A quadratic non-standard arithmetic

Dominique Michelucci
École des Mines, F-42023 Saint-Étienne 02
micheluc@emse.fr

1 Introduction

This extended abstract presents an exact real quadratic arithmetic which also handles infinitely small numbers. The quadratic arithmetic provides the same operations than an exact rational one, plus square root of non negative numbers. It computes in the real quadratic closure of $\mathbb{Q}$, noted here: $\bar{\mathbb{Q}}$. As for an example, such an arithmetic can be used to compute the $2D$ arrangement of a set of circles and lines, or the $3D$ arrangement of a set of spheres and planes. The quadratic arithmetic is classic and presented in section 2. The new part is the way infinitely small numbers are managed in the quadratic framework.

In Computational Geometry, infinitely small numbers are typically used to symbolically perturb input parameters [2, 11, 10, 3, 7]: this infinitesimal perturbation removes accidental dependencies between data, and thus eliminates geometric degeneracy (alignment of more than two points, cocircularity of more than three points, etc) so that programmers have to handle only the few generic cases. The more often, people put forward efficiency arguments to restrict used perturbation schemes: most of the time, the perturbation is valid only for a single geometric predicate (say the InCircle predicate), and infinitesimals are not explicitly represented inside the computer. As a consequence, geometric predicates must concern input parameters only, and not derived values; moreover the user has poor control, if any, on the perturbation, typically seen as a black box. In opposition, the arithmetic in this extended abstract enables the user to compute with infinitesimals like if they were ordinary numbers in $\bar{\mathbb{Q}}$. Thus this arithmetic can manage in a straightforward way all proposed perturbation schemes (assuming the quadratic framework is sufficient, of course). Here are some of them:

- In Yap's perturbation [11, 10, 7], each input parameter x_i is perturbed into $x_i + \epsilon_i$. Thus $f(x + \epsilon) = \sum_\alpha \frac{1}{\alpha!}\epsilon^\alpha f^{(\alpha)}(x)$ by Taylor's formula. A consistent ordering between all power products $\epsilon^\alpha = \epsilon_1^{\alpha_1}\epsilon_2^{\alpha_2}\ldots$ sorts derivatives in decreasing magnitude: the sign of $f(x+\epsilon)$ is the one of the first non vanishing term in that sequence. Actually, ϵs are only implicit in Yap's perturbation.
- Each input parameter x_i is perturbed into $x_i + g_i\epsilon$, where g_i describe a generic situation, in Emiris and Canny's work [3]. Then $x_i + g_i\epsilon$ is a generic configuration: a single ϵ is sufficient.
- Each input parameter x_i is perturbed into $x_i + a_i\epsilon$, where a_i is random. The idea is that random a_i are very probably generic [3]. That is a probabilist perturbation scheme.
- See quoted references for other perturbation schemes.

This quadratic arithmetic cannot guarantee that the used perturbation scheme is valid, *ie* really removes all degeneracy: that obviously remains the responsibility of the CGer. But it detects persistent degeneracy at run time.

2 A Quadratic Arithmetic

2.1 Towers of extensions

The material in this section is not new [6], it is here only for completeness. In Fortune's method [4], one has to compare numbers of the form $\frac{a+\sqrt{b}}{c}$, where a, b, c are integers. It is possible to use repeated squaring, for this restricted case. This section presents a more general *real quadratic arithmetic*, which provides exact comparisons and operations: $+$, $-$, $\div$, $\times$ and $\sqrt{}$ on non negative numbers, starting from $\mathbb{Q}$. This arithmetic is an alternative to Yap and Dubé's one [1, 9]. Lack of space does not permit a comparison.

The idea is to compute in a tower of $\mathbb{R}$eal quadratic extensions $K_0 = \mathbb{Q}, \ldots K_i = K_{i-1}(\sqrt{\alpha_{i-1}})$ where K_i is an algebraic (and quadratic) extension over K_{i-1}, and $\alpha_{i-1} \in K_{i-1}$ is $\mathbb{R}$eal and positive and has no square roots in K_{i-1}. It means $K_i = K_{i-1}(\sqrt{\alpha_{i-1}})$ is the set of the numbers $u + v\sqrt{\alpha_{i-1}}$, with u and v two elements in K_{i-1}: in other words, numbers in K_i are represented by a vector of two components: u, $v \in K_{i-1}$, and K_i is represented by $\alpha_{i-1} \in K_{i-1}$ (which we already know how to represent, by induction) and by some reference to K_{i-1}. Operations in K_i straightforwardly reduce to operations in K_{i-1}:

$$\begin{aligned}
&(u + v\sqrt{\alpha_{i-1}}) + (u' + v'\sqrt{\alpha_{i-1}}) = (u + u') + (v + v')\sqrt{\alpha_{i-1}}\\
&(u + v\sqrt{\alpha_{i-1}}) \times (u' + v' \times \sqrt{\alpha_{i-1}}) = (u \times u' + v \times v' \times \alpha_{i-1}) + (u \times v' + u' \times v)\sqrt{\alpha_{i-1}}\\
&-(u + v\sqrt{\alpha_{i-1}}) = (-u) + (-v)\sqrt{\alpha_{i-1}}\\
&1/(u + v\sqrt{\alpha_{i-1}}) = (u/[u^2 - \alpha_{i-1} \times v^2]) - (v/[u^2 - \alpha_{i-1} \times v^2])\sqrt{\alpha_{i-1}}
\end{aligned}$$

Computing the sign of $w = u + v\sqrt{\alpha_{i-1}} \in K_i$ also boils down to computations in K_{i-1}:

$$\begin{aligned}
&u = 0 \text{ or } v = 0 : \text{trivial}\\
&u > 0 \text{ and } v \geq 0 \Rightarrow w > 0\\
&u > 0 \text{ and } v < 0 \Rightarrow \text{sign}(w) = \text{sign}(u^2 - v^2\alpha_{i-1})\\
&u < 0 \Rightarrow \text{sign}(w) = -\text{sign}(-w)
\end{aligned}$$

and in the end $K_0 = \mathbb{Q}$, where we know how to compute a sign, so the recursion eventually stops.

The last required operation is the square root in K_i. Assume $w = u + v\sqrt{\alpha_{i-1}} \in K_i$ is positive. The first thing is to test if w is a square in K_i, say the square of $z \in K_i$ with $z = x + y\sqrt{\alpha_{i-1}} > 0$ with $x, y \in K_{i-1}$. We suppose u and v do not vanish, because this case trivially reduces to the same problem in K_{i-1}.

$$\begin{aligned}
&w = u + v\sqrt{\alpha_{i-1}} = (x + y\sqrt{\alpha_{i-1}})^2\\
&\Leftrightarrow u = x^2 + \alpha_{i-1} \times v^2 \text{ and } v = 2 \times x \times y\\
&\Leftrightarrow x^2 = \frac{1}{2}\left[u \pm \sqrt{u^2 - \alpha_{i-1} \times v^2}\right] \text{ and } v = 2 \times x \times y
\end{aligned}$$

Thus $w \in K_i$ is a square in K_i iff $u^2 - \alpha_{i-1} \times v^2$ is a square in K_{i-1} and if $\frac{1}{2}\left[u + \sqrt{u^2 - \alpha_{i-1} \times v^2}\right]$ or $\frac{1}{2}\left[u - \sqrt{u^2 - \alpha_{i-1} \times v^2}\right]$ is a square in K_{i-1} (Note that they cannot be both squares in K_{i-1} because their product: $\frac{\alpha_{i-1}v^2}{4}$ is not a square in K_{i-1}).

Thus testing if $w \in K_i$ is a square in K_i reduces to computations in K_{i-1}: in the end, testing if $w \in K_0 = \mathbb{Q}$ is a square in $\mathbb{Q}$ is trivial. If w is a square in K_i, the method also gives its positive square root $x + y\sqrt{\alpha_{i-1}}$. When $w \in K_i$ is not a square in K_i, we have to define the quadratic extension of K_i which contains the square root of w: call this extension $K_{i+1} = K_i(\sqrt{w})$. In particular, the coordinates of $\sqrt{w}$ in K_{i+1} are: $(0 \in K_i, 1 \in K_i)$.

2.2 Using dags

Using a single tower of quadratic extensions to perform all computations results in very poor performance, since the first required operation is to express all met numbers (even integers or rationals) in the higher field. Thus the time required for a new computation increases with the number of previously performed computations, even when they are independent. It is clearly unacceptable. A possible solution is as follows.

Numbers are first represented by expressions, or dags (Directed Acyclic Graphs), like in the lazy rational arithmetic [8] or Dubé and Yap's arithmetic. A dag may be: a usual rational number, the sum or the product of two other dags, the opposite, the reciprocal or the square root of another dag. Each dag is associated with an enclosing interval, computed with arithmetic interval. When the interval becomes insufficient, the dag at hand is evaluated in an exact way, *starting from an empty tower,* ie *a tower containing only* $\mathbb{Q}$. This way, the time needed to exactly evaluate a dag depends only on the complexity of the dag itself (and of its subdags of course, but not on previously evaluated dags). Moreover, only inevitable exact computations are performed.

This optimization exploits the fact that CG typically deals with a lot of little computation trees, instead of a big one or a few big ones like in Symbolic Computing. Actually, the depth of computation trees met in classical non re-entrant CG methods (Convex Hulls, say) can even be known a priori: it is a constant for CG methods working in 2D or 3D, and a linear function of the underlying space dimension for CG methods working in $\mathbb{R}^n$ (actually $\mathbb{Q}^n$ or $\bar{\mathbb{Q}}^n$).

2.3 Computing characteristic polynomials

If need be, the characteristic polynomial of $z \in \bar{\mathbb{Q}}$ can be computed by the following method. Assume the characteristic polynomial of z is known in some extension $K(\alpha = \sqrt{A})$: it is $f(z) = \sum_{i=0}^{d} f_i z^i = 0$, with $f_i = a_i + b_i\alpha$, where a_i and b_i are in K. Then the equation of z in K is obtained with:

$$g(z) = \left[\sum_{i=0}^{d} a_i z^i\right]^2 - A\left[\sum_{i=0}^{d} b_i z^i\right]^2 = 0$$

and has degree twice f degree. Now an initial characteristic polynomial of z is easily computed in the highest extension of the tower: the polynomial of $z = x + y\sqrt{A}$ is $(z-x)^2 - y^2 A = 0$.

Another possible method gives a companion matrix, the characteristic polynomial of which is the one of z. Then interpolation methods (e.g. Lagrange) can be used to recover the characteristic polynomial of the matrix. Let $z = x + y\alpha \in K(\alpha = \sqrt{A})$, with $x, y \in K$. Multiplication of any number $a + b\alpha \in K(\alpha)$ and z can be seen as a linear transformation of vector $(a\ b)$ by the companion matrix:

$$M(z) = \begin{pmatrix} x & y \\ yA & x \end{pmatrix}$$

More generally, each number $z \in K(\alpha)$ can be represented by $M(z)$. In particular, the characteristic equation of $M(z)$ is the one of z. Recursive replacements of $M(z)$ entries by other matrices eventually reaches a matrix with rational coefficients. For instance, let $z = x + y\sqrt{3}$ with $x = a + b\sqrt{2}$ and $y = c + d\sqrt{2}$. Then z is represented by companion matrix

$$M(z) = \begin{pmatrix} x & y \\ 3y & x \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} a & b \\ 2b & a \end{pmatrix} & \begin{pmatrix} c & d \\ 2d & c \end{pmatrix} \\ 3\begin{pmatrix} c & d \\ 2d & c \end{pmatrix} & \begin{pmatrix} a & b \\ 2b & a \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ 2b & a & 2d & c \\ 3c & 3d & a & b \\ 6d & 3c & 2b & a \end{pmatrix}$$

3 Computing with infinitely small numbers

Let $\mathbb{G}$ be a *computable* real quadratic field. Initially $\mathbb{G} = \bar{\mathbb{Q}}$, which the previous section has shown to be computable, *ie* we can perform additions, multiplications, opposites, reciprocals, comparisons, square roots of non negative numbers in $\mathbb{G}$. We introduce a new and infinitely small number, namely ϵ, which is smaller than all positive numbers in $\mathbb{G}$. This section shows that $\mathbb{G}(\epsilon)$ is also a *computable* real quadratic field. Then $\mathbb{G}(\epsilon)$ can itself be used as a new background field for another non standard extension, with another new infinitely small number, namely ϵ', which is smaller than all positive numbers in $\mathbb{G}(\epsilon)$. This way it is possible to compute in a tower of non standard extensions: that may be useful when the CG problem at hand does not admit a perturbation scheme with a single ϵ. Recall if $g = (g_1, g_2 \dots g_n)$ is a generic input of the same problem, whose input parameters are: $p = (p_1, p_2 \dots p_n)$, then $p + \epsilon g$ is generic; but it may be a difficult realizability question to find such a generic configuration for some problem, like for instance finding a polytope with a prescribed topology.

3.1 The non quadratic case

In the simplest case, square roots and divisions are not used. Numbers reachable from $\mathbb{G}$ and ϵ are thus polynomials in ϵ: $P(\epsilon) = a_0 + a_1\epsilon + \dots + a_n\epsilon^n$, where $a_i \in \mathbb{G}$. The sign of $P(\epsilon)$ is clearly the coefficient sign of its first non vanishing term. The naive method computes all coefficients of polynomials, using standard symbolic computations. A more clever method represents polynomials with (not inevitably minimal) dags, in the now usual way, and exploits laziness and Taylor's formula:

$$P(\epsilon) = P(0) + \epsilon P'(0) + \frac{\epsilon^2}{2!}P''(0) + \dots \frac{\epsilon^d}{d!}P^{(d)}(0)$$

The sign of $P(\epsilon)$ is the one of the first non vanishing term in the sequence: $P(0)$, $P'(0)$, $P''(0) \dots P^{(d)}(0)$, where d is P degree (or an upper bound). When all coefficients vanish, $P(\epsilon)$ is identically zero and the sign is zero. When polynomial P is represented by some dag, it is easy to generate on-the-fly a dag for its derivative P' (and then for P'', P''', etc), using standard derivation rules. Successive derivations of a dag eventually reach a dag without occurrence of ϵ, possibly at the cost of some supplementary derivation. For instance the dag: $d_0(\epsilon) = (\epsilon - \epsilon)$ is a non minimal dag representing the null polynomial. It vanishes in $\epsilon = 0$; it has derivative: $d_1 = (1 - 1)$ which also vanishes in 0. Since d_1 contains no more occurrence of ϵ, d_1 and d_0 are proven to be identically zero.

To summarize: if a dag $f(\epsilon)$ does not contain ϵ, its sign is trivially computed in $\mathbb{G}$. Otherwise, when $f(0)$ does not vanish, its sign gives the sign of $f(\epsilon)$. Otherwise the sign of $f(\epsilon)$ is the one of $f'(\epsilon)$. This process always ends. Note dags are evaluated in $\epsilon = 0$, thus no really symbolic computations (like polynomial gcd or resultants) are performed on polynomials, except dag derivation and construction.

This method can be seen as an extension of Yap's perturbation, whose limitations are overcome: the user has full control on the perturbation (recall Yap's perturbation systematically and implicitly perturbs the ith input parameter p_i into $p_i + \epsilon_i$): it is easy to choose the sign of the perturbation or to force a perturbed point to stay on a given half straight line for instance. Moreover the programmer is relieved of the burden of expressing all tests with input parameters only: dag handling does the job. It is true that towers of non standard extensions: $\left(\bar{\mathbb{Q}}(\epsilon)\right)(\epsilon')$... implicitly sort the set of power products in $\epsilon, \epsilon' \dots$ with reverse lexicographic order, *ie* :

$$1 >> \epsilon >> \epsilon^2 >> \epsilon^3 \dots >> \epsilon' >> \epsilon\epsilon' >> \epsilon^2\epsilon' >> \epsilon^3\epsilon' \dots >> \epsilon'^2 >> \epsilon\epsilon'^2 >> \epsilon^2\epsilon'^2 >> \epsilon^3\epsilon'^2 \dots > 0$$

though the user may prefer another *consistent ordering* (also called: *compatible ordering*), say for instance total degree ordering. But the user can easily obtain any other compatible order [7], for instance using: $\alpha = \epsilon'$ and $\alpha' = \epsilon\epsilon'$, he gets α and α' power products sorted in total then reverse lexicographic order, *ie* :

$$1 >> \alpha >> \alpha' >> \alpha^2 >> \alpha\alpha' >> \alpha'^2 >> \alpha^3 >> \alpha^2\alpha' >> \alpha\alpha'^2 >> \alpha'^3 \dots > 0$$

3.2 Quadratic case

Unfortunately, the previous solution doesn't extend with divisions and square roots: some derivatives can be undefined in 0, like $F(\epsilon) = \sqrt{\epsilon}$ the derivative of which is $F'(\epsilon) = \frac{1}{2\sqrt{\epsilon}}$. Moreover, successive derivations may never reach an expression without occurrence of ϵ. The proposed solution uses series in ϵ.

3.2.1 Using series in ϵ

Each element of $\mathbb{G}(\epsilon)$ is first represented by a dag: a dag can be a constant in $\mathbb{G}$, ϵ, the sum or the product of two other dags, the opposite or the reciprocal or the square root of another dag. Moreover, each dag is associated with a series $S(\epsilon)$, represented by a *factor* $k \in \mathbb{N}$, an *algebraic degree* $d \in \mathbb{N}$ (defined below), a shift power $p \in \mathbb{N}$, and the lazy (potentially infinite) list or array a_i of its coefficients in $\mathbb{G}$:

$$\delta = \epsilon^{\left(\frac{1}{2^k}\right)}, \ S(\epsilon) = \frac{a_0 + a_1\delta + a_2\delta^2 + \ldots}{\delta^p}$$

The dag makes possible the computation of coefficient a_i when needed. The factor k is needed because of expressions like $\sqrt{\epsilon}$. It is easy to convert a series to another one with a greater factor, thus we can suppose w.l.o.g. that series to be added or multiplied have the same factor. The shift power p typically equals 0, it is only used to avoid negative exponents, like in $\frac{1}{\epsilon} = \epsilon^{-1}$. It is ignored from now on. We give formulas for coefficients of sum, product, inverse and square root:

$(a_0 + a_1\delta + \ldots) + (b_0 + b_1\delta + \ldots) = x_0 + x_1\delta + \ldots$ where $x_k = a_k + b_k$

$(a_0 + a_1\delta + \ldots)(b_0 + b_1\delta + \ldots) = x_0 + x_1\delta + \ldots$ where $x_k = \sum_{i=0}^{i=k} a_i b_{k-i}$

$\frac{1}{a_0 + a_1\delta + \ldots} = x_0 + x_1\delta + \ldots$ where $x_0 = \frac{1}{a_0}$, $x_k = -\frac{1}{a_0}\sum_{i=1}^{k} a_i x_{k-i}$, assuming $a_0 \neq 0$

$\sqrt{a_0 + a_1\delta + \ldots} = x_0 + x_1\delta + \ldots$ where $x_0 = \sqrt{a_0}$, $x_k = \frac{1}{2x_0}\left(a_k - \sum_{i=1}^{i=k-1} x_i x_{k-i}\right)$ assuming $a_0 > 0$.

If $a_0 < 0$ there is no square root. If $a_0 = 0$, let $\delta_2 = \sqrt{\delta}$, then $\sqrt{S(\epsilon)} = \delta_2\sqrt{a_1 + a_2\delta_2^2 + a_3\delta_2^4 \ldots}$

3.2.2 A gap theorem for sign computation

The sign of a non identically null series is the sign of its first non vanishing coefficient. When the perturbation scheme really removes all degeneracy, series identically null cannot occur. However their detection is useful. It is made with the following *gap theorem*: a series $y = S(\epsilon)$ is identically zero when all its coefficients in terms $a_i\epsilon^i$ with $i \leq d$ are zero, where d is the *algebraic degree*, or an upper bound, of the series.

Let $y = f(\epsilon)$, where $f(\epsilon)$ is a dag in the variable ϵ, and let $S(\epsilon)$ be the corresponding series. ϵ and y fulfill a polynomial condition: $F(\epsilon, y) = 0$. In other words, the point (ϵ, y) lies on an algebraic curve. The degree of the dag f, and of the corresponding series, is the total degree of the polynomial F in ϵ and y: for instance, if $y = f(\epsilon) = \sqrt{1+\epsilon}$, then $F(\epsilon, y) = y^2 - (1+\epsilon) = 0$ has degree 2. Same degree for $y = f(\epsilon) = (1+\epsilon)^2$. d, the degree or an upper bound, is recursively computed like that: if the dag is ϵ or a constant in $\mathbb{G}$, then its degree is 1. If the dag is the sum or product of two other dags a and b, its degree is the product of the degrees for a and b. The degree of $-a$ and $1/a$ is the degree of dag a. The degree for $\sqrt{a}$ is twice the degree of dag a.

We sketch now the proof of the gap theorem. Assume for simplicity that the factor of the series is 0, *ie* the series is $y = S(\epsilon) = a_0 + a_1\epsilon + a_2\epsilon^2 + \ldots$. The coefficients for negative exponents are zero (otherwise, the series is clearly not 0!). The degree of the dag $y = f(\epsilon)$ is at most d. Now, $a_0 = a_1 \ldots = a_d = 0$. Thus the curve branch $y = s(\epsilon) = a_0 + a_1\epsilon + a_2\epsilon^2 + \ldots$ cuts the straight line $y = 0$ at the origin, with multiplicity $d+1$. But this branch is part of an algebraic curve $F(\epsilon, y) = 0$ with degree not greater than d. Thus the curve branch $y = f(\epsilon)$ is the straight line $y = 0$, and the series $S(\epsilon)$ is identically 0.

This argument extends as follows when the series factor is greater than 0. The dag $y = f(\epsilon)$ has always algebraic degree d but the corresponding series is now: $S(\epsilon) = a_0 + a_1[\epsilon^{\frac{1}{2^k}}]^1 + a_2[\epsilon^{\frac{1}{2^k}}]^2 + \ldots$. Let $\delta = \epsilon^{\frac{1}{2^k}} \Leftrightarrow \epsilon = \delta^{(2^k)}$ and $T(\delta) = a_0 + a_1\delta^1 + a_2\delta^2 + \ldots$. The dag $y = g(\delta) = f(\delta^{(2^k)})$ has algebraic degree $d \times 2^k$. From the previous gap theorem, when all coefficients for exponents $0, 1 \ldots, d \times 2^k$ of $T(\delta)$ vanish, then the series $y = T(\delta) = S(\epsilon)$ is identically zero. Conclude.

3.3 Limitations of perturbation methods

Though input parameters are perturbed according some perturbation scheme, computations may still yield to non generic situations. It is especially true, and annoying, with on-line or reentrant methods, where computed values may be re-used (for instance an intersection point becomes a vertex of a new edge). The most trivial example is as follows: the intersection point I between two perturbed segments AB and CD (whatever the perturbation) is exactly aligned with A and B, and with C and D! Alignment of more than 2 points is a degeneracy[1]. Note geometric applications in the real world involve edition of geometric objects, thus they need reentrant methods. These limitations of perturbation have not been seriously addressed so far.

4 Conclusion

This extended abstract has presented a quadratic arithmetic which also handles infinitely small numbers, like if they were ordinary numbers. The computation depth has not to be known or bounded *a priori* (contrarily to other CG exact arithmetics, like Fortune and Van Wyk's one [5]). Perturbations are handled using lazy series and a gap theorem to achieve termination. Exploiting laziness insures that only inevitable computations are performed. This arithmetic can be used with all existing perturbation schemes (compatible with a quadratic arithmetic). It detects at run time cases in which the perturbation scheme does not remove degeneracy. It does not add any overhead (except constant) to the intrinsic complexity of asked computations. Unfortunately, due to lack of space, this extended abstract couldn't compare it with other quadratic or algebraic arithmetics.

References

[1] T. Dubé and C. Yap. A basis for implementing exact geometric algorithms. *manuscript*, 1993.

[2] H. Edelsbrunner and E.P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph*, 9:66–104, 1990.

[3] I. Emiris and J. Canny. A general approach to removing degeneracies. *SIAM J. Computing*, 24(3):650–664, June 1995.

[4] S. Fortune. A sweep-line algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[5] S. Fortune and C. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proceedings of the 9th ACM Symposium on Computational Geometry*, pages 163–172, San Diego, May 1993.

[6] D. Michelucci. The robustness issue. *submitted to publication*, 1997.

[7] D. Michelucci. An epsilon-arithmetic for removing degeneracies. In *Proceedings of the IEEE 12th Symposium on Computer Arithmetic*, pages 230–237, Windsor, Ontario, July 1995.

[8] D. Michelucci and J-M. Moreau. Lazy arithmetic. *To be published in IEEE Transactions on Computers*, summer 1997. Available at: <ftp://ftp.emse.fr/pub/papers/LAZY/lazy.ps.gz>.

[9] K. Ouchi. Implementation of exact computation. *Masters Thesis, New York University*, 1997.

[10] C.K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *J. Comput. Syst. Sci.*, 40:2–18, 1990.

[11] C.K. Yap. Symbolic treatment of geometric degenaracies. *J. Symbolic Comput*, 10:349–370, 1990.

[1] A solution is here to perturb the intersection point with a new infinitesimal smaller than all already existing ones