FLOATING-POINT GEOMETRY: TOWARDS GUARANTEED GEOMETRIC COMPUTATIONS WITH APPROXIMATE ARITHMETICS?

Jean-Claude Bajard[a], Philippe Langlois[b], Dominique Michelucci[c], Géraldine Morin[d], Nathalie Revol[e]

[a] LIRMM, Montpellier, France; [b] DALI, Perpignan, France; [c] LE2I, Dijon, France; [d] INPT, Toulouse, France; [e] INRIA, Univ. de Lyon, France

ABSTRACT

Geometric computations can fail because of inconsistencies due to floating-point inaccuracy. For instance, the computed intersection point between two curves does not lie on the curves: it is unavoidable when the intersection point coordinates are non rational, and thus not representable using floating-point arithmetic.

A popular heuristic approach tests equalities and nullities up to a tolerance ε. But transitivity of equality is lost: we can have $A \approx B$ and $B \approx C$, but $A \not\approx C$ (where $A \approx B$ means $\|A-B\| < \varepsilon$ for A, B two floating-point values). Interval arithmetic is another, self-validated, alternative; the difficulty is to limit the swell of the width of intervals with computations. Unfortunately interval arithmetic cannot decide equality nor nullity, even in cases where it is decidable by other means.

A new approach, developed in this paper, consists in modifying the geometric problems and algorithms, to account for the undecidability of the equality test and unavoidable inaccuracy. In particular, all curves come with a non-zero thickness, so two curves (generically) cut in a region with non-zero area, an inner and outer representation of which is computable. This last approach no more assumes that an equality or nullity test is available. The question which arises is: which geometric problems can still be solved with this last approach, and which cannot?

This paper begins with the description of some cases where every known arithmetic fails in practice. Then, for each arithmetic, some properties of the problems they can solve are given. We end this work by proposing the bases of a new approach which aims to fulfill the geometric computations requirements.

Keywords: Geometric computations, robustness issue, numerical inaccuracy, interval analysis, computational geometry, CAD-CAM, tolerant modeling.

Computers are increasingly used for geometric computation, and not only for numerics. This phenomenon is illustrated by numerous applications such as Computer Aided Design (CAD) and Manufacture (CAM), finite elements methods, computer assisted medicine, video game development, special effects for the movie industry, virtual and augmented reality, robotics.

From its beginning, geometric computing has faced difficulties due to the approximations of floating-point arithmetic*. The uncertainty on numerical results - and data - is generally not the problem: an accuracy of the order of a micron is usually sufficient for the industry. The difficulty is more subtle: numerical imprecision of floating-point arithmetic, as small as it may be, sometimes leads to incoherences during the execution of programs, or inconsistencies in some of the results, such as data structures that represent the topology of geometric objects. These contradictions can be fatal: the program infinitely loops, or crashes, or its results are inconsistent (and not just approximate) which causes the failures of the following treatments. Indeed, the programs and data structures are usually not developed to withstand contradictions and inconsistencies which are, in theory, impossible, but do occur because of numerical instabilities. Numerical inaccuracy also prevents the inter-operability of geometric softwares, and the exchange of geometric data.

Section 1 illustrates with simple examples the inconsistencies due to numerical inaccuracy. It also introduces the epsilon heuristic, the first ad hoc approach proposed to cope with the inaccuracy problem, and shows its limitations. Section 2

Corresponding author: Jean-Claude.Bajard@lirmm.fr

*Personal communication with Jean-Marc Brun, one of the fathers of “Euclide”, the very first French geometric modeler for CAD; J.-M. Brun has identified problems due to catastrophic cancellations.

presents classical attempts to handle the inaccuracy problems. Inaccuracy can be removed, resorting to exact computations or decisions; the Computational Geometry community is the main user of this solution. Interval arithmetic uses enclosures of the objects, to account for the inaccuracy in their representations. Probabilistic approaches, on the other hand, do not aim to provide guaranteed results but sample the object (eg. they sample points on a surface) as required by the current processing applied to this object. Another line of thought, advocated in this paper, considers that inaccuracy is unavoidable, and proposes to embrace inaccuracy: this is *tolerant modeling*, presented in section 3. Tolerant modeling replaces pure mathematical objects (points, curves, surfaces) which have no thickness, by thickened variants which have inner representations, contrarily to pure mathematical objects. This section also presents some first geometric methods which apply to toleranced objects, and which are able to provide results with guaranteed correctness (*eg.* exact topology) using only approximate computations. The appendix lists typical geometric representations and related problems and applications.

1. EXAMPLES OF PROBLEMS WITH FLOATING-POINT ARITHMETIC

First some examples are given, of inaccuracies that occur when floating-point arithmetic is used for the computations. The possible consequences of these inaccuracies on classical algorithms in geometric computing are illustrated. We end up with an introduction of the so-called ε-heuristic, developed to handle such inaccuracies, and we illustrate its limits.

Example 1. In 2D, let us compute the smallest convex polygon containing a finite set of given isolated points. A straightforward method consists in considering in turn each pair of points (A,B): the segment AB is an edge of the convex hull if and only if all the other points lie on the same side of the line AB. This simple method is correct, but it may fail in the case of four points almost aligned and it may then output an empty convex hull, due to the imprecision of floating-point arithmetic. To get four points almost aligned, start with four points on the x-axis and apply an arbitrary rotation, using floating-point arithmetic. In such a case, the failure of this method is only due to the numerical uncertainty. The explanation is detailed in the following example. Numerical errors also cause failures of more sophisticated algorithms for computing the convex hull in 2D and 3D.

Example 2. In the plane, three distinct unaligned points A,B,C define an angle ABC that is oriented either to the left or to the right; orientation is given by the sign of the determinant:

$$\text{orien}(A,B,C) = \det \begin{pmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{pmatrix}$$

This determinant vanishes if the three points are aligned. In theory, the orientation of CBA is the opposite of the orientation of ABC: $\text{orien}(A,B,C) = -\text{orien}(C,B,A)$. In practice, one can easily generate three points A,B,C almost aligned, such that ABC and CBA have the same orientation. Indeed, $\text{orien}(A,B,C)$ is the sum of two terms: the exact result, and a "numerical noise" or numerical error. When the exact value is close to zero, the numerical error term predominates: the resulting value of $\text{orien}(A,B,C)$ and its sign are thus unpredictable.

A first possible consequence of this inaccuracy is that a convex hull of 4 points can be empty. Here is another possible consequence: let ABC and ABC' two contiguous triangles of a mesh, with the vertices C and C' located on opposite sides of AB. Let a point K located in ABC, very close to the edge AB, cf. Fig. 1. The computation of the orientations ($\text{orien}(A,B,K)$, etc...) is used to determine which triangle of the mesh contains the point K. An inconsistency on $\text{orien}(A,B,K)$ and $\text{orien}(B,A,K)$ may imply that neither the triangle ABC nor the triangle ABC' contains K. If K' is a point located within the triangle ABC', the intersection point between the triangular mesh and the line KK' can be missed, even when the line is not at all tangent to the mesh.

Computing or counting intersecting points between a mesh and a line or a half line is a basic geometric routine, typically used to decide if a given point lies inside or outside a geometric object whose boundary is described by a mesh. The point lies inside when a half line starting from this point intersects the mesh at an odd number of intersection points.

The computation of the sign of the orientation of three points is a fundamental "predicate", which is used in the if-then-else tests of 2D geometric programs. Another predicate often used in 2D determines if a point belongs to the circle circumscribed to three other points. Indeed, all of the algorithms for 2D geometry (which handles segments, lines, polygons, Delaunay triangulation, Voronoï diagrams) can be written using only these two predicates. Both predicates can easily be

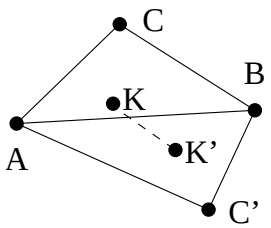

Figure 1. Example where K belongs neither to the triangle ABC nor to the triangle BAC'; the intersection between KK' and AB is missed.

generalized in 3D, and are used to perform the intersection between polyhedral or 3D Delaunay triangulation, for example, in reverse engineering for the reconstruction of objects from point clouds which sample their surface.

Example 3. Computing the intersection point between two planar segments is a trivial problem. However, the intersection point does usually not have coordinates that may be represented exactly as floating-point numbers. The computed intersection point will therefore lie not exactly on the two segments. This introduces a contradiction between on the one hand the topological information stored in the geometric representation, and on the other hand the numerical model (coordinates of the points, and line equation) of this representation. The same applies to the intersection point between two curves in 2D, or between three surfaces in 3D.

At first glance, the problem of numerical inaccuracy seems trivial, and easy to fix. It may seem sufficient to consider that the numbers whose absolute value is less than a threshold ε are zero and that two numbers differing by less than ε are equal. This technique is known as *epsilon-heuristic*, where ε measures the accuracy of the computations. But things are not that simple. For example, should relative or absolute test be used when comparing two numbers? Should the same ε be used for lengths, areas and volumes? Moreover, this heuristic assumption introduces new potential sources of inconsistency: one can find numbers $a < b < c$ such that a is equal to b up to the precision ε, b is equal to c up to the precision ε, but a and c are different because $c - a > \varepsilon$. Equality up to ε loses the transitivity property of equality. But only a few geometric algorithms still work without this property.

The proposed ε-heuristic is therefore incorrect, but it has an empirical justification: it works in most cases and its implementation seems simple. When it fails for a given set of data, the user perturbs the precision ε by trial and error until it works. The ε-heuristic is historically the first strategy used in industry, for example in geometric modelers for CAD-CAM, and it is still the most frequent today in industrial software. It requires a human operator to tune the ε. In the case of geometric modelers for CAD-CAM, users have learned tricks to avoid such errors, *eg.* boolean operations between objects must be computed before moving them around. In short, users adapt the values of parameters, tolerances or data, to move away from "difficult areas". However, this ε-heuristic cannot be used without risk for autonomous robots, when they perform geometric computation to plan their trajectories in a cluttered environment: indeed this heuristic sometimes fails.

2. THE QUEST FOR ROBUSTNESS: CLASSICAL ATTEMPTS

This section reviews different attempts, for tackling the problem of guaranteed geometric computations: first the approach proposed by the computational geometry community, then interval arithmetic, constructive analysis, practical solutions used in industrial software, and probabilistic and statistical methods.

2.1 The exact computation paradigm of Computational Geometry

Computational Geometry[1,2] focuses on the theoretical complexity of geometric problems, and it suggests optimal algorithms for some fundamental problems: convex hull, boolean operations between polyhedrals, computations of Voronoï diagrams or Delaunay triangulations, in 2D or 3D. It considers geometric objects which are numerically simple, in general, falling into the category of "linear objects": segments, lines, planes, triangles or polygonal faces in 3D. Initially, computational geometry[3] assumed a model of computers where arithmetic operations $(+, -, \times, \div, \sqrt{})$ are performed exactly in constant time. However, with floating-point arithmetic, each operation is performed in constant time, but with a rounding error. Algorithms from computational geometry are actually among the most sensitive, that is, the less resistant to numerical approximations. Indeed, these algorithms owe their efficiency to an intensive use of arithmetic coherence (the fact that $\mathbb{R}$ and $\mathbb{C}$ are fields) or geometric coherence: for example, if A is a point inside a polygon, and B a point outside,

then the segment *AB* crosses the polygon boundary an odd number of times. It turns out that numerical errors invalid such statements, as seen above.

To solve the problems caused by numerical inaccuracy, Computational Geometry uses exact computations, specifically it takes correct and therefore consistent decisions in if-then-else tests. Typically data points coordinates are rounded to integers, and the sign of determinants or polynomials in geometric predicates (orien, etc) is computed using an exact arithmetic. In passing, note that rounding is easy only for unstructured points sets, since rounding does not preserve orientation, alignment, coplanarity, convexity, etc: for instance, rounding a convex polygon can make it concave, and rounding two disjoint but close segments or polygons may introduce intersections. Actually, rounding a polyhedron or an arrangement without introducing self-intersections turns out to be a difficult problem. Fortunately, the input for 2D and 3D Delaunay triangulation is usually an unstructured set of points, and Delaunay triangulation is the method from Computational Geometry which is most frequently used in CAD-CAM.

Nowadays, a classical optimization is to use exact mathematical computations only when strictly necessary. Michelucci was one of the first to use lazy exact decisions, in his thesis work in 1984. With Jean-Michel Moreau,[4–7] they introduced the idea of *lazy arithmetic*, which became known as *filtering* in the computational geometry community and implemented in geometric softwares such as CGAL or LEDA.

Unfortunately, even with lazy arithmetic, only rational calculations are practical, while non rational numbers occur very easily, with rotations of parts, intersection points between circles, square roots of distances, etc, intersections between non linear curves or surfaces. There have been some attempts (Michelucci, Manocha, Yap, and a few others) to use algebraic lazy arithmetics, such as quadratic arithmetic. The latter allows 2D geometric constructions "with a ruler and a compass"; Bouhineau[8] has used such an arithmetic in his thesis, for Cabri Géomètre. All these attempts have confirmed that the exact computation paradigm is impractical for industrial problems: this approach can only be used temporarily, to guarantee the termination of an algorithm, for instance the computation of some Delaunay triangulation.

In CAD-CAM, exact computation methods are used only locally (when they actually are), for example to calculate Delaunay triangulation from point clouds in 2D or 3D. Most of the rendering pieces of software (Maya, for example) used for virtual movie scenes and video games do not use exact calculation. Quite the opposite, they use processors or GPU, on which the accuracy is rather lower, and therefore gives even less guarantees than a standard floating-point arithmetic processor: speed prevails on accuracy. However, for such applications, a wrong pixel has usually no consequences.

2.2 Interval analysis

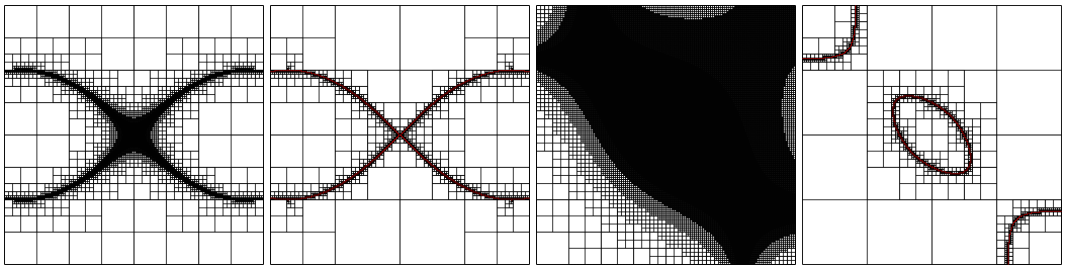

Figure 2. From left to right: Cassini curve with naive interval arithmetic, with Bernstein based arithmetic. A curve due to Martin et al,[9] with naive interval arithmetic, with Bernstein based arithmetic. Bernstein bases enable tighter intervals, and less subdivision.

Another classical approach is interval arithmetic.[10] Interval computations include interval arithmetic and numerical interval analysis. Interval computations ensure that the results of the calculations are correct: the result is an interval guaranteed to contain the exact solution. The main challenge with interval computations is to limit the growth of the width of intervals during the calculations. This growth has two main sources. On the one hand, every multidimensional object is enclosed into a parallelepiped with sides parallel to the axes: this is known as *wrapping effect*. On the other hand, the *dependency problem* is the loss of the dependence between variables. For instance, with naive interval arithmetic, for x in the interval $[0,1]$, the polynomial $x(1-x)$ lies in the interval $[0,1]$, while the exact range is $[0,1/4]$. Even worse, the polynomial $x-x^2$ lies in the interval $[-1,1]$. The first formula, $x(1-x)$ is computed as $\{x_1(1-x_2) : x_1 \in x, x_2 \in x\}$: in other words, the correlation between x_1 and x_2 is lost. Many libraries for interval computation have been proposed, for example Profil/BIAS,

IntLab or Alias. Such libraries are able to solve systems of linear and non-linear equations: typically they output a list of regular boxes, each one containing a unique regular root, and a list of residual boxes, where they can not conclude: for instance some residual boxes containing multiple roots or where the interval evaluation of the formula contains 0 simply because of overestimation. In Computer Geometry and Computer Graphics, interval analysis is classically used to compute guaranteed covers of implicit curves or surfaces:[9] Fig. 2 illustrates in 2D how a recursive subdivision of an initial square in the plane and interval evaluation of the range of a function $f(x,y)$ is used to compute a cover of the curve with equation $f(x,y) = 0$. (The figure also illustrates the importance of the choice of the polynomial basis, when extensions of interval analysis, such as Taylor models methods, are used.) This method easily extends in 3D and to objects described with CSG trees (see Appendix). The interval arithmetics have also been successfully used to compute reliable covers of the attractor of certain functions (for example, the function of Hénon, for the eponymous attractor),[11,12] or periodic orbits, repulsive or attractive, of a given period.

Interval computations are limited by the fact that they cannot reliably find the sign of a number represented by an interval containing 0. More generally, they cannot sort two numbers known by intervals that overlap. This limitation was theorized by constructive analysis (see below). As a consequence, interval computation cannot help in solving problems in Computational Geometry, which require the sign of geometric predicates. Some kind of interval computations is used for filtering: if the two interval bounds of the value orien(A,B,C) -used in a previous example- do not contain 0 and are of the same sign, then the orientation is known, and the use of an expensive exact computation is avoided.

2.3 The lessons from Constructive analysis

Constructive analysis[13] classifies what is computable using arithmetic and what is not. Exact algebraic (or transcendent, for that matter) computations are ruled out. Within constructive analysis, a number is said to be computable if there exists a program that produces a sequence of nested intervals with increasing (and prescribed) accuracy and containing this number.

In theory, the interval bounds are rational numbers. In practice, for programmers, they are often dyadic numbers, of the form $2^{-k}n, n \in \mathbb{Z}, k \in \mathbb{N}$. Not every real number can be represented, because there exists only a countable number of programs and thus a countable number of computable real number. More generally, constructive analysis shows that computable functions must be continuous. This result is intuitively obvious for the function sign: sign(x)= -1 for $x < 0$, sign(0)=0, sign(x)=+1 for $x > 0$. This function is discontinuous at $x = 0$. If the number x is zero, but is represented by a sequence of nested intervals (of non-zero width), its nullity can never be decided. On the other hand, if x is not zero, an interval of the sequence will eventually have its two bounds of the same sign. It is sometimes said that the computation of the nullity of a number, or of the equality of two numbers is half deterministic (or recursively enumerable): nullity and equality are not decidable, whereas non-nullity of a number and non-equality of two numbers are decidable.

Arrangements, Delaunay triangulation, Voronoï diagrams, geometric objects defined by systems of equations or inequalities are not continuous functions of the input and therefore are not computable.

2.4 Probabilistic methods

As previously mentioned, since the beginning of the CAD-CAM industry, numerical problems have existed and being dealt with. CAD-CAM softwares are built to be robust to geometric inconsistencies: such softwares often receive completely inconsistent geometric data like triangles soups: they then use algorithms that do not require geometric consistency, and that, for example, do deal with surfaces with cracks.

A probabilistic solution to the problem of numerical accuracy[14] is illustrated by the following example: computing the shortest path on a surface.[15] Computational geometers have proposed algorithms for triangle meshes, but these algorithms are difficult to use in practice since they assume that the coordinates of vertices are rational. However a practical and robust solution has been proposed: the mesh is sampled by a point cloud. A graph is constructed using these points as vertices, and an edge links close vertices, with distance less than a prescribed threshold. Then the Dijkstra algorithm is used to determine the shortest path between two vertices of the graph. The surface, or mesh, is then re-sampled with a greater density in the vicinity of the shortest path. This method is extremely simple and robust. However it does not provide any guarantee on the result quality. This method may be generalized to compute geodesic cycles with given homology. Software for assembly line planning during manufacture or for finding robots trajectories in environments with obstacles use the same principle of random sampling of the configuration space.

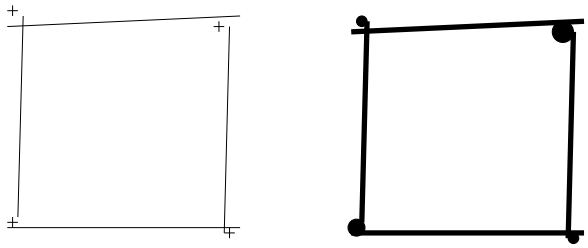

Figure 3. Left: a B-rep of a square, corrupted by (exaggerated) inaccuracy; it has no interior. Right: the toleranced square, thicknesses are exaggerated. Courtesy of Vadim Shapiro.

Computer Graphics also uses the probabilistic paradigm; for instance for radiosity computations, the rendering equation is solved with Monte Carlo integration methods.[16]

Finally the stochastic arithmetic of the CESTAC method proposed by Jean Vignes also relies on a probabilistic argument: the same computations are performed several times, with different rounding modes. They give a set of numeric samples, whose mean and variance can be computed; this way it is possible to estimate the number of probably correct bits in a numeric result, and thus its quality. This stochastic arithmetic is used to detect numerical instabilities during scientific computations: typically these algorithms iteratively converge to a solution, and do not branch in if-then-else tests, apart from the stopping criterion test. The behaviour of the whole algorithm is in this case not much perturbed by the use of stochastic arithmetic.

3. EMBRACING INACCURACY, TOLERANT MODELING

3.1 Tolerant modeling

Practitioners of Geometry Computing consider that the inaccuracy corrupting computations and geometric data structures is unavoidable. To account for it, all CAD-CAM geometric modelers use some kind of tolerant modeling. However each company uses its own peculiar, empirical, non explicit approach, which prevents the inter-operability of geometric softwares and the exchange of geometric data.

Qi and Shapiro[17] proposed a formalization of tolerant modeling. Tolerant modeling, see Fig. 3, attaches a tolerance to each cell of the simplicial complex underlying a B-rep (boundary representation, see Appendix): vertex, linear or curved edge, and surface patch. The thickened element is called a zone. The idea is that tolerances permit to fill possible gaps, for instance between edges and their incident surface patches, or between vertices and their incident edges. Also, while two planar curves, or three 3D surfaces meet at points with coordinates which are generally not representable with floating-point numbers, their corresponding zones meet (generically) in a region with non-zero measure (area in 2D, volume in 3D): this region contains points with floating-point coordinates. Thus these intersections can be represented by an inner approximation, like a box or a ball.

This corresponds to the general practice in CAD-CAM: geometric modelers sample the intersecting curve and they get points with floating-point coordinates which lie not far from this curve. These points represent approximately the curve and are interpolated, typically using spline curves. Tolerant modeling gives a solid theoretical background to this approach, in particular it gives an upper bound to the distance between the interpolated curve and the exact intersecting curve... once a small but still suitable tolerance has been determined.

To get a more detailed representation, interval analysis can be applied to compute a cover of zones, that is an outer approximation. Thus it is possible to sandwich a zone Z between an interval of solids: $I \subset Z \subset O$, where I and O are the inner and the outer representations. Moreover, it is possible to impose an upper bound to the Hausdorff distance between the inner and the outer representations. Another possible data structure is a fair cover of zones: a fair cover is a cover, each piece of which (voxels, balls) is guaranteed to intersect the covered object.

Instead of using one global tolerance per zone, in case of parametric curves or patches, it is possible to attach a tolerance to each control point.[18] The goal is to adjust the tolerance such that for instance a thickened surface and its boundary thickened curve intersect.

A question which arises from tolerant modeling[17] is: is it possible to explicit conditions to guarantee that a toleranced B-rep is valid in a certain sense? If this is possible, this again implies that tolerant modeling gives theoretical justifications to heuristic usage, by giving some guarantees on the representations. For instance, each zone should be simply connected, *ie.* topologically equivalent to a ball, since this property holds for each corresponding cell (vertex, edge, surface patch) in the simplicial complex of the B-rep. The intersection of the zones of two incident cells (a vertex and an incident edge, or an edge and an incident patch) should also be topologically equivalent to a ball. Finally, all relative interiors (a zone minus the zones of its bounding cells) should be disjoint.

Moreover, to make sense, these validity conditions for a toleranced model should be computable with approximate computations, most of the time. Here, the words "most of the time" are used to account for the situations where typically, interval analysis can not decide if two zones intersect or not, *ie.* in cases where they are tangent, or very close to tangency, due to their tolerances and the accuracy of the computations. In such cases, the computations will fail, typically when the topological properties of the inner approximation and of the outer approximation do not match.

Tolerances must be seen as degrees of freedom: the main idea (and the main difficulty) of tolerant modeling is to tune and optimize the zones tolerances, to avoid undecidable or ambiguous situations. A possible direction of research is to check whether tolerances can be optimized using some linear programming method, by expressing constraints such as $\text{dist}(A,B) + \text{threshold} \leq \text{tol}(A) + \text{tol}(B)$ and by minimizing the tolerances. To get such constraints, a first step will be to compute tight enough intervals of the distances between pairs of geometric cells – distance is a continuous function, so it is computable from a tigh enough fair cover, or from a sufficiently accurate interval of solids (*ie.* inner and outer representations).

Next section presents recent methods, whose existence suggests that these validity conditions should be computable in a near future.

3.2 Computing topological properties with interval analysis

Recently, Delanoue et al.,[19,20] and Manocha et al.[21] independently used interval methods to compute in a guaranteed manner the topological properties of geometric objects, described by boolean combinations of nonlinear inequalities (for instance the unit sphere is described by the inequality $x^2+y^2+z^2-1 \leq 0$). Each inequality $f(x,y,z) \leq 0$ describes a geometric primitive.

Both methods recursively subdivide the 3D space into voxels, until the intersection between the studied geometric object and each voxel, its faces and its edges, is simple enough: *ie.* either empty, or full, or contractible to a point: a geometric set S is contractible to a point p when all points of the segments ps lies in S for all $s \in S$. Furthermore, Delanoue et al. remark that p is a star point for $A \cup B$ and for $A \cap B$ if p is a star for A, and for B. This remark implies that it is sufficient to have a contractibility test for primitives. The contractibility test is equivalent to solving a system of nonlinear (usually algebraic) equations.

Delanoue's method constructs a simplicial complex which is guaranteed to be homotopic to the geometric object. It is thus possible to compute the homology group which is a signature of the object, to check that the set is connected, or simply connected (*ie.* it is topologically equivalent to a ball). These questions occur when wanting to prove that a toleranced B-rep is valid, as seen above. Delanoue's method is thus a first step towards an answer.

Both Delanoue's and Manocha's methods require some genericity conditions on the geometric object. Its boundary must be a manifold, *ie.* the boundary is everywhere locally homeomorphic to a disk, as well as the boundary of all geometric primitives. Moreover tangencies between distinct primitives described by inequalities are forbidden: it is due to the fact that equality and nullity are not decidable with interval computations. These methods detect when the genericity conditions are not fulfilled, and thus when they cannot produce a simplicial complex: in such cases, they fail and issue a warning.

Other cases of failure are the following. First, no geometric primitive must be tangent to one of the subdivision planes, since tangential intersections are not computable. A simple cure consists in perturbing randomly the subdivision planes to remove these accidents with probability 1. Finally, the smallest feature size of the geometric objects must be greater than the precision of the floating-point arithmetic. Some techniques based on floating-point arithmetic and yielding extra-precision could be worth considering.

A limitation of these methods is that they do not consider objects whose boundary are parametric surfaces, or thickened parametric surfaces. Nor do they consider objects defined by projections. Actually parametric curves $(x,y,z) = (f(t),g(t),h(t)), 0 \leq t \leq 1$ and parametric surface patches $(x,y,z) = (f(u,v),g(u,v),h(u,v))$, where $(u,v) \in [0,1]^2$, can be considered as projections of 4D or 5D objects on the 3D space *xyz*. The possibility to extend this kind of method to zones of a parametric curve or surface, or in an equivalent manner, to objects defined by projection of a higher dimensional object to 3D space, that is to objects occurring in tolerant modeling remains an open question. Another open question is the generalization of these methods to toleranced objects.

3.3 Remaining issues

This section lists some more remaining issues and hints of solutions.

As seen on Fig. 2, using Bernstein polynomial bases (cf. Appendix) can be a cure to a high number of subdivisions. These bases are still rarely used in interval analysis, whereas CAD-CAM practitioners, who use commonly Bernstein polynomial bases, are still not well acquainted with interval analysis. A potentially useful technique could be to represent the control points by small intervals (with a width of the order of a small multiple of the floating-point precision).

Interval analysis provides sufficient and effective conditions to establish the existence and uniqueness of a root in an interval, besides enclosing it: Brouwer theorem can be applied, Newton-Kantorovitch being the version of this theorem used in the Alias library. Could these conditions be extended, in an effective way, to the uniqueness of the intersection of three surfaces, or to the uniqueness of a whole intersecting curve?

The definition of an interval of solids should take into account further useful features. For instance, in 2D, it is desirable that a curve is sandwiched between two polygonal lines and that its cone of normals (or of its tangent vectors) is also sandwiched. If the cone of normals were sandwiched as well, then determining the uniqueness of the intersection point between two such curves would be easy: it would be sufficient that these two cones were disjoint. Let us notice that the use of Bernstein bases simplifies the determination of such cones.

The list of properties that can be computed using the tolerant modeling approach still waits to be elaborated: it has been seen that several topological properties are within reach to establish that a B-rep is valid, a more extensive list is not yet available. Literature is scarce, concerning the certification of the validity of B-reps.

Finally, a thorough comparison of the various approaches introduced in this paper shoud be led, to compare (in terms of efficiency and guarantee) the different representations: inner-outer polyhedrons, fair covers, voxel or pixel arrays.

4. CONCLUSION

The exact computation paradigm in Computational Geometry ignores inaccuracy and its damages. But this approach has a restricted scope, it can be used only temporarily and locally. In the real world, inaccuracy is unavoidable. In this paper, we propose to embrace inaccuracy with tolerant modeling, which opens a new field for research: how to get guaranteed results with only approximate computations.

Acknowledgments

D. Michelucci thanks Prof. Vadim Shapiro for his hospitality and for inspiring discussions on these topics, during his sabbatical stay at Madison University, Department of Mechanical Engineering, in 2007.

REFERENCES

[1] Preparata, F. and Shamos, M., [*Computational Geometry – An Introduction*], Springer-Verlag, New York, N.Y. (1985).

[2] Boissonnat, J.-D. and Yvinec, M., [*Algorithmic geometry*], Cambridge University Press, New York, NY, USA (1998).

[3] Mehlhorn, K., [*Data structures and Algorithms 3: Multidimensional Searching and Computational Geometry*], Springer-Verlag, Berlin (1984).

[4] Benouamer, M., Michelucci, D., and Péroche, B., "Boundary evaluation using a lazy rational arithmetic," in [*Proc. of the 2nd ACM/IEEE Symp. on Solid Modeling and Applications*], 115–126, ACM Press, Montréal, Canada (1993).

[5] Benouamer, M., Michelucci, D., and Péroche, B., "Error-free boundary evaluation based on a lazy rational arithmetic: a detailed implementation," *Computer-Aided Design* **26**(6), 403–416 (June 1994).
[6] Jaillon, P., *Proposition d'une arithmétique rationnelle paresseuse et d'un outil d'aide à la saisie d'objets en synthèse d'images*, PhD thesis, École Nationale Supérieure des Mines de Saint-Étienne (1993).
[7] Michelucci, D. and Moreau, J.-M., "Lazy Arithmetic," *IEEE Transactions on Computers* **46**, 961–975 (September 1997).
[8] Bouhineau, D., *Constructions automatiques de figures géométriques et programmation logique avec contraintes*, PhD thesis, Université Joseph Fourier, Grenoble (1997).
[9] Martin, R., Shou, H., Voiculescu, I., Bowyer, A., and Wang, G., "Comparison of interval methods for plotting algebraic curves," (2002).
[10] Moore, R., [*Interval Analysis*], Prentice Hall, Englewood Cliffs, N.J. (1966).
[11] Paiva, A., de Figueiredo, L. H., and Stolfi, J., "Robust visualization of strange attractors using affine arithmetic," *Computers & Graphics* **30**(6), 1020–1026 (2006).
[12] Michelucci, D. and Foufou, S., "Interval-based tracing of strange attractors," *Int. J. Comput. Geometry Appl.* **16**(1), 27–40 (2006).
[13] Weihrauch, K., [*Computable Analysis*], Springer-Verlag Berlin/Heidelberg (2000).
[14] Michelucci, D., Moreau, J.-M., and Foufou, S., "Robustness and randomness," in [*Reliable Implementation of Real Number Algorithms: Theory and Practice. LNCS volume [Dagstuhl Seminar 06021, Jan. 2006]*], (5045), Springer-Verlag (2008).
[15] Michelucci, D. and Neveu, M., "Shortest circuits with given homotopy in a constellation," in [*SMA'04: Proc. of the 9th ACM Symp. on Solid Modeling and Applications*], 297–302, Eurographics Association, Aire-la-Ville, Switzerland (2004).
[16] Glassner, A. S., [*Principles of Digital Image Synthesis*], Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1994).
[17] Qi, J. and Shapiro, V., "ε-topological formulation of tolerant solid modeling," *Computer-Aided Design* **38**(4), 367–377 (2006).
[18] Hu, C.-Y., Patrikalakis, N., and Ye, X., "Robust Interval Solid Modelling. Part 1: Representations. Part 2: Boundary Evaluation," *CAD* **28**(10), 807–817, 819–830 (1996).
[19] Delanoue, N., Jaulin, L., and Cottenceau, B., "Guaranteeing the homotopy type of a set defined by nonlinear inequalities," *Reliable Computing* **13**(5), 381–398 (2007).
[20] N. Delanoue, L. J. and Cottenceau, B., "Using interval arithmetic to prove that a set is path-connected," *Theoretical Computer Science, Special issue: Real Numbers and Computers* **351**(1), 119–128 (2006).
[21] Varadhan, G., Krishnan, S., Sriram, T., and Manocha, D., "Topology preserving isosurface extraction for geometry processing," in [*Second Eurographics Symposium on Geometry Processing*], 235–244 (2004).
[22] Farin, G., [*Curves and Surfaces for CAGD: A Practical Guide*], Academic Press Professional, Inc., San Diego, CA (1988).
[23] Garloff, J. and Smith, A. P., "Solution of systems of polynomial equation by using Bernstein expansion," in [*Symbolic Algebraic Methods and Verification Methods*], 87–97, Springer (2001).
[24] Elber, G. and Kim, M.-S., "Geometric constraint solver using multivariate rational spline functions," in [*SMA'01: Proc. of the 6th ACM Symp. on Solid Modeling and Applications*], 1–10, ACM Press, New York, NY, USA (2001).
[25] Mourrain, B. and Pavone, J.-P., "Subdivision methods for solving polynomial equations," Tech. Rep. RR-5658, INRIA (August 2005).

Appendix: Typical models for geometric data

This section lists the most commonly used models for representing geometric objects, and it mentions related problems. Its use is to illustrate on which objects (the representations of geometric data) the problem of inaccuracy applies: coordinates of points, sets of triangles when it is known that they share an edge, coordinates in specific polynomial bases and so on.

- The discrete representations, as 2D or 3D images, are 2D or 3D arrays of elements, pixels in 2D, voxels in 3D. An element can be a grayscale, a color, a density, or an material identifier. Medical devices (scanner, CT, MRI) are major providers of 3D images. The geometry and discrete image analysis algorithms propose to deal with such images. These images can be efficiently represented by quadtrees in 2D, and octrees in 3D. The most common treatments are thresholding, reconstruction of contours, labeling (a voxel is part of the liver, the lungs, etc.), pattern recognition, compression with or without loss to speed up the transfer of images between different devices, signature, and so on. It is possible to apply discrete data structures or methods from discrete geometry temporarily, to continuous geometric problems. For example, precomputing which triangles (or other geometric objects) are present in each

voxel of a grid can accelerate the computation of the rendering. The calculation of a cover of a surface or a curve is another example.

- Meshes, usually triangular meshes, are widely used in computer graphics, but also by CAD-CAM. In CAD-CAM, meshes are often produced from surfaces or high-level descriptions of geometric object (see below). These meshes are generally conformal, that is, they form a simplicial complex: the intersection of two cells (triangles, edges, vertices) of the mesh is a cell of the mesh. Thus, topological computations can be performed on this type of mesh, such as homology groups or cohomology groups (homotopy groups are rarely computed, because the related problems are typically undecidable). More and more, guarantees on facetisation are requested: not only the Hausdorff distance between the mesh and the exact surface object must be less than a prescribed tolerance, but the topology must be respected. The mesh must be topologically equivalent (depending on the case, homeomorphic, or homotopic, or isotopic) to the exact surface.
- Some meshes, typically tetrahedral meshes, partition the volume occupied by the objects, and not only the surface boundary of the object. These meshes are typically used for finite element simulation.
- In 2D, the Delaunay triangulation of a given set of distinct points, called seeds, is a triangulation of the convex hull of the seeds, which vertices are the seeds, and for any triangle, the open circumscribed disc (without the circumscribed circle which is the boundary of the disk) contains no other seed. This definition generalizes in 3D. The 3D Delaunay triangulation is used by reconstruction algorithms: their goal is to reconstruct the continuous surface of the frontier of an object, from a set of sample points on the surface of the object. This problem typically occurs in reverse engineering in CAD-CAM (digitize an object manufactured by a competitor, or a prototype modified manually by an artist), in the preservation and modeling of cultural heritage with the digitization of works of art (statues, paintings, buffers, buildings). Various instruments, or optical probes, are used to sample these items.

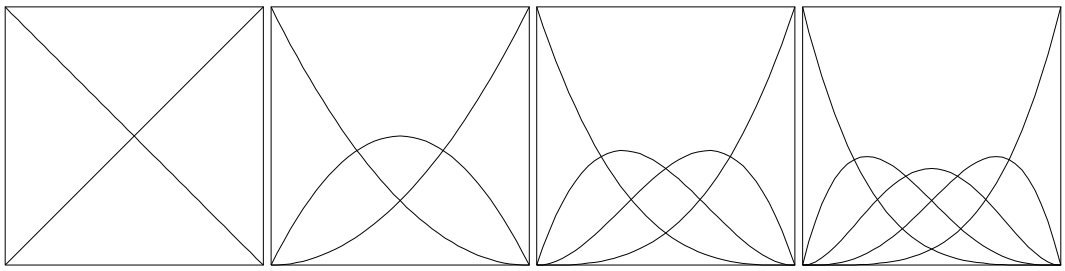

Figure 4. The Bernstein polynomials of degree 1, 2, 3, 4, from left to right. Each square is the unit square. The maximum (or minimum) value of $B_{i,d}$ occurs at $x = i/d$.

- Triangles soups are sets of triangles, approximating an object surface. The triangles may intersect, and are not necessarily connected, leaving gaps between the triangles. In Computer Graphics, a soup of triangles suffices to display an object: each triangle is displayed independently. It also often suffices for the manufacturing, if the accuracy is adapted: the cutting tool (typically a half-sphere, or a rounded cylinder with standardized dimensions) should not be able to enter a gap between two triangles that are supposed to be contiguous. Triangles soups are not simplicial complexes and do not benefit from their properties and their consistency, which prohibits many algorithms. Repairing algorithms attempt to rectify triangle soups, and convert them into meshes; generally an operator is required to interactively guide the software, a sort of 3D PhotoShop.
- Point clouds are object models even more sparse than triangles soups: the surface of objects is known only by sampling points. These clouds are provided by a variety of optical or physical sensors, which scan physical objects (a car door, a statue, etc.). The density of points is supposed to be roughly constant on a unit area on the surface (e.g. 100 to 130 dots per squared inch). It may happen that the same area is sampled by several local point clouds, which partially overlap. The software has to merge these clouds, and reconstruct a surface, such as a triangular mesh, or a set of NURBS (Non Uniform Rational Basis Splines[22]) patches after segmentation of the triangular mesh.

  The points of the cloud may be replaced by surfels: to each point of a cloud is assigned an oriented normal, a color, and possibly a radius when the surfel is considered as a disc on the surface, or even two main components of an ellipse for elliptical patches.

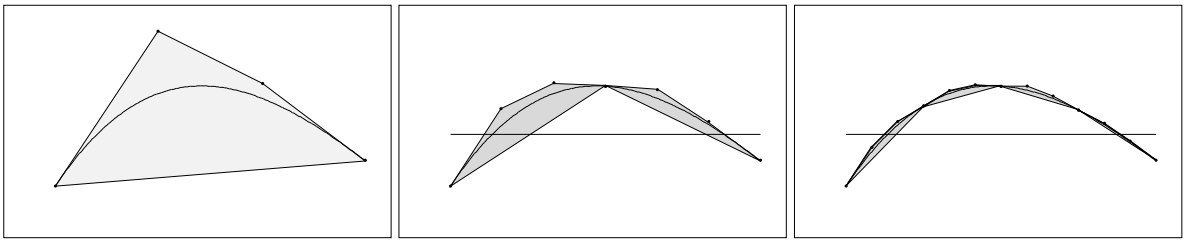

Figure 5. Two de Casteljau iterations on a Bézier curve which is the graph of an univariate polynomial of degree 3. The filled polygons are the convex hulls of the control points.

To solve certain geometric problems, it is more robust to degrade the object geometry in a point cloud or a set of surfels, for example to calculate roads or geodesic paths on a surface. This strategy uses a "probabilistic" approach.

- To interactively define a mesh is difficult. Most people prefer parametric surface representations (Bézier, B-Splines, NURBS) for interactive surface modeling: the user places and moves interactively control points, and the corresponding surface automatically follows. Recently, subdivision surfaces are also used: the user describes a mesh with few vertices, and a subdivision algorithm produces a smooth surface according to a certain scheme. The interest of subdivision surfaces is that they allow greater freedom on the topology of the control net.

- Two kinds of analytical surfaces are typical in CAD-CAM and Computer Graphics: implicit surfaces described by an equation $f(x,y,z)=0$, where f is typically algebraic, *eg.* quadrics, torii, cyclides; and parametric surfaces $x=X(u,v), y=Y(u,v), z=Z(u,v)$ where X,Y,Z are polynomial or rational functions, and u,v are parameters lying in $[0,1]^2$: Bézier surfaces, Splines, NURBS are examples of parametric surfaces. Computer Graphics and CAD-CAM also uses procedural surfaces: subdivision surfaces, and fractal surfaces.

- We will define only Bézier curves and surfaces, since they are typical and at the roots of parametric curves and surfaces. A Bézier curve is expressed as $p(t)=\sum_{i=0}^{d} B_{i,d}(t)p_i$ where $0\leq t\leq 1$: the p_is are 2D or 3D control points, and the $B_{i,d}(t)=\binom{d}{i}t^i(1-t)^{d-i}$ are the Bernstein polynomials of degree d (see Fig. 4 for the Bernstein families of degrees 1 up to 4). They are a base of the polynomials of degree d or less. The conversion between the canonical base: $(1,t,t^2,\ldots t^d)$ and the Bernstein base is a linear mapping, for example, for degree $d=3$, $B_{0,3}(t)=(1-t)^3$, $B_{1,3}(t)=3t(1-t)^2$, $B_{2,3}(t)=3t^2(1-t)$, $B_{3,3}(t)=t^3$ or in matricial form:

$$\begin{pmatrix} B_{0,3}(t)\\ B_{1,3}(t)\\ B_{2,3}(t)\\ B_{3,3}(t)\end{pmatrix}=\begin{pmatrix} 1 & -3 & 3 & -1\\ 0 & 3 & -6 & 3\\ 0 & 0 & 3 & -3\\ 0 & 0 & 0 & 1\end{pmatrix}\begin{pmatrix} 1\\ t\\ t^2\\ t^3\end{pmatrix}\Leftrightarrow\begin{pmatrix} 1\\ t\\ t^2\\ t^3\end{pmatrix}=\begin{pmatrix} 1 & 1 & 1 & 1\\ 0 & 1/3 & 2/3 & 1\\ 0 & 0 & 1/3 & 1\\ 0 & 0 & 0 & 1\end{pmatrix}\begin{pmatrix} B_{0,3}(t)\\ B_{1,3}(t)\\ B_{2,3}(t)\\ B_{3,3}(t)\end{pmatrix}$$

For all $t\in[0,1]$, the Bernstein $B_{i,d}(t)$ are positive and their sum is 1, so $p(t)$ is a linear convex combination of the p_i: $p(t)$ lies inside the convex hull of its control points. Control points are actually coefficients of polynomials expressed in the Bernstein base. Contrarily to coefficients in the canonical base, control points have a very intuitive interpretation: the user can move control points with the mouse and predict the shape of the polynomial from the control polygon $p_i, i=0,d$. Moreover the convex hull property allows to compute tight enclosures of polynomials $p(0\leq t\leq 1)$: it is bounded by the smallest and the greatest coefficient in the Bernstein base. Finally the de Casteljau algorithm bisects Bézier curves 5, *ie.* it computes the half and right control polygons for $0\leq t\leq 1/2$, and for $1/2\leq t\leq 1$. Actually the $1/2$ can be replaced with any constant, even outside $[0,1]$. The de Casteljau algorithm (see Fig. 5) is used to compute covers of Bézier curves, and extends to parametric polynomial or rational curves.

- Bézier surface patches $p(0\leq u\leq 1, 0\leq v\leq 1)$ are defined by tensor product:

$$p(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n} B_{i,m}(u)B_{j,n}(v)p_{ij}$$

The surface patch lies in the convex hull its $(1+m)\times(1+n)$ control 3D points p_{ij}. The convex hull property and the de Casteljau method extend to bivariate and multivariate polynomials. Some numerical solvers of algebraic systems of equations rely on the convex hull properties of tensor Bernstein bases.[23–25]

- Boundary representations (B-reps) are generalizations of meshes, where the embedding of the edges and faces is no longer linear: the planar faces are replaced by Bézier surface patches or other NURBS surface patches.
- Higher level geometric descriptions depend on the applications; clearly medical software, video games and CAD-CAM do not need the same geometric models. However, a high-level description fairly widespread in computer graphics and CAD-CAM are the CSG (constructive solid geometry) trees. The nodes represent affine transformations (translations, rotations, scaling) or deformations (twisting), boolean operations (union, intersection, difference), and the leaves carry primitive shapes (unit cube, sphere, cylinder) or geometric objects described by a systems of equations and inequalities, mostly algebraic (toroids, cyclides, etc). The geometric modelers of CAD-CAM manage two models: a CSG tree, and its corresponding B-rep. Users modify the CSG tree and the software updates incrementally its corresponding B-rep, this operation is called boundaries evaluation and is a generalization of surface meshing methods (for example, boolean operations must be performed). In CAD-CAM, the same guarantees as for the meshing of surfaces are requested: the Hausdorff distance between the calculated frontiers and the exact ones must be less than a prescribed tolerance, and the topology must be respected; for instance, no self-intersection must be introduced. Computer Graphics is less demanding.
- In fact, the previous description is outdated: for fifteen years, all geometric CAD-CAM modelers have been providing one or two additional software layers. These are the parametric modeling and the variational modeling. The CSG trees are parametrized by length, angles, or non-geometric parameters (colors, types and properties of materials, etc).

  The parametric modeling enables the user to interactively modify the values of these parameters. Dynamic geometry software such as Cabri, Cinderella, GeoPlan, GeoSpace have popularized this type of modeling: the user defines a geometric figure typically starting from points; then he or she may define the line or the circle passing through the points previously defined, or intersection points between the lines or circles yet defined. The user specifies in the same time an example and a construction plan, which is stored in some data structure. Then the user can move the starting points with the mouse, and the software updates interactively the whole figure coherently. This allows to see geometric theorems in action: for example, when three vertices are forced to move on the circle circumscribed to three other vertices, then the opposite sides of the hexagon (6 points define 5! possible hexagons) meet at 3 aligned points, after Pascal's theorem.

  In variational modeling, the user no more specifies a geometric construction, but specifies only constraints, for example, distances or angles between vertices or axes, parallelisms, incidences and tangencies. These constraints provide a system of equations, typically algebraic, which are then numerically solved. The most frequently used method is the Newton-Raphson's iteration: the initial guess is the sketch, which is the approximate solution interactively provided by the user. Other used numerical solvers are: gradient descent, homotopy or continuation, interval solvers. Symbolic computations are usually not practical for real world problems.