Robustness and Randomness

Dominique Michelucci[1], Jean Michel Moreau[2], and Sebti Foufou[1]

[1] LE2I UMR CNRS 5158, UFR Sciences, Université de Bourgogne,
BP 47870, 21078 Dijon Cedex, France
dmichel@u-bourgogne.fr, sfoufou@u-bourgogne.fr
[2] LIRIS UMR 5205, Nautibus, Université Claude Bernard Lyon 1,
46 Bd du 11 Nov. 1918, 69622, Villeurbanne, France
Jean-Michel.Moreau@liris.univ-lyon1.fr

Abstract. Robustness problems of computational geometry algorithms is a topic that has been subject to intensive research efforts from both computer science and mathematics communities. Robustness problems are caused by the lack of precision in computations involving floating-point instead of real numbers. This paper reviews methods dealing with robustness and inaccuracy problems. It discusses approaches based on exact arithmetic, interval arithmetic and probabilistic methods. The paper investigates the possibility to use randomness at certain levels of reasoning to make geometric constructions more robust.

1 Introduction

Mastering the robustness of computational geometry algorithms (algorithms intended to solve geometric computing problems such as surfaces intersections, shortest paths on surfaces, planification of trajectories, etc.) is a topic that has attracted big attention from both computer science and mathematics communities. Robustness problems are caused by the lack of precision in computations involving floating-point instead of real numbers, in that case robust implementation of geometric algorithms is highly nontrivial and the strange behaviors (crashes, infinite loops, inconsistent outputs, etc.) of these algorithms are due to their inaccurate computations. Although a lot of work has been done to solve this problem, it is still impossible to find a systematic, simple and fast method that eliminates the sources of all these robustness problems.

Robustness and non-robustness issues in geometric computations has important scientific and economic impact (barrier to full automation, programmers' productivity, failure of critical missions, etc.). This impact has motivated intensive researches on the subject during the last twenty years, which generated a large literature and surveys [1–4]. We refer the reader to the excellent recent survey by C. Yap in the Handbook of Discrete and Computational Geometry [4]. In another good survey J. Keyser classified robustness problems in two main categories: problems due to precision and problems due to degeneracies [5]. He

Dagstuhl Seminar Proceedings 06021
Reliable Implementation of Real Number Algorithms: Theory and Practice
http://drops.dagstuhl.de/opus/volltexte/2006/716

presented the issues involved with each of these classes and discussed some of the solutions that have been proposed for dealing with them. In an earlier paper, D. Goldberg presented a tutorial on the aspects of floating-point that have a direct impact on designers of computer systems [1]. The paper begins with a background on floating-point representation and rounding error and continues with a discussion of the IEEE floating point standard. Robustness problems in computer aided design and geometric modelling has been studied by C. Hoffmann in [3] where exact arithmetic, symbolic reasoning, and reliable calculations (interval arithmetic) was identified as possible strategies to address this problem. Siguhara and Iri introduced the topology-based approach which avoids failure by using floating-point arithmetic, but places higher priority on topological consistency than on numerical values [6]. So decisions by this approach ensure that the result is always coherent from the topology point of view. But, there is no guarantee that any other software that works with such a result will give coherent outputs from it. Topology-oriented implementations have been applied to a number of geometric problems such as Voronoi diagram computations and convex polyhedra intersections [7, 8].

In this paper we survey some inaccuracy issues in computational geometry, we discuss the classical solutions that have been suggested in the last twenty years, and then show how randomness may sometimes be used in order to help reduce the impact of inaccuracy in geometric computations. The probabilistic approach has received less attention than other robustness tackling methods, our intention here is to highlight the positive role probabilistic algorithms can play. The use of randomness is not intended to solve all robustness problems of geometric algorithms, but to provide probabilistic algorithms as an alternative for geometric computations. These algorithms are costly in time, but tolerant and can resist to inaccuracies. They operate using weak oracles that very often converge to the true decision, but can also say "I do not know" when the right decision is out of reach.

The paper is organized as follow: Section 2 describes some of the problems caused by inaccuracy. Section 3 reviews the classical solutions that were designed to help prevent these inaccuracy problems. Section 4 explores new methods, based on probabilistic approaches used in various unrelated domains.

2 Consequences of inaccuracy on geometric algorithms

The inaccuracy of floating-point arithmetic has dramatic consequences on geometric computations. Inaccuracy causes inconsistencies both in geometric programs and their data structures, so that geometric programs may crash or yield inconsistent results.

As a first illustration, let S_n be a set of $n \geq 4$ points in the Euclidian plane, with no more than two on the same line. The convex hull $\mathcal{C}$ of S_n is the smallest convex polygon enclosing all its elements. A simple method to construct the

edges of $\mathcal{C}$ by enumeration is to identify all pairs (a, b) in S_n^2 such that all points in $S_n \backslash \{a, b\}$ lie on the same side of infinite line (ab) (note that imposing no more than two aligned points in S removes special cases here). Although this method is correct from a theoretical point of view, it may fail to yield consistent results in practice: to see this, consider applying it to $n = 4$ nearly aligned points. Because the points may be arbitrarily close to being aligned without being exactly so, the previous test may easily fail on any pair of points, due to inaccuracy in the computations! This failure may seem striking at first glance, but is it really more striking than the impossibility to verify identities such as: $(\sqrt{2})^2 = 2$ or $(1/3) \times 3 = 1$ or $(\cos\theta)^2 + (\sin\theta)^2 = 1$ with floating-point arithmetic?

To get more insight on what goes wrong in this example, consider a, b, c to be 3 distinct points in the Euclidian plane. The "orientation" of the three points, $\mathcal{O}(a, b, c)$ is defined as the sign of the determinant:

$$\begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix},$$

which represents the signed volume of the parallelepiped generated by the vectors $(a, 1)$, $(b, 1)$, $(c, 1)$. Intuitively, the three points form a left, right, or null "turn" depending on whether $\mathcal{O}(a, b, c)$ is positive, negative or null. Obviously, $\mathcal{O}(a, b, c)$ and $\mathcal{O}(c, b, a)$ must have opposite signs, but one may easily generate three almost (but not exactly) aligned distinct points a, b, c with floating-point coordinates, which contradict this property. To help solve such inconsistencies, D.E. Knuth [9] suggested a set of axioms fulfilled by the orientation predicates, assuming for the simplicity of proofs that no more than two data points may lie on the same line. He later realized that he had set up the axioms for oriented matroids with rank 3. All his axioms and resulting theorems are contradicted by floating-point configurations, due to inaccuracy.

Inaccuracy introduces inconsistencies in geometric data structures. Among the most basic geometric data structures, some represent point / line or point / plane incidences. Typically a 2D point is described by its cartesian coordinates (x, y), and a line with equation $ax + by + c = 0$ by the triple (a, b, c). The intersection point between 2 lines (a, b, c) and (a', b', c') is easily computed with standard linear algebra. Due to inaccuracy, $\Omega(x_\Omega, y_\Omega)$, the computed intersection point will not lie on these lines (*i.e.,* $ax_\Omega + by_\Omega + c \neq 0$ and $a'x_\Omega + b'y_\Omega + c' \neq 0$): a contradiction between a numerical test and the incidence fact stored in the data structure. Of course, most geometry programmers are aware of this difficulty, and hence, to find out the position of a vertex v relatively to a line D, they first check whether the data structure does not explicitly hold the information "v lies on D"; if not, a numerical orientation test is performed. This two-stage procedure eliminates the more obvious inconsistencies. However, many geometric theorems of projective geometry—or even geometric constructions—imply non-

trivial incidences, which such simple "precautions" cannot detect, as we shall now see through five well-known theorems from the field of classical geometry.

Theorem 1 (Harmonic conjugate, Fig. 1, left). *Let A, B, X be 3 distinct aligned points. Let L be any line through X, s any point outside L and ABX. Then the point X' defined by the construction: $a = sA \cap L$, $b = sB \cap L$, $s' = aB \cap Ab$, $X' = ss' \cap AB$, depends neither on L nor on s.*

X' is called the harmonic conjugate of X relatively to A, B. The harmonic conjugate of X' is X.

Theorem 2 (Desargues theorem, Fig. 1, right). *In 2D or 3D, if two triangles abc and ABC are such that aA, bB, cC concur (the two triangles are said to be "perspective"), then homologous sides meet in 3 aligned points,* i.e., *$ab \cap AB, bc \cap BC, ca \cap CA$ are collinear. The converse is true as well.*

Theorem 3 (Pappus theorem, Fig. 2, left). *In 2D (*i.e., *in the projective plane), if p_1, p_2, p_3 are three distinct aligned points, and if q_1, q_2, q_3 are three distinct aligned points, then the three intersection points $p_1q_2 \cap p_2q_1$, $p_1q_3 \cap p_3q_1$ and $p_2q_3 \cap p_3q_2$ are aligned as well.*

Theorem 4 (Pascal, Fig. 2, right). *6 coplanar points belong to the same conic if and only if the 3 opposite sides (in any order) meet in 3 aligned points.*

Theorem 5 (Pouzergues (hexamys)). *In the projective plane, an* hexamys *is a (possibly concave and self-intersecting) hexagon such that its three opposite sides meet in three aligned points. Then every permutation of an hexamys is also one.*

Pouzergues' theorem may be seen as a particular case of Pascal's theorem with no conic involved. All those geometric theorems are "wrong" when using the

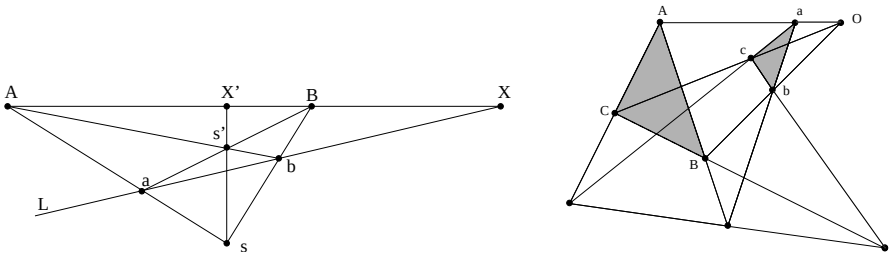

Fig. 1. Left (harmonic conjugates): for 3 given aligned points A, B, X, the point X' does not depend on L nor s. Right: Desargues theorem.

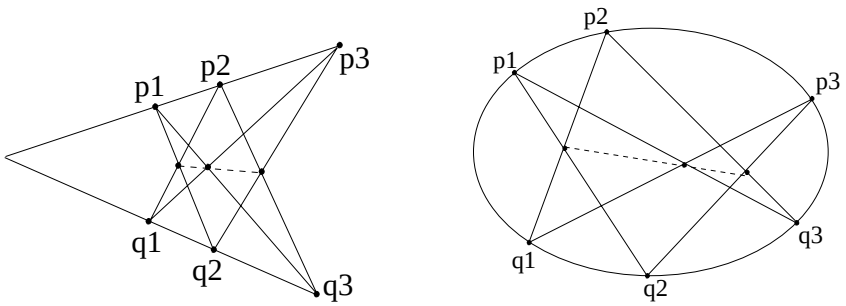

Fig. 2. Pappus' and Pascal's theorems.

floating-point arithmetic, and "true" when using an exact arithmetic. An exact rational arithmetic is sufficient to prove the harmonic conjugate theorem, Pouzergues', Pappus' and Desargues' theorems, assuming the initial coordinates are rational. An algebraic arithmetic is required for Pascal's theorem, if the points on the conic are intersection points between general conics.

3 Classical methods

3.1 The epsilon heuristic

To overcome inaccuracy, the most popular trick used in geometric modelers is the ϵ heuristic. When two floating-point numbers differ by less than a given threshold traditionally called ϵ, they are considered to be the equal. The test may be made in an absolute ($|a-b| < \epsilon$) or relative ($|a-b| < \epsilon \times max(|a|, |b|)$) manner. Some modelers use several ϵ values, say one for lengths, another for areas, another for angles, etc.

This heuristic loses the equality transitivity: it is easy to find a, b and c so that $a =_\epsilon b$, $b =_\epsilon c$, but $a \neq_\epsilon c$, with $=_\epsilon$ meaning "equal for the ϵ heuristic": thus inconsistencies remain possible.

Moreover, finding the relevant value(s) for ϵ(s) is much of a difficult task, depending on the usual range of numbers (itself depending on the applications), and on the format of floating-point numbers: it is common folklore in the CAD-CAM community that the conversion from 32-bits floating-point numbers to 64-bits has required a not so easy ϵ's updating. Of course the ϵ heuristic may fail, and sometimes it does. In practice, it seems to work not so bad and to improve the geometric modelers' robustness.

3.2 The exact computation paradigm

In geometric computations, a lot of effort has been put to design theoretically fast methods, assuming that an exact arithmetic is available for free. These sophisticated methods do not resist inaccuracy and the induced inconsistencies. They require consistency to work.

Exact rational arithmetics In the presence of alternatives and tests, geometric methods branch according to the (positive, negative or null) sign of expressions, called predicates: the orientation test of three points in the plane is a typical example. One method to prevent inconsistencies in computations is to take "exact" decisions in the branching tests, by means of an exact arithmetic. Systematically using such an arithmetic consumes too much time and space resources, and hence various authors have advocated for the use of some sort of 'filtering" (also known as "laziness"):

- First compute guaranteed bounds on the expressions to be tested. Most of the time, those are sufficient to determine the sign of the expression.
- Whenever they are not (*i.e.,* 0 lies within the bounds), use an exact arithmetic to determine the sign of the expression.

An example of such a method is the lazy rational arithmetic [10]: a lazy number is represented by an enclosing interval, and by a definition (either an initial rational number, or the sum, product, opposite, inverse of other lazy numbers). The interval is systematically computed. When it is not sufficient to decide the sign of a number, the definition associated with the number is evaluated using an exact rational arithmetic.

All filter-based solutions use the same basic scheme, they may differ by the way they define aspects such as:

- the exact arithmetic used (remainder number system, *i.e.,* modular arithmetic; strings of digits, ...),
- the evaluation strategy,
- the method for storing the exact values or the definition itself,
- the method for evaluating the bounds (statically at compile time, or dynamically at run time).

Such techniques, routinely used in major geometric applications, for instance CGAL [11], XSC [12], LOOK [13] or LEDA [14] libraries, are, unfortunately, limited to computations in the field of rational numbers. However, it is possible to generalize the lazy (or filter) paradigm if an exact algebraic arithmetic is available, as the "gap arithmetic" used in LEDA::real, CORE and CGAL [15, 16, 11].

Gap arithmetics Canny's gap theorem gives a way to *numerically* prove that a number *is* zero: compute a (guaranteed) interval containing it, with width smaller than ϵ_c [17]. As soon as the interval does not contain 0, the number is clearly not 0 and its sign is known. Otherwise, if the interval contains 0 and has width less than ϵ_c, the number can only be 0.

Theorem 6 (Canny's gap theorem). *Let x_1, $x_2 \dots x_n$ be the solutions of an algebraic system of n equations and n unknowns, having a finite number of solutions, with maximal total degree d, with relative integer coefficients smaller or equal to M in absolute value. Then, for all $i \in [1, n]$, either $x_i = 0$ or $|x_i| > \epsilon_c$ where $\epsilon_c = (3Md)^{-(nd^n)}$.*

Unfortunately, there are several problems. First, ϵ_c is far much smaller than the ϵ used in geometric modelers; actually ϵ_c is generally much smaller than the smallest positive floating-point number, even in simple examples, hence the need for some "big-float" arithmetic. Second, given such an arithmetic, the computational scheme described here has an exponential cost: an exponential number of digits is needed to prove the nullity of a number because of the nd^n term in Canny's theorem. There is no hope to significantly widen Canny's gap in the worst case, because it is almost reached in the following simple instance: $x_1(Mx_1 - 1) = 0$, $Mx_2 - x_1^2 = 0 \dots Mx_n - x_{n-1}^2 = 0$. See [18–20] for implementations of gap arithmetics or gap theorems, and [21, 22] for related root separation bounds.

Pros and cons of the exact computation paradigm CGAL geometric library relies on filters and lazy rational arithmetic to achieve robustness [11]. CGAL is well-known for its reliability and speed; its Delaunay routine is often used in industry for surface reconstruction from a set of sampling points. This alone stands as a good point of the paradigm. However, the exact approach has important limitations:

- Exact algebraic arithmetics are too slow to be practical. Unfortunately, algebraic numbers are ubiquitous in geometric computations: rotating an object by an angle $k\pi, k \in \mathbb{Q}$, intersecting conics or other algebraic curves, intersecting quadrics or other algebraic (parametric or implicit) surfaces, all those "primitives" introduce algebraic numbers. [23] implemented robust boolean operations between 3D algebraic shapes, but the corresponding program is an order of magnitude too slow.
- In applications such as CAD-CAM, computer graphics and GIS, data are inaccurate; it does not make sense to compute results which are more accurate than data itself; the only justification could be the attempt to make the most fragile algorithms "work"; moreover the exact results are typically rounded to communicate with the rest of the world, which uses only floating-point arithmetics.

– Industrial applications use the floating-point arithmetic to represent geometric object. Translations from exact representations to floating-point representations and vice-versa are thus essential. Another reason for such "rounding" is that geometric modelers are shape editors, and the algebraic complexity of the edited shape increases with each editing operation. Rounding floating-point geometries to exact (and consistent) geometries is as much difficult as "repairing" inconsistent geometric objects (a situation known as the "polygon soup").

When the cost of exact arithmetics is taken into account, some of these algorithms may become unpracticable, or slower than more rudimentary methods (see section 4), which are more robust and still work with inaccuracy, because they do not propagate inaccurate results.

3.3 Interval computations

A natural idea to rid geometrical computations of inconsistencies is to resort to some kind of interval computations.

Basic interval arithmetic The first and most basic interval arithmetic is due to R. Moore [24]. Basically, numbers are represented with intervals, which stand for the uncertainty associated with each one, a notion which was obviously borrowed from the everyday practice of physicists. Interval computations are defined for the sum, difference, product and inverse of intervals. Hence, it is possible to maintain intervals for combinations of, and even simple functions on, elementary data. It is also possible to define interval variants for the exponential ($\exp([a,b]) = [\exp(a), \exp(b)]$), logarithm, sine, cosine functions, etc. For non-monotonous functions, the interval argument must be decomposed into subintervals on which the function is monotonous.

Following the basic theory, the width of the sum or the difference of two intervals is the sum of the widths of the two added or subtracted intervals. Thus, $X - X$ is not equal to 0: interval arithmetic "loses the dependence between variables". A consequence of this is the wrapping effect: the overestimation of intervals increases with the number of operations.

To restrict such a wrapping effect when evaluating polynomials, a possibility is to use the central evaluation form, as follows: $f(X) \subset f(X_c) + (X - X_c)f'(X)$, where X_c is the center of the interval X; $X - X_c$ is the halfwidth of X; $f'(X)$ is an interval enclosing the derivative of f inside X; it is computed either with the naive arithmetic, or recursively with the central evaluation form. This extends to multivariate polynomials. The analytical definition of f is explicitly required, and may not be considered as an "oracle" or a black box by itself.

Affine interval arithmetic As mentioned just above, the naive interval arithmetic "loses the dependence between variables"; the affine interval arithmetic was intended to fix this flaw [25, 26]. It is designed as a model for "self-validated computations", that keeps track of the first-order correlations between computed and input quantities. Quantities manipulated by the arithmetic are represented by affine expressions of the form:

$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n,$$

where the ε_i are called the "noise symbols", and the x_i are real numbers. Each noise symbol stands for an independent component of the total uncertainty of the ideal quantity it is associated with. Standard operations are defined over those forms, and affine interval arithmetic may be be seen as a generalization of interval arithmetic with richer properties. In particular, it is possible to use the noise symbols to identify the dependency between variables (or even one variable appearing in more complex algebraic expressions, as in $(1+x)(1-x)$), and hence to prevent the dynamic interval bounds on expressions from growing as rapidly as they would if an interval arithmetic used. This technique requires good approximations of higher degree expressions with affine forms in the appropriate noise components. This of course is fairly more time-consuming than standard interval arithmetic.

Bernstein-based intervals In the CAD-CAM and computer graphics communities, since the pioneering works by Bézier and de Casteljau, it is well-known that the properties of the Bernstein basis yield sharp enclosing intervals for polynomials [27–29], as Fig. 3 gives visual evidence. This knowledge finally percolated to other communities [30–32]. A Bernstein basis solver for polynomial systems written by Mourrain and Pavone is available in GALAAD [33].

The canonical basis for degree d polynomials in t is: $T = (1, t, t^2 \ldots t^d)$. The Bernstein basis is: $B = (B_0^{(d)}(t), B_1^{(d)}(t), \ldots B_d^{(d)}(t))$ where $B_i^{(d)}(t) = \binom{d}{i} t^i (1-t)^{d-i}$.

The conversion between the Bernstein and the canonical bases is a linear mapping, representable by a $(d+1)\times(d+1)$ square matrice M such that: $B = TM$. For instance, for $d = 3$:

$$(B_0, B_1, B_2, B_3) = (1, t, t^2, t^3) \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}$$

Some remarkable properties of the Bernstein basis are:

- The polynomial lies inside the convex hull of its coefficients z_{ij} in the Bernstein basis. Thus $z = f([0,1],[0,1])$ lies in $[\min_{i,j} z_{ij}, \max_{i,j} z_{ij}]$. This property extends to all dimensions.

– The de Casteljau method computes the coefficients in the Bernstein basis of the polynomials $f(2x)$ and $f(2x-1)$ without having to refer to the canonical basis; it allows to subdivide the studied interval $[0,1]$ into two subintervals $[0,1/2]$ and $[1/2,1]$. This method extends to multivariate polynomials.
– The control points of the image of a curve (or surface) by some affine transform T are the images by T of the control points of the curve (or surface). *i.e.,* $T(\text{CtrPts(Surface)}) = \text{CtrPts}(T(\text{Surface}))$, where T is any affine transform. This extends to all dimensions.

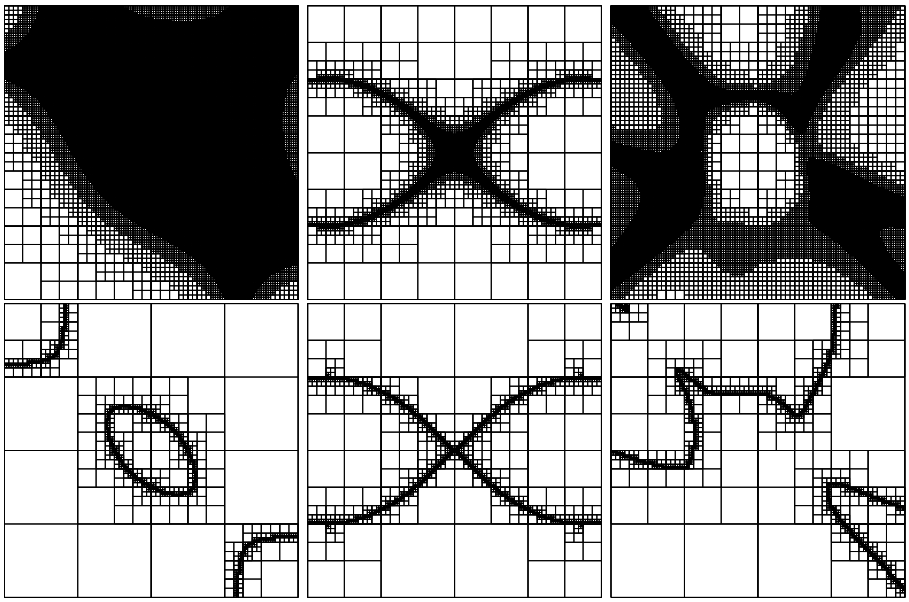

Fig. 3. The naive interval arithmetic is used in the top row, and the Bernstein-based interval arithmetic in the bottom row, for displaying the same three curves. Left column: the curve has equation: $f(x,y) = 15/4 + 8x - 16x^2 + 8y - 112xy + 128x^2y - 16y^2 + 128xy^2 - 128x^2y^2 = 0$ and is displayed in the square $[0,1] \times [0,1]$. Middle column: Cassini's oval. Right column: a random curve with degree 14.

What interval analysis can do In this paragraph we discuss five applications of interval analysis:

a. tracing of implicit curves or surfaces,
b. displaying of strange sets,
c. solving non-linear systems of equations using Newton method,
d. solving non-linear systems of equations using Bernstein basis; and
e. representing boundaries of geometric objects (interval geometry).

a. Tracing implicit curves or surfaces:
Interval analysis is used in computer graphics to display semi-algebraic sets defined by boolean combinations of polynomial inequalities, in 2D (such as $f(x, y) \geq 0$ and $g(x, y) \geq 0$) or 3D (such as $f(x, y, z) \geq 0$ and $g(x, y, z) \geq 0$). See Fig. 3 for an illustration. Without loss of generality, we present the classical subdivision method used to display 2D curves defined by an implicit equation $f(x, y) = 0$ as follows:

```
Subdivide(f: function, X : interval, Y: interval, depth: int) {
  interval F = f(X, Y);
  draw_rectangle(X, Y);
  if (F contains 0)
   { if (depth==0) fill_rectangle(X, Y)
     else
      {
        interval X1 = [min(X), middle(X)];
        interval X2 = [middle(X), max(X)];
        interval Y1 = [min(Y), middle(Y)];
        interval Y2 = [middle(Y), max(Y)];
        Subdivide(f, X1, Y1, depth-1);
        Subdivide(f, X1, Y2, depth-1);
        Subdivide(f, X2, Y1, depth-1);
        Subdivide(f, X2, Y2, depth-1);
      }
  }
}
```

b. Displaying strange sets:
Interval analysis methods may also be used to compute guaranteed covers of fractals or strange sets such as Julia sets or the Hénon attractor (see Fig. 4), a process that we now sketch rapidly:

- The "Hénon mapping" sends the point $(x, y) \in \mathbb{R}^2$ onto $(x', y') = (y + 1 - ax^2, bx)$, where a, b are two parameters; classically, $a = 1.4$ and $b = 0.3$.
- The Hénon set is the set of points whose orbit (set of H-iterates) remains bounded.
- Assume, for the sake of simplicity, that a square bounding box B of the Hénon set is known:
  - Subdivide B in 16×16 square cells;
  - For every cell C, compute an enclosure of $H(C)$, using an interval arithmetic;
  - Each time $H(C)$ overlaps a cell C_i, add an arc $C \to C_i$ in a graph H_G, whose vertices are cells partitioning B.
  - Compute the strongly connected components of H_G: a strongly component is *transient* if and only if it contains only one cell C and there is

no loop $C \to C$ in H_G; then this cell cannot contain any point in the Hénon set.
- For non-transient cells C, the graph H_G contains at least one loop $C \to C$ or $C \to C_1 \to \dots C$, thus C may contain points of the Hénon set. Non-transient cells are subdivided, and the method is applied again.

– The recursion stops when an accurate picture is obtained.

This method guarantees s sharp cover of the Hénon set, and shows that the classical orbit method used to display strange sets may yield erroneous results (for some values of a, b). Even the need of an initial bounding box may be relaxed, since the projective plane is bounded (it may be mapped to a sphere). Refer to [34] for more details.

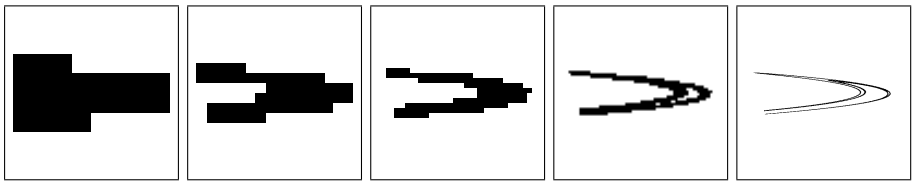

Fig. 4. The Hénon set with increasing $8^2, 16^2, 32^2, 64^2, 1024^2$ resolutions.

c. Solving non-linear systems of equations using interval Newton methods:
Let $F(x) = 0$ be a system of n equations in n unknowns. The classical Newton-Raphson method iterates: $x_{n+1} \leftarrow x_n + F(x_n)F'(x_n)^{-1}$ until convergence. A variant is the secant method which does not update the inverse of the derivative F' at each step, and iterates: $x_{n+1} \leftarrow S(x_n) = x_n + F(x_n)J^{-1}$, where J is the Jacobian of F at x_0.

To find roots of F inside a prescribed initial box X, a natural idea is to compute $S(X)$ for an interval X. Since $S(X) = X+$ something, the width of $S(X)$ is always greater than the width of X, if evaluated with the naive interval arithmetic, which prevents convergence. A solution is to use the central form evaluation to compute $S(X)$, which leads directly to the Krawczyk (or Krawczyk-Moore) operator.

The interval secant method may be described as follows: if $S(X) \subset X$, then S is contractant inside X, thus X contains an isolated root of F, and the secant method will converge to this root starting from any point of X. If $X \cap S(X) = \emptyset$, then X contains no root of F. Otherwise possible roots of F in X can only be in $X \cap S(X)$; if $X \cap S(X)$ is significantly smaller than X, then the method is resumed on $X \cap S(X)$, *i.e.,* $S(X \cap S(X))$ is computed, etc. Otherwise, the former is likely to contain several roots of F, and is bisected: eventually, these bisections

separate roots. The combinatorial complexity of the method is due of course to the bisection steps.

The computation of

$$X \cap S(X) = (X_1 \cap S_1(X_1, \dots X_n), \dots X_n \cap S_n(X_1, \dots X_n)),$$

can be optimized using the most recent value of $X_i, i = 0, \dots k$ when computing $S_k(X)$, and hence by cascading the computations:

$$X_1 \leftarrow X_1 \cap S_1(X_1, \dots X_n), X_2 \leftarrow X_2 \cap S_2(X_1, \dots X_n),$$

and so, see [35, 36] for more details.

d. Solving non-linear systems of equations using Bernstein basis:
The Bernstein-based subdivision method may also be used to solve polynomial systems of equations, as it was done by Patrikalakis [29] and Garlof [30]. These authors use the tensorial Bernstein basis, which has limitations: it is a dense representation (even if the polynomial is sparse in the canonical basis), and has an exponential number of coordinates.

A typical solution is likely to use the simplicial Bernstein basis, as obtained from the development of: $(x_0 + x_1 + x_2 + \dots x_n)^d$, where d is the total degree, and $x_0 + x_1 + x_2 + \dots x_n = 1$. The simplicial basis only has $O(n^d)$ coefficients, has the same convex hull property as the tensorial basis, and the de Casteljau method also allows to divide a Bernstein simplex into two, along one of its edge. Recently, Nataraj *et al.* [31, 32] combine tensorial Bernstein basis and Taylor expansions, to achieve superconvergence of inclusion functions, and to account for non polynomial functions.

e. Representing boundaries of geometric objects:
Quite recently also, several authors (*e.g.,* [37]) suggested to extend the principle of intervals, which enclose boundaries in 1D, to geometric objects in 3D (interval geometry). A boundary surface is thus enclosed inside two polyhedra (typically with rational coordinates). The inside and outside polyhedra may have different topologies, for instance a different number of connected components. Constructive Analysis extends to such objects, *i.e.,* it is possible to compute (using increasing computing resource) closer and closer nested approximations.

This approach does not suffer from the incompatibility between, on the one hand, classical boundary representations or geometric algorithms which require to compute the exact sign of numbers: $0, +, -$, and on the other hand interval arithmetics which are intrinsically unable to compute the sign of 0 from an enclosing interval. However interval geometry suffers from the same intrinsic restriction as interval arithmetics: interval arithmetics cannot compute the sign of 0, and interval geometry cannot detect if two shapes are tangent: it can only detect whether they are disjoint or intersect.

In another effort similar to the one of interval geometry, Foufou *et al.* introduced the fuzzy geometry and proposed to use it to classify surfaces against their

intersection status [38] . Geometric entities are replaced by thicken entities. The associated fuzzy intersection algorithm provides a three-state classification of surfaces couples: certainly intersecting, certainly non intersecting and potentially intersecting.

What interval computations cannot do All interval-based solutions have the intrinsic limitation of being only "half deterministic", in the following sense. If a number is zero, no interval computation can detect it (in finite-time); on the other hand, if the number is either strictly positive or strictly negative, a tight enough interval will find its sign. Thus interval analysis cannot decide nullity. For the same reason, it cannot decide equality (*i.e.,* the nullity of the difference of two equal numbers). This argumentation has been formalized by Constructive Analysis [39] as follows.

By definition, a number is computable if a finite-time algorithm provides an arbitrarily tight interval enclosing it. For instance, the interval arithmetic provides a stream of (nested) Cauchy intervals. Interval arithmetics which are capable of computing such arbitrarily tight enclosures are called Real Arithmetics. Several implementations have been suggested (and sometimes proven) [40, 41].

By definition, a function $f(x_1, x_2 \dots x_n)$ is computable if $f(x_1, x_2 \dots x_n)$ is computable whenever $x_1, x_2 \dots x_n$ are computable.

No interval arithmetic may be more powerful than the Real Arithmetic, an ideal interval arithmetic which does not suffer from practical limitations (*e.g.,* restricted memory). But even Real Arithmetic is intrinsically restricted:

Theorem 7 (Real Analysis). *Only continuous functions are computable.*

This theorem has considerable implications, which even today have not been fully realized. For instance let us just consider that the sign function is equal to +1 for positive numbers, to -1 for negative numbers, and to (say) 0 for zero. This function is discontinuous in 0; since no interval computation may establish that a number is zero, the sign of zero is not computable.

There is a quantitative variant of this theorem: the greater $|f'_{x_i}|$, the sharper the interval for x_i so as to compute $f(x_1, \dots x_n)$ with a prescribed precision. For a discontinuous function, its slope is infinite at the discontinuity, thus the argument has to be known with infinite precision to compute the function at that point.

In practice, an arbitrarily tight interval cannot be computed, because of storage limitations (and the patience limitation of the user). But the theorem holds even without taking this common-sense argument into account. This theorem has dramatic consequences on geometric computing.

The most basic geometric algorithms typically require to compute non-continuous functions, such as: do two geometric objects intersect or not? does this point lie

on the left, on the right, or just on a boundary surface? does this triple of 2D points turn left, right, or are they aligned?, and so on.

As we have seen, geometric algorithms use "predicates" for branching; the branch is chosen according to the sign of a predicate, such as the orientation predicate. Predicates being discontinuous functions, they are not computable with Real Analysis. Hence, interval computations (which seem to be the only realistic guaranteed way of computing) are not compatible with the most basic geometric algorithms of Computational Geometry, and not compatible with the most basic geometric data structures used to represent incidence between points, curves and surfaces. Nearly all geometric modelers use these data structures.

Stated crudely: no geometric programs computing non-continuous functions, and using floating-point arithmetic, or using interval computations or even Real Arithmetic, may be proven. Actually, they all are wrong. The best they can achieve is to work most of the time (*i.e.,* not fail more often than their competitors).

This limitation of interval arithmetic is the main argument to promote discrete geometry, and stochastic or probabilistic approaches.

4 Probabilistic approaches

4.1 Solutions at the arithmetic level

Probabilistic gap arithmetics A practicable but only probabilistic method is to compute with some "big-float" library, and to use the ϵ heuristic (see section 3.1) with a small ϵ, *e.g.,* $\epsilon = 10^{-200}$, and hope that life will not be so bad as to produce a counterexample. We are not aware of any report on this kind of experiment.

The stochastic arithmetic The stochastic arithmetic, implemented in CADNA [42], estimates and limits the propagation of round-off errors of the floating-point arithmetic in scientific software (*e.g.,* simulation of fluid mechanics), and may detect the source of numerical instabilities. A stochastic number is represented by a mean value (a floating-point number) and a variance. Each floating-point operation is performed with n (typically $n = 3$) samples of the stochastic number. A stochastic zero is a number with no significant bits.

While the wrapping effect of interval arithmetics overestimates the interval widths, the stochastic arithmetic gives more realistic confidence intervals, which accounts for the fact that round-off errors often compensate. The stochastic arithmetic is well-adapted to programs with few branchings, where tests are used mainly to detect the convergence of a numerical algorithm.

For geometric computations, the stochastic method will decide the sign of numbers equal to 0 or close to 0 only randomly, thus it takes neither exact nor consistent decisions and it does not seem to make sense to use a stochastic arithmetic for geometric algorithms (*e.g.,* computing a convex hull or a Delaunay triangulation). However, it does make sense to use the stochastic scheme at the geometric level, as we shall soon see.

Zero free arithmetic There are 2 kinds of exact arithmetics: those (*e.g.,* rational arithmetics) which provide a test against zero (*i.e.,* is a number equal, superior or inferior to 0?), and those that do not (*e.g.,* real constructive arithmetics). In the latter class, real numbers are typically represented by streams (potentially infinite lists) of nested Cauchy intervals.

In geometric methods, programmers usually round data to integers or rationals, and then use the first kind of arithmetics. This precludes Computational Geometry from addressing algebraic non-rational problems (*e.g.,* intersection between algebraic curves and surfaces).

The idea of the zero-free exact arithmetic [43] is to round data numbers to algebraically independent, transcendental numbers, by perturbing them with a stream of random digits. Assume that all the tests in geometric programs involve the signs of polynomials: $f(u_1, u_2 \dots u_n)$ where $u_1, \dots u_n$ are initial data numbers; if the latter are rounded on algebraically independent numbers, then the only polynomial f which can vanish is the identically zero polynomial. It makes no sense to ask for the sign of this polynomial in an instruction such as `if(0==0) then ...`. In consequence, tests never evaluate to zero, and it is possible to use real constructive arithmetics. Moreover this method also solves the problem of degeneracies (the zero-free arithmetic may be seen as a generalization of the so-called "simulation of simplicity" (*SoS*) technique suggested by Edelsbrunner and Mcker [44], with the major difference that *SoS* requires an arithmetic capable of exactly testing the sign of numbers.): three points will never be aligned, four points will never be coplanar nor cocyclic, etc. Finally, this solution allows geometric methods to address problems requiring non-rational arithmetics: algebraic arithmetics, or even transcendental numbers. It becomes possible to use numbers defined by a convergent algorithm, which computes roots of polynomial systems for instance, as long as we are sure that all tests involve algebraically independent numbers.

This assumption is the cornerstone of the zero-free arithmetic, and its main weakness: some geometric programs allow to derive geometric objects or numbers (coordinates for intersection points) from the initial (perturbed) ones, with some geometric constructions. Clearly, the former algebraically depend on the latter. For instance, cutting one (perturbed) initial line with three other (perturbed) initial lines produces three aligned points; the orientation test for these three points will not terminate: the related determinant is zero. A lot of geometric theorems (Pappus, Desargues) allow to construct less trivial alignments, and

occurrences of zero. At the other end of the spectrum, computing $x-x$, where the two instances arise from different computation contexts (*e.g.,* the dependance of the variables has been lost as in interval arithmetic), is null but not "identically null".

Combining intervals and randomized nullity tests The previous method does not terminate in case of a null number. In order to circumvent this problem, it is possible to combine it with a randomized nullity test [45–48]. For instance, assuming all numbers involved are rational, a number is probably null if its interval is sharp enough and contains zero, and the number's hashed value (result of a modular computation modulo a large prime) is zero. This method is efficient and simple in the rational case, but is less appealing in the algebraic case [47, 49, 47, 50].

Probabilistic tests can also be used for polynomials and not only for numbers. J.T. Schwartz [45] introduced this method to test algebraic identities: if a polynomial (*e.g.,* the determinant of a square matrix with polynomial entries) vanishes when evaluated at random values of its variables, it is likely identically null. This test is only probabilistic, but extremely fast. This probabilistic principle, called *proof by example*, is also used for probabilistic proofs of geometric theorems [51–53]. It has been extended beyond polynomials [49, 47, 50], *e.g.,* Tulone *et al.* [49] extend it to radical expressions which occur in ruler and compass geometric constructions and related geometric theorems, such as Pascal's for a circle.

This probabilistic test may be made deterministic in several ways. For the sake of simplicity, consider a polynomial in one variable: if an upper bound d of the degree is known, and if the polynomial vanishes in $d+1$ distinct sample values, then it may only be the zero polynomial; or, if we have an upper bound for the magnitude of the coefficients, we can also use some formula [20] to compute an upper bound for the magnitude of the root module, or a lower bound for the magnitude of the root inverse, so if the polynomial vanishes for a number outside these bounds, then it can only be the zero polynomial [51]. This principle for univariate polynomials may be extended to the multivariate case, and always with an exponential cost.

Clarkson and Shor used random sampling for several geometric algorithms and showed that random subsets can be used optimally for divide-and-conquer algorithms and for bounds computations for incremental building of geometric structures [54, 55].

4.2 Solutions at the geometric level

The motion planning problem In robotics, the motion planning problem (also called the piano mover's problem) consists in finding a trajectory for a robot, which moves from a starting position to a final one in an environment

cluttered with obstacles. The robot is represented by a point in a configuration space; the configuration space is the usual Euclidean space (which allows to describe the location of some origin point of the robot), augmented with all parameters which describe the spatial orientation (3 angles for a rigid body in 3D) and the configuration (one angle for each articulation of the robot, one length for each jack) of the robot. Obstacles – and the constraints of avoiding self-intersection of the robot – forbid some areas of the configuration space. There is a trajectory for the robot if its initial and final configurations lie in the same connected component of the configuration space. Typically, the routine that decides whether a given point in the configuration space is collision-free or not, returns in fact a signed "interpenetration depth" or some kind of signed minimal distance to the closest obstacle. In the first stages of the algorithm, interpenetration is tolerated in order to find a coarse path rapidly. In the final stages, the acceptable tolerance is gradually reduced.

This problem is decidable, and exact and deterministic algorithms from computer algebra (say the Collin's Cylindrical Algebraic Decomposition, and its variants and optimizations) solve it by computing an explicit representation of the feasible part (the part not forbidden by obstacles and self-avoidance constraints of the robot) of the configuration space. However, such approaches turn out to be definitively not practical due to their high combinatorial costs, to their lack of robustness (these methods do not resist inaccuracy and degeneracy), and also to the high dimension of the configuration space (≈ 200 for a human body, ≈ 50 after simplification; from 10 to 20 for a simple robot) occurring in industrial problems.

By the end of the 1980's some roboticians broke away from this exact and deterministic approach. They no more compute an explicit representation of the feasible configuration space. Instead, they randomly sample the configuration space, keeping only collision-free samples; they build a graph, called the *road map*, where close collision-free samples of the configuration space are linked when the line segment (in configuration space) connecting the two samples is collision-free, or, more precisely, when sampling regularly the line segment, all samples are collision-free (the detection of collision for a point in the configuration space is simpler than for a segment). In this approach, finding a trajectory reduces to finding a path in the road map graph, with a standard algorithm like Dijkstra's. Once a path has been found, it may be refined in several ways, for instance by trying to connect two non-contiguous vertices of the path by a segment and resampling more densely a neighborhood of the path.

This method is neither deterministic nor exact; it may fail to find complicated paths in very cluttered environments. Its completeness is probabilistic: it will find a solution (when there is one) with probability one if it runs indefinitely; the probability of failure decreases exponentially with the running time.

This approach is a technological breakthrough as it solves in interactive running times problems of industrial size which are completely out of reach of the deter-

ministic exact methods; moreover it is extremely robust against inaccuracy or degeneracy: for deciding if a sample point of the configuration space is collision-free or not, it uses the standard floating-point arithmetic. We refer the reader to Laumond's book for more details [56].

Roadmap and topologic computations A similar approach is used in [57] to compute (approximated) shortest circuits with a prescribed topology ($\approx$ shape) on a given surface. Actually, the authors started with trying to use mesh representations provided by some industrial software; but these meshes turned out to be inconsistent, containing self intersecting triangles, and gaps (holes between triangles), which means that some sections of the meshes were not even connected. They then decided to sample the mesh, and to build a graph (a roadmap) where each edge connects two close enough samples. They also assume that each triangle in this roadmap (three pairwise-connected vertices) corresponds to a triangular patch on the sampled surface: this assumption is sufficient for roadmaps to allow topological computations, required to decide the equivalence of two circuits on a surface with holes.

Topological computations are usually done from some simplicial complexes such as triangulated meshes; these sophisticated data structures are terribly fragile – it is the reason why CGAL requires exact computations. Roadmaps are not triangulations, and they are sufficient to perform topological computations. Moreover, roadmaps are rudimentary and robust: they use floating-point computations without problems.

No motion planning algorithm uses topological computations yet, as far as we know. Such computations can detect there are several kinds of paths or circuits (*e.g.,* passing at the left or at the right of an obstacle) in the configuration space.

Note: the length of a line through rational vertices is a sum of square roots of rational numbers [58]; this length is an algebraic number with degree 2^{n-1} if there are n vertices. A truly exact algorithm, which exactly computes these lengths (in order to compare them when they are very close) has thus an exponential running time. For this reason, even proclaimed "exact" methods do not use exact computations of lengths.

The radiosity problem The radiosity problem [59, 60] consists in computing photo-realistic images of virtual scenes. Monte Carlo methods simulate the propagation of light by following samples of photons, from the light sources and their reflections or refractions in the scene, until they are absorbed by a surface. Counting the number of absorbed or reflected photons on each surface patch of the scene gives an estimation of the radiosity. Monte Carlo methods do not require an explicit representation of surfaces bounding the objects in the scene, but only a procedure capable of computing the intersection between a line and geometric objects (and the normal to the surface at the intersection point);

computing this intersection is an easily to solve one-dimensional problem. Monte Carlo methods are extremely robust: they never fail because of inaccuracy.

Ray tracing [59] is another method used for scene rendering; it follows light rays, *i.e.,* photons, but start from the (virtual) eye. Like Monte Carlo methods, ray tracing does not need an explicit representation of the scene (similarly the roadmap method for the motion planning problem does not need an explicit geometric representation of the configuration space). It uses the same intersection procedure as Monte Carlo methods. Actually ray tracing and Monte Carlo methods are combined: the former accounts for specular (mirror-like) light reflections, and the latter accounts for diffusion and scattering of light.

Numerous deterministic methods were proposed to compute more or less realistic images of virtual scenes. They rely on an explicit representation of the boundary of objects in the scene, typically approximating meshes (so they were not even exact). As these deterministic methods use sophisticated data structures (*e.g.,* boundary representations, visibility graphs), they are terribly complicated to implement, and fragile. Note that ray tracing and Monte Carlo radiosity are roughly as old as the deterministic methods; it is the increasing power of computers which has made them practical over the last ten years.

5 Conclusion: is randomness the problem or the solution?

The inaccuracy of the floating-point arithmetic is often considered as a source of random noise. The latter perturbs computations, introduces inconsistencies in deterministic geometric algorithms and causes strange behaviors such as crashes or infinite loops. Thus randomness is a problem for deterministic algorithms.

Worse even, exact computations turn out to be intractable, and not relevant for real-world applications (except in some very restricted cases). Randomness kills deterministic geometric algorithms.

Probabilistic geometric methods are simpler, very robust, and become relevant and tractable with the increasing power of computers: randomness may very well be the solution to randomness.

References

1. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys **23** (1991) 5–48
2. Schirra, S.: Precision and robustness in geometric computations. In van Kreveld, M., Nievergelt, J., Roos, T., Widmayer, P., eds.: Algorithmic Foundations of GIS. LNCS 1340. Springer (1997)
3. Hoffmann, C.M.: Robustness in geometric computations. Journal of Computing and Information Science in Engineering **1** (2001) 143–156

4. Yap, C.: Robust geometric computation. In Goodman, J.E., O'Rourke, J., eds.: Handbook of Discrete and Computational Geometry. CRC Press (2004) 927–952
5. Keyser, J.: Robustness issues in computational geometry. Technical report, Comp. 234 Final Paper, Duke University (1997)
6. Sugihara, K., Iri, M.: A solid modelling system free from topological inconsistency. Journal of Information Processing **12** (1989) 380–393
7. Sugihara, K., Iri, M.: A robust topology-oriented incremental algorithm for voronoi diagrams. IJCGA **4** (1994) 179–228
8. Sugihara, K.: A robust and consistent algorithm for intersecting convex polyhedra. Computer Graphics Forum **13** (1994) 45–54
9. Knuth, D.E.: Axioms and Hulls. Volume 606 of Lecture Notes Comput. Sci. Springer-Verlag, Heidelberg, Germany (1992)
10. Michelucci, D., Moreau, J.M.: Lazy arithmetic. IEEE Transactions on Computers **46** (1997) 961–975
11. Fabri, A., Giezeman, G.J., Kettner, L., Schirra, S., Schönherr, S.: The CGAL kernel: A basis for geometric computation. In Lin, M.C., Manocha, D., eds.: Proc. 1st ACM Workshop on Appl. Comput. Geom. Volume 1148 of Lecture Notes Comput. Sci., Springer-Verlag (1996) 191–202
12. Klatte, K., Kulisch, U., Lawo, C., Rausch, M., Wiethoff, A.: C-XSC, A *C++* class library for extended scientific computing. Springer-Verlag, Berlin/Heidelberg/New York, N.Y. (1993)
13. Funke, S., Mehlhorn, K.: LOOK – a lazy object-oriented kernel for geometric computations. In: Proceedings 16th Annual ACM Symposium on Computational Geometry, Hong-Kong, ACM Press (2000) 156–165
14. Mehlorn, K., Naher, S.: LEDA: A platform for combinatorial and geometric computing. Communications of the ACM **38** (1995) 96–102
15. Mehlhorn, K., Naher, S.: The LEDA Platform for Combinatorial and Geometric Computing. Cambridge University Press (1999) 1018 pages.
16. Karamcheti, V., Li, C., Pechtchanski, I., Yap, C.: A core library for robust numeric and geometric computation. In: Proceedings 15th Annual ACM Symposium on Computational Geometry, ACM Press (1999) 351–359
17. Canny, J.: The complexity of robot motion planning. M.I.T. Press, Cambridge, Mass. (1988)
18. Pion, S., Yap, C.: Constructive root bound method for k-ary rational input numbers. In: Proc. 18th ACM Symp. on Computational Geometry, ACM Press (2003) San Diego, California.
19. Li, C., Yap, C.: A new constructive root bound for algebraic expressions. In: 12th ACM-SIAM Symposium on Discrete Algorithms (SODA). (2001)
20. Mignotte, M., Stefanescu, D.: Polynomials: An algorithmic approach. In: Discrete Mathematics and Theoretical Computer Science Series. Volume XI. Springer (1999)
21. Burnikel, C., Fleischer, R., Mehlhorn, K., Schirra, S.: A strong and easily computable separation bound for arithmetic expressions involving square roots. In: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms table of contents, New Orleans, Louisiana, United States (1997) 702–709
22. Scheinerman, E.R.: When close enough is close enough. American Mathematical Monthly **107** (2000) 489–499
23. Keyser, J., Culver, T., Foskey, M., Krishnan, S., Manocha, D.: ESOLID - a system for exact boundary evaluation. Computer-Aided Design (Special Issue on Solid Modeling) **36** (2004) 175–193
24. Moore, R.: Interval Analysis. Prentice Hall, Englewood Cliffs, N.J. (1966)

25. Andrade, M.V.A., Comba, J.L.D., Stolfi, J.: Affine arithmetic. In: Abstracts of the International Conference on Interval and Computer-Algebraic Methods in Science and Engineering (INTERVAL'94), St. Petersburg (Russia) (1994) 36–40
26. de Figueiredo, L.H., Stolfi, J.: Affine arithmetic: Concepts and applications. Numerical Algorithms **37** (2004) 147–158
27. Farin, G.: Curves and Surfaces for Computer Aided Geometric Design. Academic Press (1990)
28. Hu, C.Y., Patrikalakis, N., Ye, X.: Robust interval solid modelling. part 1: Representations. Part 2: Boundary evaluation. CAD **28** (1996) 807–817, 819–830
29. Sherbrooke, E.C., Patrikalakis, N.: Computation of the solutions of nonlinear polynomial systems. Computer Aided Geometric Design **10** (1993) 379–405
30. Garloff, J., Smith, A.P.: Investigation of a subdivision based algorithm for solving systems of polynomial equations. Journal of nonlinear analysis : Series A Theory and Methods **47** (2001) 167–178
31. Nataraj, P.S.V., Kotecha, K.: Global optimization with higher order inclusion function forms part 1: A combined Taylor-Bernstein form. Reliable Computing **10** (2004) 27–44
32. Nataraj, P.S.V., Kotecha, K.: Higher order convergence for multidimensional functions with a new Taylor-Bernstein form as inclusion function. Reliable Computing **9** (2003) 185–203
33. Mourrain, B., Rouillier, F., Roy, M.F.: Bernstein's basis and real root isolation. Technical Report 5149, INRIA Rocquencourt (2004)
34. Michelucci, D., Foufou, S.: Interval based tracing of strange attractors. International Journal of Computational Geometry and Applications **16** (2006) 27–39
35. Kearfott, R.: Rigorous Global Search: Continuous Problems. Kluwer, Dordrecht, Netherlands (1996)
36. Hansen, E.R., Walster, G.W.: Global Optimization Using Interval Analysis. Marcel Dekker, New York (2003)
37. Edalat, A., Lieutier, A.: Foundation of a computable solid modelling. Theoretical Computer Science **2** (2002) 319–345
38. Foufou, S., Brun, J., Bouras, A.: Surfaces intersection for solid algebra: A classification algorithm. In Strasser, W., Klein, R., Rau, R., eds.: In Proc. Theory and Practice of Geometric Modeling'96, Tubingen, Germany, Springer Verlag (1996)
39. Weihrauch, K.: Computable Analysis An Introduction. Springer (2000)
40. Boehm, H.J., Cartwright, R., Riggle, M., O'Donnell, M.: Exact real arithmetic: a case study in higher order programming. In: Proc. ACM Conference on Lisp and Functional Programming. (1986) 162–173
41. Lester, D., Gowland, P.: Using pvs to validate the algorithms of an exact arithmetic. Theoretical Computer Science **291** (2003) 203–218
42. Vignes, J., Alt, R.: An efficient stochastic method for round-off error analysis. In: Accurate Scientific Computations. (1985) 183–205
43. Michelucci, D., Moreau, J.M.: ZEA – a zero-free exact arithmetic. In: Proceedings 12th Canadian Conference on Computational Geometry, Fredericton, New Brunswick (2000) 153–157
44. Edelsbrunner, H., Mücke, E.: Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. ACM Trans. Graph **9** (1990) 66–104
45. Schwartz, J.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM **4** (1980) 701–717
46. Agrawal, A., Requicha, A.G.: A paradigm for the robust design of algorithms for geometric modeling. Computer Graphics Forum (EUROGRAPHICS'94) **13** (1994) C–33–C–44

47. Monagan, M., Gonnet, G.: Signature functions for algebraic numbers. In: Proc. ISSAC, ACM Press (1994) 291–296
48. Benouamer, M., Jaillon, P., Michelucci, D., Moreau, J.: Hashing lazy numbers. Computing **53** (1994) 205–217
49. Tulone, D., Yap, C., Li, C.: Randomized zero testing of radical expressions and elementary geometry theorem proving. In: International Workshop on Automated Deduction in Geometry (ADG'00). (2000)
50. Gonnet, G.H.: New results for random determination of equivalence of expressions. In: SYMSAC'86: Proceedings of the fifth ACM symposium on Symbolic and algebraic computation, New York, NY, USA, ACM Press (1986) 127–131
51. Hong, J.: Proving by example and gap theorem. In Press, I.C.S., ed.: 27th symposium on Foundations of computer science, Toronto, Ontario (1986) 107–116
52. Kortenkamp, U.: Foundations of Dynamic Geometry. PhD thesis, ETH Zurich, Institut fur Theoretische Informatik (1999)
53. Foufou, S., Jurzak, J.P., Michelucci, D.: Numerical decomposition of geometric constraints. In: Proc. ACM Conference on Solid and Physical Modeling. (2005) 143–151
54. Clarkson, K.L., Shor, P.W.: Applications of random sampling in computational geometry, II. Discrete and Computational Geometry **4** (1989) 387–421
55. Clarkson, K.L.: New applications of random sampling in computational geometry. Discrete and Computational Geometry **2** (1987) 195–222
56. Laumond, J.P., ed.: Robot Motion Planning and Control. Lecture Notes in Control and Information Science, Springer Verlag (1998)
57. Michelucci, D., Neveu, M.: Shortest circuits with given homotopy in a constellation. In: 9th ACM Symp. Solid Modeling and Applications. (2004) 297–302
58. Choi, J., Sellen, J., Yap, C.: Approximate Euclidean shortest path in 3-space. Int'l. J. Computational Geometry and Applications (1997) 271–295 Journal special issue. Also in 10th ACM Symposium on Computational Geometry, 1994.
59. Glassner, A.: An Introduction to Ray Tracing. Andrew Glassner ed., Academic Press (1989) ISBN 0-12-286160-4.
60. Glassner, A.S.: Principles of Digital Image Synthesis. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1994)