How to Compute the Voronoi Diagram of Line Segments: Theoretical and Experimental Results

Christoph Burnikel Kurt Mehlhorn Stefan Schirra

Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany

Abstract. Given a set of non-intersecting (except at endpoints) line segments in the plane we want to compute their Voronoi diagram. Although there are several algorithms for this problem in the literature [Yap87, For87, CS89, BDS+92, KMM90] nobody claims to have a correct implementation. This is due to the fact that the algorithms presuppose *exact* arithmetic and that the Voronoi diagram of segments requires to compute with non-rational algebraic numbers. We report about a detailed study of the numerical precision required for evaluating the geometric test exactly and about first experimental experiences. More specifically, we improve the precision bound implied by classical root separation results by more than two orders of magnitude and we compare the implementation strategies suggested by our theoretical results.

1 Introduction

The Voronoi diagram for a set S of sites partitions the plane into a set of regions one for each site. The region of site $s \in S$ contains all points in the plane that are closer to s than to any other site in S. The Voronoi diagram is one of the most useful structures in computational geometry, cf. [Aur91, OBK92] for a survey. In this paper we concentrate on the Voronoi diagram of line segments, i.e., S is a set of points and open line segments. We assume that for each segment s in S its two endpoints also belong to S.

Several algorithms for the Voronoi diagram of line segments are known; they are based on divide-and-conquer [Yap87], plane sweep [For87] and randomized incremental construction [CS89, BDS+90, KMM90]. *In this paper we discuss the implementation of these algorithms.* The descriptions of all algorithms mentioned above presuppose exact real arithmetic for the implementation of the underlying geometric tests. The so-called *incircle test* is a basic tool in all algorithms: Given a Voronoi vertex defined by three sites and a fourth site decide whether the fourth site intersects, touches, or avoids the clearance circle centered at the Voronoi vertex, cf. Figure 1. Note that the coordinates of Voronoi vertices are *non-rational algebraic numbers* even when all segment endpoints have rational coordinates. How can one implement the incircle test or more generally the geometric primitives needed in an algorithm? There are three approaches.

The first approach uses floating point arithmetic. In order to perform an incircle test the (approximate) coordinates of the Voronoi vertex v are computed

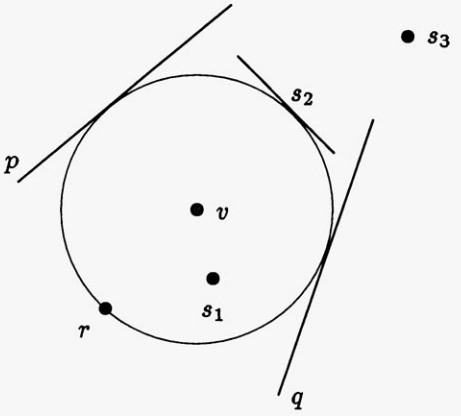

Fig. 1. The Incircle Test: The Voronoi vertex v is defined by sites p, q, and r. The three sites s_1, s_2, and s_3 intersect, touch, and avoid the clearance circle centered at v.

first and then the distances between v and one of the defining sites and between v and the new site are compared. If the distances differ less than some parameter ε then they are considered equal. The choice of the parameter ε is witch-craft. Butz and Rech [But94] and Seel [See94] have implemented the algorithm of Yap [Yap87] and Klein, Mehlhorn, and Meiser [KMM90] using this approach. The implementations work on many examples but are also known to fail on some examples. Similar experiences were reported to us by Drysdale and Dobrindt.

The second approach also uses floating point arithmetic but resolves the outcome of "doubtful" tests in a different way. Whenever floating point arithmetic does not suffice to determine the outcome of a test the outcome is fixed arbitrarily *but in a way that is consistent* with previous tests. In this way the state of the program is always legal, i.e., reachable for some input. It is not clear, however, whether this input is "close" to the real input for an appropriate definition of "close". This approach was used by Li and Milenkovic [LM92] for the convex hull problem, by Fortune [For92] for the Delaunay triangulation of point sets and by Sugihara, Ooishi, and Imai [SOI90] for the Voronoi diagram of line segments. Li and Milenkovic can prove that their algorithm computes an approximate convex hull, Fortune can show that his algorithm computes a planar graph and an embedding of it such that the embedded graph is close to the true Delaunay triangulation (however, the embedding is not necessarily planar), and Sugihara et al compute a planar graph and an embedding of it and verify experimentally that the diagram is close to the true Voronoi diagram (again, the embedding is not planar, in general). Note that only the solution for the convex hull problem can be considered fully satisfactory. However, even in this case the algorithm and in particular its analysis is quite involved so that it seems unlikely that the second approach can be carried out for many geometric problems.

The third approach uses exact arithmetic. This approach was used by Kara-

sick, Lieber, and Nackmann [KLN91] and Fortune and van Wyk [Fv93] for the Delaunay diagram of points and by Ottmann, Thiemt, and Ullrich [OTU87] and Benoumer et al [BJMM93] and Mehlhorn and Näher [MN] for the intersection of line segments. These papers report that a careful implementation of arbitrary precision arithmetic and the use of a floating point filter yields a *provably correct* implementation whose running time is only a small constant factor (less than five) above the running time of an implementation using floating point arithmetic. For both problems mentioned above all objects computed have rational coordinates. In our case the coordinates are in general non-rational algebraic numbers. It is well known that exact computation with algebraic numbers is possible [Loo82, Mig92, Yap94] but only recently, Yap [Yap93], advocated their actual use in geometric computations. However, Yap did not report yet about any actual experiences. We do.

Theoretical Results: Let us assume that all point sites (recall that we assume that the endpoints of all segment sites are point sites) have integral homogeneous coordinates with at most k binary digits ($\equiv$ precision k). Classical root separation results (see for example [Mig92, Sec.4.3]) imply that precision about $9000k$ suffices to compute the incircle test exactly, i.e., if two distances agree in the first $9000k$ binary digits then they are equal. We improve this bound to $48k$ for the incircle test by observing that the radicals involved in the incircle test obey some algebraic identities and that the incircle test can therefore be resolved by repeated squaring.

The proof of the improved root separation bound suggests one way to compute the incircle test: Use repeated squaring and arbitray precision integer arithmetic (BIGNUM). Another possibility is to use arbitrary precision floating point arithmetic (BIGFLOAT) to compute the first $48k$ binary digits of the quantities to be compared. Clearly, if the quantities are different the difference is, on average, not in the last digit and hence much smaller precision should suffice on average. However, if the quantities are equal, the full precision is always required. This will be the case whenever the input is highly degenerate.

Blömer [Bl93] has shown that testing whether a sum of radicals is zero can be done in polynomial time. His result applies to the incircle test. In his algorithm [Bl93], the algebraic numbers involved in the incircle test have to be approximated to precision $100k$. We show that a zero test can be done by a computation with $12k$-bit integers.

This result suggests the following alternative strategy for the incircle test. Decide first whether the fourth site touches the clearance circle of the given three sites (computation with $12k$-bit integers). If not then compute the two distances to be compared (note that we know already that they are different) with precision k, $2k$, $4k$, $8k$, $16k$, $32k$, $48k$ until they are different. Both implementation strategies have their merit: The repeated squaring approach starts out with moderate precsision and needs the full precision only for the final steps of the computation. The BIGFLOAT approach might be able to decide a test by evaluating it with small accuracy. But in order to get the final result with some accuracy a larger accuracy is needed for the intermediate results.

We performed experiments with various types of data sets. The results are shown in Section 3.

2 Theory

We describe our theoretical results in the context of the randomized incremental algorithm by Klein, Mehlhorn, and Meiser [KMM90]. Following Sugihara, Ooishi, and Imai [SOI90] we first insert all point sites in random order and then all open segments in random order. This simplifies some of the geometric tests. Whenever a site s is added it is necessary to determine all features of the current diagram that conflict with s. A vertex v conflicts with s if s touches or intersects the clearance circle centered at v and an edge e conflicts with s if the Voronoi region of the new site s intersects e. The latter intersection can have one of four types: the entire edge, a single segment incident to one of the endpoints, two segments incident to the endpoints, a single segment not incident to any endpoint. In fact, the last case cannot occur since we insert all point sites first. Conflicts between vertices and sites can be decided by the incircle test, conflicts between edges and sites can be determined similarly; we are not going to treat them in this extended abstract. Once all conflicts are known it is easy to update the Voronoi diagram. This is a purely combinatorial operation. Since we insert all point sites first the standard analysis of randomized incremental constructions does not apply. However, we can still show:

Lemma 1. *The expected running time of the algorithm is* $O(n \log n)$.

Proof. (sketch): This is clear for the insertion of the point sites. For the insertion of the segment sites we observe that all conflicts can be found in time proportional to their number (there is no need to search through the history graph) and that the expected number of conflicts of the i-th segment is $O(n/i)$. The total structural change is therefore $O(n \log n)$. □

Let us turn to the incircle test next. Let v be a Voronoi vertex; v is defined by three sites. Here we will only discuss the case that two of the defining sites are lines and one is a point. All other cases are no more demanding computationally. We represent a point p by its homogeneous coordinates (x_p, y_p, z_p). For lines we use the equation $ax + by + c = 0$. We assume that x_p, y_p, z_p and a, b, c are k-bit integers. The bisector of a point $p = (x_p, y_p, z_p)$ and a line $\ell : ax + by + c = 0$ is given by the parabola

$$(ax + by + c)^2 = N((x - \tfrac{x_p}{z_p})^2 + (y - \tfrac{y_p}{z_p})^2),$$

and the bisector of intersecting lines $\ell_1 = a_1x + b_1y + c_1$ and $\ell_2 = a_2x + b_2y + c_2$ is given by the lines

$$\frac{a_1x + b_1y + c_1}{D_1} = \pm\frac{a_2x + b_2y + c_2}{D_2}$$

where $D_i = \sqrt{N_i}$ and $N_i = a_i^2 + b_i^2$.

Assume now that v is defined by the lines $\ell_1 = \bar{a}_1 x + \bar{b}_1 y + \bar{c}_1$ and $\ell_2 = \bar{a}_2 x + \bar{b}_2 y + \bar{c}_2$ and the point $p = (x_p, y_p, z_p)$. We transform the coordinate system such that $p = (0, 0, 1)$. This transforms the equation of line ℓ_i into

$$(\bar{a}_i z_p)x + (\bar{b}_i z_p)y + (\bar{a}_i x_p + \bar{b}_1 y_p + \bar{c}_1 z_p) = 0.$$

Hence the coefficients of the transformed lines are $(2k+2)$-bit integers. We use a_i, b_i, c_i to denote the coefficients of the transformed lines.

The Voronoi vertices (there are two) defined by the lines ℓ_1, ℓ_2, and point p are given by the intersection of a parabola and a line and have coordinates

$$\left(J + \alpha\sqrt{G}, I - \alpha\,\mathrm{sign}(rs)\sqrt{F}, X\right) \tag{1}$$

where $\alpha \in \{-1, 1\}$,

$$\begin{array}{ll} I = b_1c_2 + b_2c_1 & J = a_1c_2 + a_2c_1 \\ F = 2c_1c_2(D_1D_2 + b_1b_2 - a_1a_2) & G = 2c_1c_2(D_1D_2 + a_1a_2 - b_1b_2) \\ r = a_1D_2 - a_2D_1 & s = b_1D_2 - b_2D_1 \\ X = D_1D_2 - a_1a_2 - b_1b_2 & \end{array}$$

Note that $D_1D_2 = \sqrt{(a_1^2 + b_1^2)(a_2^2 + b_2^2)}$, i.e., the coordinates involve nested roots. It is useful to observe that $|r|\sqrt{G} = |s|\sqrt{F}$. In particular $\sqrt{FG}$ is an integer. This observation is crucial for what follows. The two Voronoi vertices differ in the cyclic order in which the three defining sites occur on the clearance circle. We do not discuss here how to select α given the cyclic ordering but only mention that our approach also handles that problem.

Assume now that the fourth site is a line ℓ_3 with the equation $a_3x + b_3y + c_3 = 0$. a_3, b_3, c_3 are again $2k$-bit integers. We compare the square of the distance of $v = (x_s, y_s, z_s)$ and $p = (0, 0, 1)$ with the square of the distance of v and ℓ_3. So we need to decide

$$\left(\frac{x_s}{z_s}\right)^2 + \left(\frac{y_s}{z_s}\right)^2 < \left(\frac{a_3\frac{x_s}{z_s} + b_3\frac{y_s}{z_s} + c_3}{D_3}\right)^2. \tag{2}$$

Inequality (2) is the algebraic formulation of the considered incircle test. It can be reformulated as

$$D_3^2((J+\alpha\sqrt{G})^2 + (I - \alpha\,\mathrm{sign}(rs)\sqrt{F})^2) < \left(a_3(J+\alpha\sqrt{G}) + b_3(I+\alpha\,\mathrm{sign}(rs)\sqrt{F}) + c_3X\right)^2$$

After rearranging terms we get an expression of the form

$$\mathrm{A}_1 + \mathrm{A}_2\sqrt{N} < \mathrm{A}_3\sqrt{F} + \mathrm{A}_4\sqrt{G} + \sqrt{N}(\mathrm{A}_5\sqrt{F} + \mathrm{A}_6\sqrt{G}) \tag{3}$$

where A_1 is a $12k$, $\mathrm{A}_2, \mathrm{A}_3, \mathrm{A}_4$ are $8k$, and $\mathrm{A}_5, \mathrm{A}_6$ are $4k$-bit integers. $N = N_1N_2$ is a $8k$-bit integer. After squaring we get an expression of the form

$$\mathrm{A}_7 + \mathrm{A}_8\sqrt{N} < \mathrm{A}_9 + \mathrm{A}_{10}\sqrt{N} \tag{4}$$

since $\sqrt{FG}$ is an integer and

$$(A_3\sqrt{F} + A_4\sqrt{G})^2 = A_3^2F + A_4^2G + 2A_3A_4\sqrt{FG}$$
$$\left(\sqrt{N}(A_5\sqrt{F} + A_6\sqrt{G})\right)^2 = N(A_5^2F + A_6^2G + 2A_5A_6\sqrt{FG})$$
$$\sqrt{N}(A_3\sqrt{F}+A_4\sqrt{G})(A_5\sqrt{F}+A_6\sqrt{G}) = \sqrt{N}(A_3A_5F+(A_3A_6+A_4A_5)\sqrt{FG}+A_4A_6G)$$

Rearranging (4) and squaring again leads to an integer relation. This integer relation can be decided using roughly $48k$-bit integers. Note that we have to determine the sign of both sides of a relation before we square them, but this can be done analogously and requires less precision.

Our argumentation also implies a bound on the precision required for evaluating the test exactly with floating point arithmetic. It is sufficient to compute the first $48k + O(1)$ binary digits of both sides of relation (3):

Lemma 2. *Let l be the lefthand side and r be the righthand side of relation (3). Then either $l = r$ or there are $u, v \in \mathbb{Z}$ such that $2^u \geq |l|, |r|$ and $|l - r| \geq 2^{u-v}$ where $v = 48k + O(1)$.*

Proof. (sketch): We first observe that squaring affects the number of relevant bits only by an additive constant, i.e., if the difference of a^2 and b^2 shows up in the t-th most significant bit then the difference of a and b shows up in the $(t + 1)$-th significant bit.

Lemma 3. *Let $0 \leq a < b$ with $a^2, b^2 \leq 2^s$ and $b^2 - a^2 \geq 2^{s-t}$. Then $a < b \leq 2^{s/2}$ and $b - a \geq 2^{s/2-t-1}$.*

Proof. Clearly $a, b \leq 2^{s/2}$. Also $2^{s-t} \leq (b^2 - a^2) = (b-a)(b+a) \leq (b-a) \cdot 2^{s/2+1}$. □

In the case of relation (3) we obtained an integer relation after squaring twice. So $t = s = 48k+O(1)$ after the second squaring. Applying Lemma 3 twice proves Lemma 2. □

How good is this bound? Our bound is more than two orders of magnitude better than one implied by classical root separation results. The *measure* of an algebraic number α gives a simple bound on the precision that is sufficient to decide the sign of α, see [Mig92, Sec. 4.3]. For a polynomial P with complex roots $z_1, z_2, \ldots, z_d$ and leading coefficient c_d define the measure of P by

$$M(P) = |c_d| \prod_{i=1}^{d} \max(1, |z_i|).$$

The measure $M(\alpha)$ of an algebraic number α is the measure of its minimal polynomial over $\mathbb{Z}$. We have $M(\alpha) > |\alpha| > M(\alpha)^{-1}$. However, we don't know a minimal polynomial for the algebraic numbers whose sign we are interested in. We only have a formula involving sums (and products) of radicals. The following estimates can be used to compute a bound on the degree and the measure of an algebraic number, where the degree $\deg(\alpha)$ of an algebraic number α is the

degree of its minimal polynomial. Let β and β' be algebraic numbers of degrees d and d' respectively.

$$\begin{aligned} \mathrm{M}(\beta \cdot \beta') &\leq \mathrm{M}(\beta)^{d'} \cdot \mathrm{M}(\beta')^{d} & \deg(\beta \cdot \beta') &\leq \deg(\beta)^{d'} \cdot \deg(\beta')^{d} \\ \mathrm{M}(\beta + \beta') &\leq 2^{dd'} \mathrm{M}(\beta)^{d'} \cdot \mathrm{M}(\beta')^{d} & \deg(\beta + \beta') &\leq \deg(\beta)^{d'} \cdot \deg(\beta')^{d} \\ \mathrm{M}(\sqrt[k]{\beta}) &\leq \mathrm{M}(\beta) & \deg(\sqrt[k]{\beta}) &\leq k \cdot \deg(\beta) \end{aligned}$$

For $p/q \in \mathbb{Q}$, $p, q \in \mathbb{Z}$, p, q relative prime, $\mathrm{M}(p/q) = pq$ and $\deg(p/q) = 1$. Using the estimates above we can compute a bound on the measure of an algebraic number given as an arithmetic expression involving roots. Direct application of the estimates above to the expression $\mathrm{A}_1+\mathrm{A}_2\sqrt{N}+\mathrm{A}_3\sqrt{F}+\mathrm{A}_4\sqrt{G}+\sqrt{N}(\mathrm{A}_5\sqrt{F}+\mathrm{A}_6\sqrt{G})$ of relation (3) gives a bound on its measure of $2^{86016k+1024}$. Rearranging the expression to $\mathrm{A}_1 + \mathrm{A}_2\sqrt{N} + \sqrt{F}(\mathrm{A}_3 + \mathrm{A}_5\sqrt{N}) + \sqrt{G}(\mathrm{A}_4 + \mathrm{A}_6\sqrt{N})$ leads to a "better" bound. of $2^{9216k+1024}$, i.e., roughly 9000-fold precision is sufficient to decide relation (3). It is likely that this bound can be improved using arguments of Dube and Yap [DY94]. In contrast our bound is $48k + O(1)$.

Can our bound be improved further? There is some indication that it can. Our argumentation applied to the incircle test for four points yields a precision bound of $20k$. However, this particular test can be done by lifting the points onto a paraboloid of revolution and checking the orientation of the lifted points. This approach requires only precision $8k$.

The derivation of the precision bound directly translates into an exact algorithm for the incircle test (3). We only have to use a BIGNUM package and to go through the steps of squaring and rearranging sums outlined above. We report on our experiences with that method below.

An alternative method to implement the incircle test is to compute the first $48k+O(1)$ binary digits of both sides of relation (3) using a BIGFLOAT package. It is reasonable to expect that if the two sides of (3) are distinct then much smaller precision will suffice to detect that fact. We report on some experimental evidence below.

However, if the sides of (3) are equal the full $48k$-bit precision is required to detect that fact. Blömer has shown that testing whether a sum of radicals is zero can be done in polynomial time [Bl91, Bl93]. A similar result for determining the sign of such a sum is not known. This indicates that testing for equality might be easier then determining the sign and that it could be wise to test an expression involving algebraic numbers for zero before a more expensive computation of its sign. The algorithm of Blömer can be applied in our case. Applying his algorithm directly does not give an improvement. Corollary 6.4 in [Bl93] implies that all roots appearing in relation (3) have to be approximated to precision $112k$. We already know that precision $48k$ suffices to decide relation (3). However, using a distinct and more careful analysis we can show that the equality test in relation (3) can be performed by a computation involving only $12k$-bit integers. Details will be given in the full paper.

3 Experimental Results

We have implemented two algorithms RS and FF for the incircle test of Equation (3) in C++.

Algorithm RS uses exact integer arithmetic provided by a BIGNUM package. In order to reduce the overhead of this BIGNUM package we first try to decide relation (3) by computing with doubles. For each appearing type of algebraic numbers we introduced appropriate C++ classes and overloaded the operations $+, -, *$. In addition, we had routines that compute the sign of these numbers by repeated rearranging, sign testing and squaring of the involved expressions.

Algorithm FF is based on a BIGFLOAT package. An element of the corresponding class bigfloat is a floating point number whose maximal number of bits can be chosen by the user. The operations $+, -, *, \sqrt{}$ are carried out within this chosen precision. In order to decide whether the BIGFLOAT evaluation of inequality (3) with precision p leads to a reliable sign test, we used static error analysis similar to the error analysis used in [FvW93] and [MN]: The absolute error in a BIGFLOAT evaluation of an expression E was estimated by

$$Ind(E) * Max(E) * 2^{-p} \tag{5}$$

where $Max(E)$ is an a priori bound on the value of E, p the current precision and $Ind(E)$ a (small) constant. Our error analysis implies that precision $2k+3$ is always required. We started algorithm FF with this precision and then increased the precision to $3k, 4k, \ldots$ until the sign was reliable.

Experimental Results: We tested the implementations with three types of input data: easy data, realistic data, and difficult data. The endpoints of the line segments were given by their cartesian coordinates (x, y) where x, y are k–bit integers.

- Easy data: We considered $k = 10$, 20, and 30. Point sites and segment endpoints defining the lines involved were chosen randomly and uniformly.
- Realistic data: We used polygonal environments that have been created as examples for a motion planning algorithm that moves a disc along the Voronoi diagram of the polygonal environment, see Figures 2–13. In all examples $k = 10$.
  We run Seel's implementation of the randomized incremental construction of Klein et al. [KMM90] on these examples and kept a log of all incircle tests performed. The log-file was then used as the input file for algorithms RS and FF.
- Difficult data: We considered again $k = 10$, 20, 30. In all these tests, relation (3) was satisfied with equality so that we always needed full precision.

Let us first consider the easy data (Table 1). The items in the columns $p = c \cdot k$ are the numbers of tests which could be decided with precision $p = c \cdot k$ but not with precision $(c-1) \cdot k$. Although algorithm FF required very low precision, algorithm RS was faster, since it could make all decisions using doubles and never needed BIGNUMs.

		Precision p				*Time*	
k	#tests	$2k+3$	$3k$	$4k$	$5k$	RS	FF
10	10000	2781	6801	398	20	26	80
20	10000	9962	38			25	38
30	10000	all				26	70

Table 1. Easy data: number of tests decided with precision p (in algorithm FF) and running times of both algorithms in seconds (on a SUN SPARCstation 10)

		Precision p							*Time*	
file	#tests	$\leq 3k$	$4k$	$5k$	$6k$	$7k$	$8k$	$9k$	RS	FF
B1	387	8	122	136	75	9			3.9	2.5
B2	398	45	204	134	12	3			2.3	1.9
B3	918	10	224	620	64				6.3	7.8
B4	1126	35	449	628	14				6.6	6.8
B5	4149		125	2310	1697	17			47	44
B6	13021		23	1266	10376	1310	42	4	141	138
B7	140	25	80	35					0.73	0.95
B8	276	3	63	197	13				2.1	2.4
B9	376		17	191	143	13	12		4.0	3.8
B10	414		46	235	119	14			4.3	3.8
B11	912	11	21	217	641	18	4		11.5	8.9
B12	385		2	7	341	30	5		4.9	15.2

Table 2. Realistic data: number of tests decided with precision p (in algorithm FF) and running times of both algorithms in seconds (on a SUN SPARCstation 10)

The results for the realistic data are shown in Table 2. The last 6 examples were specifically created to contain near-degeneracies. These examples used higher precision on average, but the required precision is quite low: precision $9k$ was sufficient for all of our tests. If we take into account that the points were not given in homogeneous form (i.e. their z-coordinate was 1), the general precision bound derived in Section 2 is $32k$ instead of $48k$. This is still much more than we actually needed. Furthermore the running times of algorithms FF and RS are comparable.

The difficult data (Table 3) show the worst case behaviour of the algorithms. Algorithm FF makes a lot of lower precision computations before it reaches the maximal precision.

Conclusion. In practice all our computations for non–degenerate inputs could be made with very moderate precision. Both the approach using exact integer arithmetic and the approach using BIGFLOATs have their advantages. The BIGFLOAT solution is more flexible and is supposed to work for other problems

k	#tests	*Time* RS	*Time* FF
10	1000	11	40
20	1000	12	60
30	1000	14	97

Table 3. Difficult data: running times of both algorithms in seconds (on a SUN SPARCstation 10)

as well while the effort for the adaptation of the symbolic computation machinery is higher. On the other hand, the integer solution is better suited for degenerate inputs.

References

[Aur91] F. Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, 1991.

[BDS+90] J.D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. Technical report, INRIA, 1990.

[BDS+92] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete and Computational Geometry*, 8:51–71, 1992.

[BJMM93] M.O. Benouamer, P. Jaillon, D. Michelucci, and J-M. Moreau. A "lazy" solution to imprecision in computational geometry. In *5th Canadian Conf. on Computational Geometry*, pages 73–78, 1993.

[Bl91] J. Blömer. Computing sums of radicals in polynomial time. In *FOCS91*, pages 670–677, 1991.

[Bl93] J. Blömer. Computing sums of radicals in polynomial time. Technical Report B93-13, Freie Universität Berlin, 1993.

[But94] B. Butz. Robuste Implementierung eines Algorithmus zur Berechung eines Voronoi-Diagrams für Polygone. Diplomarbeit, 1994.

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

[DY94] T. Dube and C.K. Yap. A basis for implementing exact computational geometry. extended abstract, 1994.

[For87] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[For92] S. Fortune. Numerical stability of algorithms for 2d Delaunay triangulations and Voronoi diagrams. In *ACM Symposium on Computational Geometry*, volume 8, pages 83–92, 1992.

[Fv93] S. Fortune and C. van Wyk. Efficient exact arithmetic for computational geometry. *Proc. of the 9th Symp. on Computational Geometry*, pages 163–171, 1993.

[FvW93] S. Fortune and C. van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. of the 9th ACM Symp. on Computational Geometry*, pages 163–172, 1993.

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

[KMM90] R. Klein, K. Mehlhorn, and S. Meiser. On the construction of abstract Voronoi diagrams ii. In *Proc. SIGAL Symp. on Algorithms*, Tokyo, 1990. Springer Verlag. LNCS 450.

[LM92] Z. Li and V. Milenkovic. Constructing strongly convex hulls using exact or rounded arithmetic. *Algorithmica*, 8:345–364, 1992.

[Loo82] R. Loos. Computing in algebraic extensions. In B. Buchberger, G.E. Collins, and R. Loos, editors, *Computer Algebra*, pages 173–187. Springer Verlag, 1982.

[Mig92] M. Mignotte. *Mathematics for Computer Algebra*. Springer Verlag, 1992.

[MN] K. Mehlhorn and S. Näher. Implementation of a sweep line algorithm for the segment intersection problem. manuscript.

[OBK92] A. Okabe, B. Boots, and Sugihara K. *Spatial tessellations: concepts and applications of Voronoi diagrams*. Wiley, New York, 1992.

[OTU87] T. Ottmann, G. Thiemt, and C. Ullrich. Numerical stability of geometric algorithms. In *Proc. of the 3rd ACM Symp. on Computational Geometry*, pages 119–125, 1987.

[OY85] C. O'Dúnlaing and C. Yap. A "retraction" method for planning the motion of a disk. *Journal of Algorithms*, 6:104–111, 1985.

[See94] M. Seel. Ausarbeitung und Implementierung eines Algorithmus zur Konstruktion abstrakter Voronoi-Diagramme. Diplomarbeit, 1994.

[SOI90] K. Sugihara, Y. Ooishi, and T. Imai. Topology-oriented approach to robustness and its applications to several Voronoi-diagram algorithms. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 36–39, 1990.

[Yap87] C. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete and Computational Geometry*, 2:365–393, 1987.

[Yap93] C.K. Yap. Towards exact geometric computation. In *5th Canadian Conf. on Computational Geometry*, pages 405–419, 1993.

[Yap94] C.K. Yap. *Fundamental Problems in Algorithmic Algebra*. Princeton University Press, 1994.

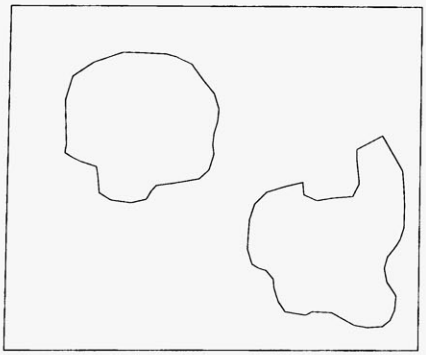

Fig. 2. B1

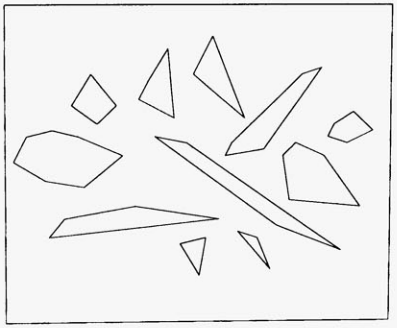

Fig. 3. B2

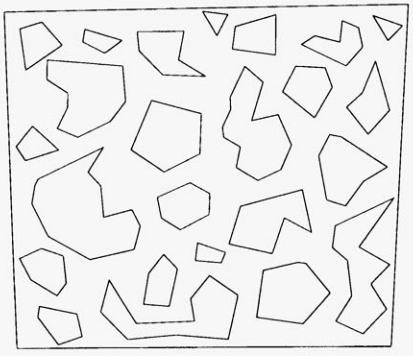

Fig. 4. B3

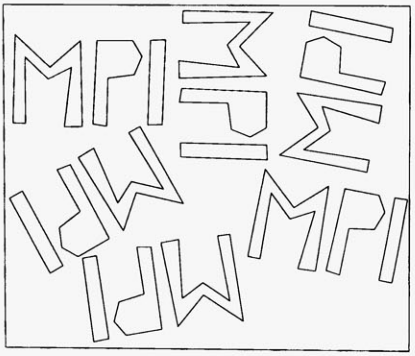

Fig. 5. B4

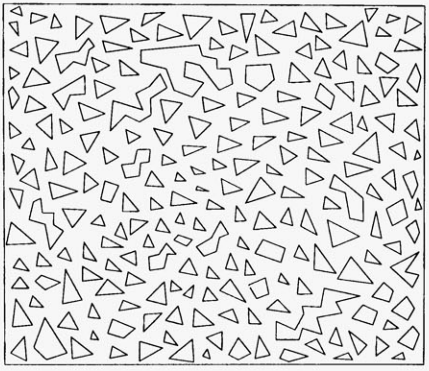

Fig. 6. B5

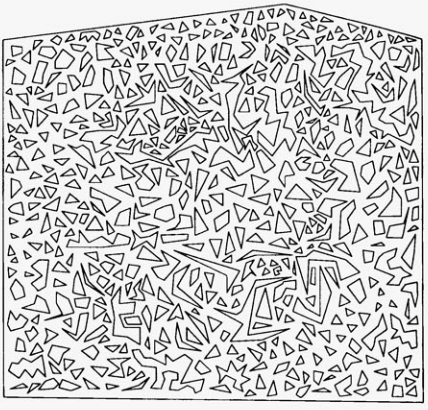

Fig. 7. B6

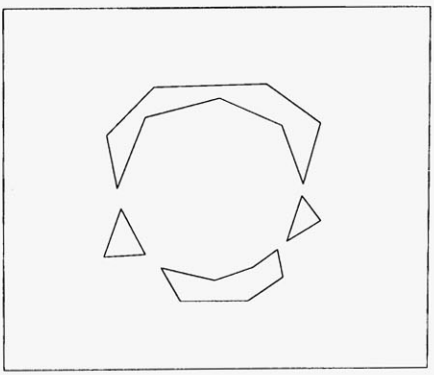

Fig. 8. B7

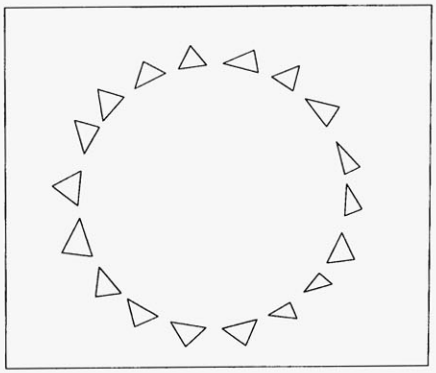

Fig. 9. B8

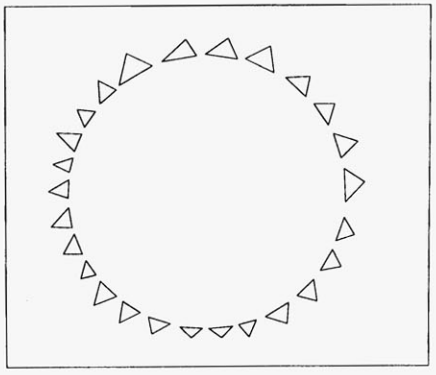

Fig. 10. B9

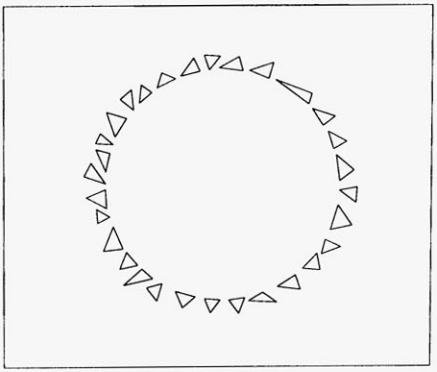

Fig. 11. B10

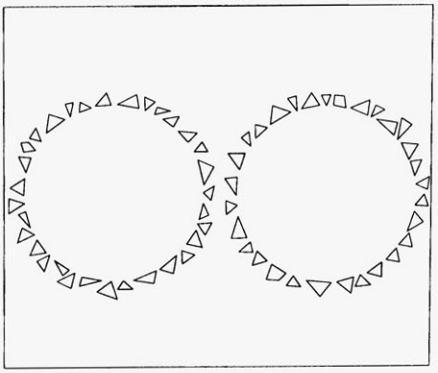

Fig. 12. B11

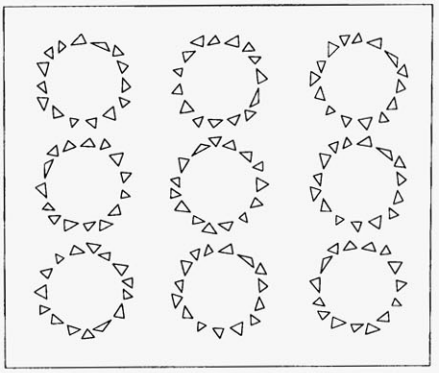

Fig. 13. B12