H_d} \le \frac{1}{2n} \quad \text{or} \quad \varepsilon \le \bigl(\frac{\delta}{M}\bigr)^d \frac{1}{C(2ncN)^{H_d}} .$$

We summarize in:

THEOREM 5.1. *If $p \ge d\log(M/\delta) + H_d \log nC_n + O(dH_d)$, the guarded convex hull algorithm succeeds with probability at least* $1/2$.

6 Delaunay Triangulations in Arbitrary Dimensions

The simplest way to construct the Delaunay triangulation of a set of points in $\mathbb{R}^d$ is to construct the convex hull of the lifted points in $d+1$-dimensions. The projection of the lower hull is the Delaunay triangulation. The only predicate used by the algorithm is the orientation predicate of the lifted points; it is equivalent to the insphere predicate of the original points.

We cannot use the results of the preceding section as we perturb the original points and not the lifted points. Since the additional coordinate is a function of the original coordinates we cannot perturb it independently of the others. However, we can reuse the analysis of the preceding section and combine it with the analysis of the Delaunay algorithm in the plane. Details are given in the full paper.

7 Forbidden Regions in Surprise-Free RICs

In the d-dimensional convex hull problem we associated a forbidden region with each l-face, $0 \le l < d$, of the current hull, but *not* with every l-subset of the current point set, the reason being that only the former subsets can develop into facets by further insertions. In this section we generalize this observation to arbitrary surprise-free RICs. RICs were introduced by Clarkson and Shor [CS89].

Let S be a set with n elements, which we call objects, and let $\mathcal{F}(S)$ be a multi-set of subsets of S. For simplicity, we assume that all subsets have the same size d. We call the elements of $\mathcal{F}(S)$ ranges. For a region $F \in \mathcal{F}(S)$ and an object $x \in F$, we say that F relies on x or x supports F. For $R \subseteq S$, define $\mathcal{F}(\mathcal{R}) = \{\mathcal{F} \in \mathcal{F}(\mathcal{S})\ ;\ \mathcal{F} \subset \mathcal{R}\}$ with multiplicities preserved. We also assume a conflict relation $C \subseteq S \times \mathcal{F}(S)$ between objects and regions with the property that if $(x,F) \in C$ then F does not rely on x.

In the convex hull problem, the regions are d-subsets F of S and each subset occurs twice. The two copies correspond to the two open halfspaces $H_1(F)$ and $H_2(F)$ defined by F. An object x is in conflict with the region denoting $H_i(F)$ iff $x \in H_i(F)$. The regions in $\mathcal{F}(S)$ which do not conflict with any point in S correspond to the facets of the convex hull of S.

For a subset $R \subseteq S$, $\mathcal{F}_0(R)$ denotes the set of all $F \in \mathcal{F}(R)$ having no $x \in R$ with $(x,F) \in C$, that is, $\mathcal{F}_0(R)$ is the set of regions over R which do not conflict with any object in R. In the randomized incremental construction [CS89] of $\mathcal{F}_0(S)$, the objects are considered in random order $x_1,\ldots,x_n$ and, in the general step, $\mathcal{F}_0(R_{j+1})$ is constructed from $\mathcal{F}_0(R_j)$ where $R_j = \{x_1,\ldots,x_j\}$.

The only test used by the generic RIC (concrete realizations may use other tests for increased efficiency) is the conflict test $C(F,p)$ between regions F and objects p. Moreover, when $p = x_j$ then $F \in \mathcal{F}_0(F_i)$ for some $i < j$. The perturbation of S must guarantee that all conflict tests are computed without error. It is natural to perturb the objects in S one by one. When x_j is inserted, it is perturbed and the perturbations of x_1 to x_{j-1} are already fixed. The perturbation must guarantee that all conflict tests $C(F,x_j)$ with $R\mathcal{F} \in \cup_{i<j}\mathcal{F}_0(R_i)$ are safe and it must also "prepare" for conflict tests $C(F',x_k)$ with $k > j$ and $x_j \in F'$.

What do we mean by prepare? For all l, $1 \le l \le d$, define the l-faces of R_j as the set of all l-subsets f of R_j for which there is a $d-l$-subset f' of $S \setminus R_j$ with $f \cup f' \in \mathcal{F}_0(R_j \cup f')$. If all elements in f' are inserted before any conflict of the region $f \cup f'$, then the algorithm might perform conflict tests $C(f \cup f', \cdot)$. The perturbation of x_j must prepare for these tests. In the convex hull example, assertions 2. and 4. of Lemma 5.1 serve this purpose.

As in the analysis of the convex hull problem it seems natural to introduce auxiliary assertions A_i for $1 \le i \le d+1$. The assertion A_i depends on i objects and we have: $A_{d+1}(R,p)$ implies that the floating point evaluation of $C(R,p)$ gives the correct result. For objects $p_1,\ldots,p_j$ define their forbidden region $\text{forbidden}(p_1,\ldots,p_j) = \{p\ ;\ \neg A_{j+1}(p_1,\ldots,p_j,p)\}$. We can now state and prove the

LEMMA 7.1. *Let $\mathcal{F}_0 = \mathcal{F}_0(R_i)$ be the current set of conflict free regions and and $p = x_{i+1}$ be the point to be inserted. If*

1. *$A_l(f)$ holds for every l-face f of $\mathcal{F}_0$,*
2. *$p \notin \text{forbidden}(F)$ for any region F of any $\mathcal{F}_0(R_j)$ with $j \le i$*
3. *$p \notin \text{forbidden}(f)$ of any l-face f of $\mathcal{F}_0$.*

then the insertion of p succeeds and 2. holds for all faces of $\mathcal{F}'_0 = \mathcal{F}_0(R_{i+1})$.

Proof: When we insert p, we perform conflict tests $C(F,p)$ where F is a region of some $\mathcal{F}_0(R_j)$ with $j \le i$. Since p is not in the forbidden region of F, we have $A_{d+1}(F,p)$ which in turn guarantees that the floating point evaluation of $C(F,p)$ returns the correct result.

An l-face f' of $\mathcal{F}'_0$ is either an l-face of $\mathcal{F}_0$ or has the form (f,p) where f is an $l-1$-face of $\mathcal{F}_0$. In the former case there is nothing to prove. In the latter case, we have $A_l(f,p)$ by the definition of forbidden region. □

How many l-faces can R_i have? For a general RIC we have no non-trivial bound. Call a RIC *surprise-free* if any l-face, $0 \le l \le d$, of R_i is a subset of some region in $\mathcal{F}_0(R_i)$. The RICs for convex hulls, Delaunay diagrams and line segment intersection are surprise-free. In fact, we are not aware of any RIC which can be turned into an efficient algorithm and is not surprise-free. There are however RICs which are not surprise-free. Take for example as the set of regions all subsets of size d which contain a particular element x. Before the insertion of x, there are no conflict-free regions. However, any l-subset of the current point set, $1 \le l < d$, is an l-face. For surprise-free RICs the number of l-faces of R_i is bounded by $\binom{d}{l}|\mathcal{F}_0(R_i)|$.

8 Experiments

We have implemented the controlled perturbation RIC for Delaunay triangulations in C++. We have implemented standard and lazy perturbations. Our implementation is able to use various number types for the underlying arithmetic. The lazy perturbation algorithm also checks the additional properties introduced in section 4.1, namely that two points have at least distance ξ and that every triangle hast at least area ξ_Δ. To test our algorithm, we compiled it with GNU C++ 3.3 under Linux and ran it on a 3.06GHz Intel Xeon. As input data set we used grids of various sizes and random points on the "flower" formed by eight intersecting circles. All input points have coordinates with absolute value less than 1000, hence $M = 1000$. We further compared the running time of a double version and a double interval version of our algorithm to an implementation using exact arithmetic. For all following tests we ran the algorithm a couple of times and give the average value.

Grid-size	CP (doubles)	CP (interval)	Exact
2601	0.05	0.13	0.27
10201	0.23	0.65	1.29
40401	1.10	3.02	5.96
160801	5.23	13.86	26.17

Tabelle 2: Timings for the lazy algorithm on a Grid using controlled perturbation (CP) or exact arithmetic.

8.1 Behavior of Standard and Lazy Controlled Perturbation

First we ran both the standard and lazy approach to determine the perturbation needed. For this we increase the perturbation if a predicate fails. The results are shown in table 3. While both algorithms need considerably smaller perturbations than suggested by our worst case formulae, for larger inputs the perturbation required by the lazy algorithm is much smaller. Recall however that the perturbation bound derived by us does not hold for the lazy algorithm, and we also do not know if its expected running time is still $O(n \log n)$. Hence we also monitored the number of triangles generated by the algorithm and the number of performed point-in-triangle tests. The results are also shown in table 3 and suggest that in practice these values are no worse than for the normal algorithm. More extensive experiments can be found in [Kl04]. We also note that by choosing a higher precision p or by using interval arithmetic, the required amount of perturbation can be drastically reduced, in particular the standard perturbation algorithm is then able to solve the large instances with a reasonably small amount of perturbation.

8.2 Running Time of the Lazy Algorithm

Since the lazy algorithm gives a better perturbation bound and our experiments suggest that it is usable in practice, we compared it to an exact implementation of the RIC Delaunay algorithm. We also ran our algorithm a second time with interval arithmetic instead of the static bounds given by table 1. While interval arithmetic gives smaller perturbations, its running time is much worse. It is however still faster than the exact implementation. Timings are shown in table 2. Observe that for the lazy algorithm, the auxiliary assertions need to be checked. But this only requires the calculation of three additional expressions per incircle test, which furthermore are numerically less demanding than the incircle test itself.

9 Conclusions

We pointed out that controlled perturbation is a general scheme for converting idealistic algorithms, i.e., algorithms designed for non-degenerate inputs and the Real-RAM model of computation, into algorithms which can be executed with multi-precision floating point arithmetic: every branch on the sign of an expression E is guarded by a guard predicate $\mathcal{G}_E$ with the property: if $\mathcal{G}_E$ evaluates to true when evaluated

	Standard Perturbation			Lazy Perturbation	
Input, pts	triangles	locates	avg.perturbation	max.perturbation.	avg.perturbation
Flower, 400	3396	6220	0.05877	0.001	0.001
Flower, 2000	17319	43065	0.20473	0.0076	0.0023
Flower, 10000	100389	284043	0.79845	127.83	51.76
Grid, 441	3867	7516	0.00308	0.001	0.001
Grid, 2601	23255	62217	0.00675	0.0076	0.0043
Grid, 10201	91981	293882	0.01299	0.292	0.105
Grid, 160801	1448884	6514681	0.05181	exceeded grid-size	

Tabelle 3: Comparison of lazy and standard perturbation

with floating point arithmetic, the evaluation of E with floating point arithmetic will give the correct sign. One obtains the guard predicates by error analysis.

Instead of executing the program on the actual input it is executed on a perturbed input $\tilde{x}$ selected at random from a δ-neighborhood of the true input. For fixed input parameters (number of objects, maximal coordinate of any object), the perturbation bound δ and the precision p of the floating point system depend inversely on each other, the smaller δ, the larger p, and vice versa. The details of the relation can be determined either analytically or experimentally. In the experimental setting and for fixed δ, one simply doubles p starting from a small value, say $p_0 = 52$, the precision of native double precision floating point arithmetic, until the algorithm succeeds.

In the analysis, one has to give geometric meaning to statements of the form: the value of an expression E is larger than a certain bound B_E. The geometric interpretations lead to forbidden regions for the placement of points.

If one is only interested in implementing a CP algorithm, interval arithmetic can be used to check the guard predicates without having to actually derive them. Hence the only additional knowledge needed is what a δ-perturbation of the input objects means.

References

[CGA] CGAL (Computational Geometry Algorithms Library). http://www.cgal.org.

[Cla92] K.L. Clarkson. Safe and effective determinant evaluation. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS'92)*, pages 387–395, 1992.

[CMS93] K. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Computational Geometry: Theory and Applications*, 3:185–212, 1993. http://www.mpi-sb.mpg.de/ mehlhorn/ftp/CMS-FourResults.ps.

[CS89] K.L. Clarkson and P.W. Shor. Applications of random sampling in computational geometry, II. *Journal of Discrete and Computational Geometry*, 4:387–421, 1989.

[DH91] P. Deuflhard and A. Hohmann. *Numerische Mathematik: Eine algorithmisch orientierte Einführung*. Walter de Gruyter, 1991.

[FvW93] S. Fortune and C. van Wyk. Efficient exact integer arithmetic for computational geometry. In *7th ACM Conference on Computational Geometry*, pages 163–172, 1993.

[GKS92] L. Guibas, D. Knuth, and M. Sharir. Randomized Incremental Construction of Delaunay and Voronoi Diagrams. *Algorithmica* 7: 381-413, 1992.

[HL03] D. Halperin and E. Leiserowitz. Controlled perturbation for arrangements of circles. In *SoCG*, pages 264–273, 2003.

[HR] D. Halperin and S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. available from Halperin's home page; a preliminary version appeared in SoCG 1999, pages 163–172.

[HS98] Halperin and Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *CGTA: Computational Geometry: Theory and Applications*, 10, 1998.

[JRZ91] M. Jünger, G. Reinelt, and D. Zepf. Computing correct Delaunay triangulations. *Computing*, 47:43–49, 1991.

[KLN91] M. Karasick, D. Lieber, and L.R. Nackman. Efficient Delaunay triangulation using rational arithmetic. *ACM Transactions on Graphics*, 10(1):71–91, January 1991.

[Kl04] C. Klein. Controlled Perturbation for Voronoi Diagrams. Master Thesis, Universität des Saarlandes, 2004.

[KMP$^+$] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computations. to appear in ESA 2004, http://www.mpi-sb.mpg.de/ mehlhorn/ftp/ClassRoomExample.ps.

[LED] LEDA (Library of Efficient Data Types and Algorithms). http://www.mpi-sb.mpg.de/LEDA/leda.html.

[MN94] K. Mehlhorn and S. Näher. The implementation of geometric algorithms. In *Proceedings of the 13th IFIP World Computer Congress*, volume 1, pages 223–231. Elsevier Science B.V. North-Holland, Amsterdam, 1994. http://www.mpi-sb.mpg.de/ mehlhorn/ftp/ifip94.ps.

[MN99] K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999. 1018 pages.

[ST04] Daniel A. Spielman and Shang-Hua Teng. *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time*, In *Journal of the ACM* 51(3), 385–463, 2004

[Yap93] C.K. Yap. Towards exact geometric computation. In *Proceedings of the 5th Canadian Conference on Computational Geometry (CCCG'93)*, pages 405–419, 1993.