Finite Precision Elementary Geometric Constructions

Olivier Devillers, Philippe Guigue

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Finite Precision Elementary Geometric Constructions

Olivier Devillers — Philippe Guigue

N° 4559

Septembre 2002

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4559--FR+ENG

INRIA
SOPHIA ANTIPOLIS

Finite Precision Elementary Geometric Constructions

Olivier Devillers , Philippe Guigue

Thème 2 — Génie logiciel
et calcul symbolique
Projets Prisme

Rapport de recherche n° 4559 — Septembre 2002 — 17 pages

Abstract: In this paper we propose a new approach for the robust computation of the nearest integer lattice points of some specific geometric constructions (intersection of two planar segments, circumcenter of a planar triangle and of a spatial tethraedron). Given that the data and the final results of the geometric constructions are stored using single precision floating point representation (typically fixed size integers), the proposed algorithms first perform the geometric construction in IEEE double precision floating point arithmetic, the rounding error is estimated, and only if the error estimation indicates that the result of the floating point computation may be wrong, the computation is repeated with exact arithmetic. The basic advantage is that exact computations are in most cases avoided, thus reducing both the storage and the required computation time.

Key-words: Computational geometry, Robustness, Floating point Arithmetic, Floating point filter.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Constructions géométriques élémentaires à précision fixée

Résumé : Nous proposons dans ce rapport une nouvelle approche pour le calcul robuste du plus proche point de la grille entière de quelques constructions géométriques élémentaires telles que le point d'intersection de segments dans le plan, le centre du cercle circonscrit à un triangle dans le plan ou le centre de la sphère circonscrite à un tétraèdre dans l'espace. Sous l'hypothèse que les données et le résulat final de ces constructions géométriques sont représentés par des nombres flottants IEEE simple précision (typiquement des entiers de taille fixée), les algorithmes proposés calculent dans un premier temps le résultat de la construction à l'aide de l'arithmétique des nombres flottants IEEE double précision, l'erreur d'arrondi est ensuite estimée, et seulement si l'estimation de l'erreur d'arrondi indique que le résultat peut être faux, on demande un calcul plus précis de la valeur à l'aide d'une arithmétique exacte.

L'avantage principal de ce type de méthode est que dans la plupart des cas l'utilisation d'une arithmétique exacte est évitée, l'espace et le temps nécessaires aux calculs sont ainsi réduits.

Mots-clés : Géométrie algorithmique, Robustesse, Arithmétique des nombres flottants, Filtrage numérique.

1 Introduction

Implementing computational geometric algorithms had to face robustness issues, the emerging solution to this problem is based on the *exact geometric computing* paradigm [YD95]. An efficient application of such technique will be obtained if the data has a controlled fixed size representation (e.g. integers represented in single precision). Thus it is important, if algorithms have to be *cascaded* i.e. if we want to use results of algorithms as input of others algorithms, to also produce results in such a fixed size representation. Going from the usual representation of the result to such fixed sized representation will be called *geometric rounding*. By these aspects of fixed sized representation, the current concerns in geometric computing are closed to some traditional aspects of discrete geometry.

Exploiting standard IEEE double precision arithmetic and generalizing floating point filtering technique which applied successfully to evaluation of predicates to the case of rounded constructions, this paper describes very efficient basic geometric constructors for computing the nearest integer lattice point of the intersection point between two line segments, the circumcenter of a triangle and the center of the circumsphere of a tetrahedron.

More precisely, the constructors presented herein use fast double precision approximate computation of the result (intersection point, center,...) and a filtering predicate to certify that this result corresponds actually to the rounding specification. If the certifier failed, the correct rounded result will be either found in a neigborhood, either computed through an exact computation before rounding.

Floating Point Computations In order to understand the significance of the IEEE arithmetic vis-a-vis our robustness goal, it is important to understand some basic properties of such a floating point system. We stipulate to employ floating-point arithmetic conforming to the IEEE 754 standard [Gol91]. In this standard, single precision is encoded in 32 bits using 1 bit for the sign, 8 bits for the exponent, and 23 bits for the significand. However, it uses a hidden bit, so the significand is 24 bits ($p_1 = 24$), even though it is encoded using only 23 bits. Similarly, double precision occupies two consecutive 32 bit words and the significand is 53 bits ($p_2 = 53$). Thus, single and double precision allow to represent p_1-bits and p_2-bits integers respectively.

Throughout this paper, the symbols $\oplus$,$\ominus$,$\otimes$, and $\oslash$ represent p_2-bit IEEE 754 floating-point addition, subtraction, multiplication and division with rounding *to the nearest* (i.e. the computed result is the floating point number that best approximates the exact result). When round to nearest is ambiguous, i.e. when the result is exactly halfway between two representable values, ties are broken by using the IEEE round-to-even digit rule.

We will use mainly one basic property of floating point arithmetic: for all arithmetic operations, $\mathbf{u} = 2^{-p_2}$ is a strict bound to the modulus of the relative error, hence, for $\circ \in \{+, -, \times, /\}$ there exists γ such that $(x \circledcirc y) = (x \circ y)(1 + \gamma)$ and $|\gamma| < \mathbf{u}$. If the relative error in a computation is $n\mathbf{u}$, then the number of affected digits is $\approx \log_2 n$. In particular, additions, subtractions and multiplications performed on pairs of integers smaller than 2^{p_2} are performed exactly as long as the result is also smaller than 2^{p_2}. Most importantly, as double precision carries more than twice as many digits as single precision; we can multiply up to $p_1 + 2$-bit precision numbers exactly by multiplying with double precision.

The algorithms presented herein rely on the assumption that the coordinates of the input (endpoints of the segments, vertices of the tetrahedron) are p_1-bits integers stored in variable of type *float* (i.e. single precision), and the computations are carried out in double precision. This allows to benefit of the fast floating point arithmetic of the processor and to handle more easily overflows.

Geometric Computations The numerical computations of a geometric algorithm are basically of two types each with clearly distinct roles: *predicates* which determines a combinatorial property of geometric objects, and *constructors* which produce a new geometric object from previous objects. Tests (predicates) are associated with branching decisions in the algorithm that determine the flow of control, whereas constructions are needed to produce the output data. If we consider each input data coordinate as a *variable*, a predicate (resp. a constructor) is the sign $-, 0,$ or $+$ (resp. the value) of a rational function whose arguments are a subset of the input variables.

If non-robustness is problematic in purely numerical computation, it is more intractable in *geometric* computations because of the way they closely couple symbolic and numerical data. If the main cause of numerical non-robustness is arithmetic, then it may appear that the problem can be solved with the right kind of arithmetic package. We may roughly divide the approaches into two camps, depending on whether one uses finite precision arithmetic or insists on exactness (arbitrary precision arithmetic). However, no matter which exact numerical representation one chooses, cascading geometric constructions can result in prodigious bit-length growth of coordinate data. More precisely, the space required to represent a point or line segment grows exponentially with the number of operations. As a consequence, the running time and the space consumption of the algorithms increase dramatically when operating with constructed objects. For an instance, Yu [Yu92] concluded that exact rational for 3-D polyhedral modeling is impraticable.

Geometric rounding Although, there exists solutions to avoid exponential cost as tools like LOOK [FM00] which uses floating-point filter techniques on construction level in combination with a lazy evaluation scheme, in most cases a geometric system has to employ rounding schemes after every construction to keep the numerical complexity and space requirements low. Note though, that these rounding schemes actually require a proof that they do not affect the final result considerably. These proofs are non-trivial and usually cannot be generalized. So each application has to be considered separately.

Several methods have been designed in order to embed exact constructions in an integer lattice while preserving some topological properties (alignment of points, cells convexity, ...) which guarantee the algorithm robustness. Satisfactory high-level rounding algorithms are known for polygonal subdivisions in two dimensions and polyhedral subdivisions in three dimensions. Such algorithms deal with the rounding of arrangement of the line segments [GGHT97, GM98], 2D Voronoi diagram [DG99] and arrangement of triangles in 3D [For99]. The general framework of these approaches involves the computation of the nearest lattice point of a basic geometric construction. Typically, *snap rounding* method for rounding arrangement of line segments address the robustness problem by perturbing all vertices of the arrangement to their nearest grid point. The correctness of the algorithms then depends on the underlying elementary geometric constructors which must fulfil the rounding specification. Algorithms for surface reconstruction [AB99, BC01] need to compute cen-

ter of spheres passing through four points in 3D. Using rounded computation without precautions to compute such centers may cause big errors which invalidate these algorithms, the certified rounded computation proposed in that paper is a solution to that problem.

A Fixed Model of Geometry Geometric rounding rounds to a *finite precision lattice*. Precisely speaking, there is a set of sites G in the plane and a cell $\mathtt{cell}(g)$ associated with each site $g \in \mathsf{G}$ such that each point p in the plane lies in exactly one cell. The rounding function maps p to the unique site $g \in \mathsf{G}$ such that $p \in \mathtt{cell}(g)$.

Conforming to exact rounding with tie-breaking round-to-even digit rule, we consider the following finite precision model of geometry: let $\mathsf{R}_{\mathsf{even}} = [-\frac{1}{2}, \frac{1}{2}]$, $\mathsf{R}_{\mathsf{odd}} =]-\frac{1}{2}, \frac{1}{2}[$ and $\lfloor x \rceil$ the rounded value of x. If $i \in \mathsf{G}$, then $\lfloor x \rceil = i$ iff $x \in i + \mathsf{R}_i$, where $+$ denotes the Minkowski sum and R_i equals $\mathsf{R}_{\mathsf{even}}$ or $\mathsf{R}_{\mathsf{odd}}$ depending on the parity of the actual value of i.

Similarly, we may define $\mathsf{R}_{i,j} = \mathsf{R}_i \times \mathsf{R}_j$ and define the square (or pixel) of influence of a grid point $g = (i, j) \in \mathsf{G}$ to be $g + \mathsf{R}_{i,j}$.

For three dimensional geometry, we consider that all representable points must come from the regular 3-dimensional voxel grid. We then define $\mathsf{R}_{i,j,k} = \mathsf{R}_i \times \mathsf{R}_j \times \mathsf{R}_k$ and define the cube (or voxel) of influence of a voxel grid point $g = (i, j, k)$ to be $g + \mathsf{R}_{i,j,k}$.

We can view rounding from the view point of the lattice: all representable points must come from the unit lattice and 2^{p_1} is the maximum magnitude of any coordinate.

Floating point filtering We propose a method for designing exactly rounded robust constructors that reduces the cost of exact arithmetic by appropriately engineering the floating point arithmetic as a filter. The idea is to couple a *heuristic* algorithm using double precision arithmetic which permits to quickly obtain an *often correct* approximation of a given primitive with a certifier that either accepts the result, or says *not sure*: if forward error analysis indicates that the value of the approximate result cannot be trusted, one uses an appropriate algorithm which not necessarily requires arbitrary precision arithmetic in order to provide a certified result. This technique has been successfully used in exact geometric computation [DP98, FV96].

Error bounds can be computed *a priori* (*static filters*) if specific information on the input data is available, (e.g., if all input data are integers from a bounded range) or *on the fly* (*dynamic filters*) i.e. parallel to the evaluation in floating-point arithmetic by using runtime information about the actual values of the variables. The following constructors use *semi-dynamic filters* which partially pre-compute the error bound a priori.

Notations For clarity of exposition, it is convenient to introduce the following notations and definitions: the difference $x_j - x_i$ is represented in double subscript notation x_{ji}; $\tilde{x}$ denotes the rounded value of the expression x i.e. the double precision number obtained by substituting double precision operations for each exact arithmetic operations involved in x. Finally, let $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$ and $p_k = (x_k, y_k)$ be three points in the plane, we define

$$[p_i, p_j, p_k] = \begin{vmatrix} x_i & x_j & x_k \\ y_i & y_j & y_k \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} x_{ji} & x_{ki} \\ y_{ji} & y_{ki} \end{vmatrix} = p_i p_j \times p_i p_k \tag{1}$$

(where $\times$ denotes the cross product) whose sign tells whether the triplet p_i, p_j, p_k represents a right-turn or left-turn, that is, if the point p_k is at the right or at the left of the oriented segment from p_i to p_j (cf. Figure 1) and whose magnitude is twice the area of the triangle p_i, p_j, p_k.

Note that if input coordinates are integers whose absolute value is bounded by 2^{p_1}, this expression is bounded by 2^{2p_1+2} from (1) and needs at most $2p_1 + 2$ bits to be represented, so, double precision arithmetic suffices to perform this computation exactly.

2 Intersection Point of Two Straight Line Segments in Two Dimensions

Problem Given four grid points $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$, $p_k = (x_k, y_k)$ and $p_l = (x_l, y_l)$, decide whether the line segment $s_1 = p_i p_j$ intersects the line segment $s_2 = p_k p_l$ at a unique point, and if so, compute the coordinates (x, y) of the nearest grid point of the exact intersection point. That is, if the exact intersection point is $I' = (x', y')$, then return the unique grid point $\mathbf{i} = (x, y) \in S$ such that $I' \in \mathbf{i} + \mathsf{R}_{x,y}$ (cf. Figure 2).

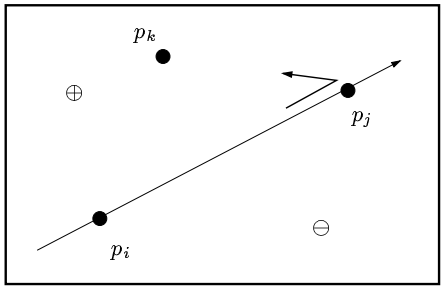

Figure 1: The orient(p_i, p_j, p_k) predicate.

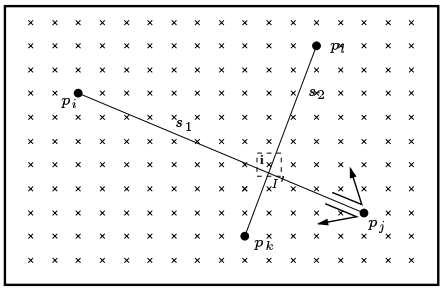

Figure 2: The 2D segment intersection constructor.

To deal more easily with particular cases we assume that the endpoints of each segment are lexicographically ordered, i.e. that $x_i \prec_{xy} x_j$ and $x_k \prec_{xy} x_l$ where $\prec_{xy}$ denotes the xy-lexicographically order. Furthermore we assume without loss of generality that s_1 is lexicographically less than or equal to s_2, $s_1 \preceq_{xy} s_2$ (otherwise we exchange the roles of s_1 and s_2) and that s_1 is not a vertical segment, i.e. $x_i < x_j$. If s_1 is vertical, the same algorithm apply exchanging the role of the x and y coordinates. In practice one should start the algorithm by testing whether the bounding boxes intersect; this test will take care of most calls in typical applications, and it guarantees the preconditions as a side effect. The predicate $s_1 \cap s_2 \neq \emptyset$ can be implemented using the code in Algorithm 1.

Note that, if $x_l < x_j$ and $[p_i, p_j, p_k] = [p_i, p_j, p_l] = 0$, then both segments are parallel and $s_1 \cap s_2 = p_k p_l$, similarly, if $x_l \geq x_j$ and $[p_i, p_j, p_k] = [p_k, p_l, p_j] = 0$ then $s_1 \cap s_2 = p_k p_j$. Thus, after a few coordinate comparisons (degree one), the intersection detection test basically amounts to

compute the sign of only two $(2p_1 + 3)$-bit integers determinants.

The coordinates of the intersection point $\mathbf{i} = s_1 \cap s_2$ are given by

$$\left(x_i + \frac{p_i p_k \times p_i p_l}{p_i p_j \times p_l p_k} x_{ji} \quad , \quad y_i + \frac{p_i p_k \times p_i p_l}{p_i p_j \times p_l p_k} y_{ji} \right) \tag{2}$$

Position Computation errors In the implementation of numerical filters, we need to compute sharp upper bounds on numerical expressions. We recall that we assume that the coordinates of the endpoints of the segments are represented as p_1-bit integers stored as single precision floating-point numbers and that all the computations are carried out in double precision. From (1), $(2p_1 + 2)$-bit arithmetic precision suffice to compute $N = [p_i, p_k, p_l] = p_i p_k \times p_i p_l$ and $D = [p_i, p_j, p_k] - [p_i, p_j, p_l] = p_i p_j \times p_l p_k$ exactly. The expression $\tilde{X} = (x_{ji} \otimes N) \oslash D$ satisfies the following relations:

$$\left| \frac{x_{ji} N}{D} \right| (1 - 2\mathbf{u}) \approx \left| \frac{x_{ji} N}{D} \right| (1 - \mathbf{u})^2 < |\tilde{X}| < \left| \frac{x_{ji} N}{D} \right| (1 + \mathbf{u})^2 \approx \left| \frac{x_{ji} N}{D} \right| (1 + 2\mathbf{u})$$

We obtain

$$|\tilde{X} - \frac{x_{ji} N}{D}| \lesssim 2\mathbf{u} |\tilde{X}|$$

Hence, only the last two significant bit of $\tilde{X}$ at most are incorrect.

Potential roundoff errors arise when $\lceil \tilde{X}(1 - 2\mathbf{u}) \rfloor \neq \lceil \tilde{X}(1 + 2\mathbf{u}) \rfloor$ where $\lfloor \tilde{X} \rceil$ denotes the integer nearest to $\tilde{X}$ (note that the nearest integer of an expression X can be computed by adding and subtracting the value 3.2^{51} to X provided that the absolute value of X is less than 2^{51} and upon use of rounding mode to nearest). The algorithm consists of computing the coordinate as best as possible using double precision and then use a general technique to check the predicate

$$0.5 - |\tilde{X} - \lfloor \tilde{X} \rceil| > 2\mathbf{u} |\tilde{X}| \tag{3}$$

Note that $2\mathbf{u} = 2^{-52}$ is exactly representable, and that the product $2\mathbf{u} \otimes \tilde{X}$ is exact as it involves shifting the exponent only.

If the predicate (3) is satisfied, double-precision computations are enough to get the correct output result, if not, the algorithm may have to resort to arbitrary precision [She97, BKM+95, Gra96] to perform the two error prone operations and to decide which is the nearest integer $\lfloor X \rceil$ to the exact value of X.

Since $\lfloor \tilde{X} \rceil$ and x_i are $(p_1 + 1)$-bits integers, there is no error in the final addition.

Remark: A similar result has been obtained by Boissonnat and Preparata [BP00] for points with integer coordinates. More precisely, they proved that if the endpoints of the segments are p_1-bits integers, then the coordinates of an intersection point can be rounded to one of its two nearest single

precision integers using only double precision floating-point arithmetic operations. Priest [Pri92] has proposed a stronger result for points with floating-point coordinates. His algorithm uses double precision floating-point arithmetic and rounds s to the nearest single precision floating-point number. Both results imply the following monotonicity property: $e <_x \tilde{s} \Rightarrow e <_x s$ where e is an endpoint, s is an intersection point, and $\tilde{s}$ is the corresponding rounded point, which is sufficient for geometric rounding method like [Mil00]. Yet, they don't necessarily solve the intersection problem as stated previously.

Contrary to the assertions of Priest in [Pri92], taking the quotient of the approximations of the numerator and the denominator accurate to double precision gives coordinates which are accurate to all but the (at most) last three significant bits and then not always sufficiently accurate to round to the correct single precision quantity. For example, with

$$p_i = \begin{pmatrix} -4\alpha + 12 \\ 3\alpha - 7 \end{pmatrix}, \quad p_j = \begin{pmatrix} 4\alpha \\ 3\alpha + 1 \end{pmatrix}, \quad p_k = \begin{pmatrix} 3\alpha - 3 \\ -4\alpha + 7 \end{pmatrix}, \quad p_l = \begin{pmatrix} 3\alpha + 1 \\ 4\alpha - 1 \end{pmatrix}$$

and if α is chosen such that both segments intersect, the abscissa x' of the intersection point of segments p_ip_j and p_kp_l is equal to $x' = 3\alpha + \frac{1}{2} - \frac{1}{2\alpha^2 - 5\alpha + 2}$.

With $\alpha = 2^{22} - 1$, taking the quotient of the computed values accurate to double precision of the numerator and the denominator gives $x = 12582909.5$. Converting this number, with tie-breaking rule to even, to single precision number (that is to the nearest integer in this case) gives 12582910 which is different of the nearest single precision number to x' which is 12582909. The 2D segments intersection point constructor in Algorithm 2 fixes the error for integer inputs.

3 Circumcenter of a Triangle in Two Dimensions

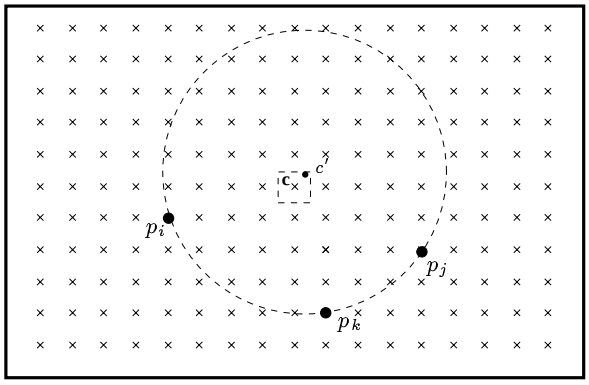

Figure 3: The circumcenter(p_i, p_j, p_k) constructor.

Problem Given three grid points $p_i = (x_i, y_i), p_j = (x_j, y_j), p_k = (x_k, y_k)$, compute the coordinates (x, y) of the nearest grid point of the circumcenter. That is, if the exact center of the unique circle passing through p_i, p_j and p_k is $c' = (x', y')$, then return the unique grid point $\mathbf{c} = (x, y) \in S$ such that $c' \in \mathbf{c} + \mathsf{R}_{x,y}$ (cf. Figure 3).

If p_i, p_j and p_k are not collinear, the coordinates of the circumcenter of a triangle $p_i p_j p_k$ in the plane may be written as

$$\left(x_i - \frac{\begin{vmatrix} y_{ji} & ||p_{ji}||^2 \\ y_{ki} & ||p_{ki}||^2 \end{vmatrix}}{2\begin{vmatrix} x_{ji} & y_{ji} \\ x_{ki} & y_{ki} \end{vmatrix}}, y_i + \frac{\begin{vmatrix} x_{ji} & ||p_{ji}||^2 \\ x_{ki} & ||p_{ki}||^2 \end{vmatrix}}{2\begin{vmatrix} x_{ji} & y_{ji} \\ x_{ki} & y_{ki} \end{vmatrix}} \right) \tag{4}$$

where $||p_{ji}||$ denote the Euclidean norm of $p_j - p_i$.

Position Computation Errors The denominator D is a degree two polynomial in the input variables and is computed exactly with double precision arithmetic if the input is p_1-bits integers. Expanding the numerator determinant Δ clearly shows that it is a degree three computation. More precisely, the coefficients of the first and the second column are respectively $(p_1 + 1)$ and $(2p_1 + 3)$-bit integers, hence, with $\tilde{N}_+ = |y_{ji}| \otimes ||p_{ki}||^2 \oplus |y_{ki}| \otimes ||p_{ji}||^2$

$$|\tilde{\Delta} - \Delta| \leq (2\mathbf{u} + \mathbf{u}^2)\tilde{N}_+$$

an upper bound on forward error of the rational expression is

$$\begin{aligned} |\frac{\Delta}{D} - (\tilde{\Delta} \oslash D)| &\leq \frac{(2\mathbf{u} + \mathbf{u}^2)\tilde{N}_+}{|D|} + \mathbf{u}|\tilde{\Delta} \oslash D| \\ &\leq (1 + \mathbf{u})(2\mathbf{u} + \mathbf{u}^2)|\tilde{N}_+ \oslash D| + \mathbf{u}|\tilde{\Delta} \oslash D| \\ &\leq (3\mathbf{u} + 3\mathbf{u}^2 + \mathbf{u}^3)|\tilde{N}_+ \oslash D| \end{aligned}$$

So, the product $4\mathbf{u} \otimes |\tilde{N}_+ \oslash d|$, which just involves shifting the exponent, is a strict bound on forward error of the rational polynomial. As in the previous primitive, there is no error in the final addition.

This bound apply even if the circumcenter is outside the range of integers, but in such a case exact computation may need different tools. Pseudo-code of the circumcenter constructor is given in Algorithm 3.

4 Circumcenter of a Tetrahedron in Three Dimensions

Problem Given four grid points in three dimensions $p_i = (x_i, y_i, z_i)$, $p_j = (x_j, y_j, z_j)$, $p_k = (x_k, y_k, z_k)$ and $p_l = (x_l, y_l, z_l)$, compute the coordinates (x, y, z) of the nearest grid point of the

center of the circumsphere. That is, if the exact center of the unique sphere passing through p_i, p_j, p_k and p_l is $c' = (x', y', z')$, then return the unique grid point $\mathbf{c} = (x, y, z) \in S$ such that $c' \in \mathbf{c} + \mathsf{R}_{x,y,z}$.

The coordinates of the center of the circumsphere of the tetrahedron $p_i p_j p_k p_l$ may be written as

$$p_i + \frac{||p_{li}||^2 q_{jk} + ||p_{ki}||^2 q_{lj} + ||p_{ji}||^2 q_{kl}}{2 \begin{vmatrix} x_{ji} & y_{ji} & z_{ji} \\ x_{ki} & y_{ki} & z_{ki} \\ x_{li} & y_{li} & z_{li} \end{vmatrix}} \tag{5}$$

where $q_{jk} = p_{ji} \times p_{ki}$, $q_{lj} = p_{li} \times p_{ji}$ and $q_{kl} = p_{ki} \times p_{li}$.

Notes on stability The above expressions (2), (4) and (5) incur roundoff error proportional to the *differences* between vertices, but do not incur roundoff error proportional to the *absolute* coordinates of the vertices until the final additions. The advantage is that in most applications, vertices are usually nearer to each other than to the origin. When operations are implemented in floating point, then fewer bits are lost to rounding, increasing the probability of correct approximations.

Position Computation Errors For this particular primitive, the computation of a sharp upper bound on the numerical expressions for the coordinates is not as easy as in previous primitives since the rounding error ϵ_D of a straightforward computation of the denominator is not necessarily proportional to its double precision approximation $\tilde{D}$. Thence, one cannot obtain a lower bound of $|\tilde{D} - \epsilon_D|$. However, using a splitting technique, it is possible to compute the denominator accurate to double precision number (24 elementary operations instead of 6). Furthermore, this ensures $\tilde{D} = 0$ iff $D = 0$ since $\epsilon_D \leq \mathbf{u}\tilde{D}$.

The numerator of the rational polynomial is a sum of three terms having degree four. An upper bound on the absolute error of its approximate computation using double precision arithmetic is obtained when summing the round off error of the three multiplications with the two additions. This gives:

$$\begin{aligned} |\tilde{N} - N| &\leq (3\mathbf{u} + 2\mathbf{u}^2) \left(\left(\left| ||p_{li}||^2 \otimes q_{jk} \right| \oplus \left| ||p_{ki}||^2 \otimes q_{lj} \right| \right) \oplus \left| ||p_{ji}||^2 \otimes q_{kl} \right| \right) \\ &= (3\mathbf{u} + 2\mathbf{u}^2)\tilde{N}_+ = \epsilon_N \end{aligned}$$

Hence, an upper bound on the entire geometric primitive is given by the following calculation

$$\begin{aligned}
|\tilde{X} - X| &\leq \frac{\epsilon_N|\tilde{D}| + \epsilon_D|\tilde{N}|}{|\tilde{D}|(|\tilde{D}| - \epsilon_D)} + \mathbf{u}|\tilde{N} \oslash \tilde{D}| \\
&\leq \frac{(3\mathbf{u} + 2\mathbf{u}^2)\tilde{N}_+}{(1-\mathbf{u})|\tilde{D}|} + \frac{\mathbf{u}|\tilde{N}|}{(1-\mathbf{u})|\tilde{D}|} + \mathbf{u}|\tilde{N} \oslash \tilde{D}| \\
&\leq \frac{\mathbf{u}}{1-\mathbf{u}}(5 + 5\mathbf{u} + 2\mathbf{u}^2)|\tilde{N}_+ \oslash \tilde{D}| \\
&< (5\mathbf{u} + 10\mathbf{u}^2 + 7\mathbf{u}^3 + 2\mathbf{u}^4)|\tilde{N}_+ \oslash \tilde{D}| < 8\mathbf{u}|\tilde{N}_+ \oslash \tilde{D}|
\end{aligned}$$

This analysis leads to the 3D circumcenter constructor in Algorithm 4.

5 Implementation

Implementation has been done using C++, and the GNU C++ compiler on a Pentium III 1GHz. Times have been obtained using the `gettimeofday` command.

The presented codes (denoted *filtered* in the tables) have been compared with the direct floating-point computation (*double*), with the long int (*int32*), the long long int (*int64*) integer computations provided by the compiler, and with the *rational* of LEDA (release 4.2) which provides exact computation. These implementations allow the following precision on the entries (i.e. the maximum bit precision allowed by computation to keep correct results) for the following geometric constructors:

constructor	segment_2d_intersection	circumcenter_2d	circumcenter_3d
double	16	16	11
int32	9	8	6
int64	20	19	14

The adjective *not robust* (resp. *robust*) is added for codes that give a wrong (resp. correct) results in degenerate case.

The presented results have been obtained on the following kinds of entries:
Prand-N - constructions with pseudo random integer entries (using the `random` command) uniformly distributed in $]-2^N, 2^N[$.

Values are obtained by averaging on about 10^8 constructions. The segment intersection constructor is tested with randomly choosen 2D segments that intersect. The circumcenter2d and circumcenter3d constructors are tested with 2D and 3D points uniformly distributed in the open square with side length 2^{N+1} centered at the origin.

segment_2d_intersection	Time usec.	Avg. % filter failed
Prand-9		
double (robust)	0.1983	
int32 (robust)	0.2957	
filtered (robust)	0.3873	0.0315%
Prand-20		
double (not robust)	0.1988	
filtered (robust)	0.3885	0%
int64 (robust)	0.7459	
Prand-24		
double (not robust)	0.2023	
filtered (robust)	0.4160	0%
rational (exact)	44.6120	

Table 1: Execution times of segment2d_segment2d_intersection in micro-seconds

circumcenter_2d	Time usec.	Avg. % filter failed
Prand-8		
double (robust)	0.2042	
filtered (robust)	0.4176	1.0579%
int64 (robust)	0.72770	
Prand-19		
double (not robust)	0.2044	
filtered (robust)	0.2847	0.00066%
int64 (robust)	0.7638	
Prand-24		
double (not robust)	0.1999	
filtered (robust)	0.2998	0%
leda rational (exact)	32.8064	

Table 2: Execution times of circumcenter2d in micro-seconds

circumcenter_3d	Time usec.	Avg. % filter failed or % Bad answers
Prand-14		
double (not robust)	0.3275	
filtered (robust)	0.7327	0%
int64 (robust)	1.6271	
Prand-24		
double (not robust)	0.3450	0.00018%
filtered (robust)	0.7747	0.000016%
leda rational (exact)	82.4527	

Table 3: Execution times of circumcenter3d in micro-seconds

6 Conclusion

We have illustrated in this paper techniques to build efficient rounded geometric constructors. This kind of rounded constructions is the basis for the design of robust geometric algorithms based on the *exact computation paradigm* [YD95].

By a precise and deterministic specification of the result of a rounded construction in a way similar to the specification of IEEE rounded arithmetic, we allow to build, on these constructions, new algorithms and prove their validity. To some extent, this point of view is an alternative to the full *exact computation paradigm*: exact computation guarantee robustness but increase the memory size needed to represent exactly the geometric objects; what we advocate is to use exact computation "locally" to ensure correctness and to round results in a well specified manner to control the memory size.

We have illustrated our approach on the design of two constructors: the line segment intersection constructor which is the basic constructor for the construction of arrangement with clear application in visualisation or in solid modelling, and the circumcenter constructor which is used in surface reconstruction.

Our approach of constructing an approximated result and certified it as conforms to the specifications has been proved to be much more efficient than a pure integer implementation (cf Table 1,2,3).

References

[AB99] Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. *Discrete Comput. Geom.*, 22(4):481–504, 1999.

[BC01] Jean-Daniel Boissonnat and Frédéric Cazals. Coarse-to-fine surface simplification with geometric guarantees. In A. Chalmers and T.-M. Rhyne, editors, *Eurographics'01*, Manchester, 2001. Blackwell.

[BKM+95] Christoph Burnikel, Jochen Könnemann, Kurt Mehlhorn, Stefan Näher, Stefan Schirra, and Christian Uhrig. Exact geometric computation in LEDA. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C18–C19, 1995.

[BP00] Jean-Daniel Boissonnat and Franco P. Preparata. Robust plane sweep for intersecting segments. *SIAM J. Comput.*, 29:1401–1421, 2000.

[DG99] Olivier Devillers and Pierre-Marie Gandoin. Rounding Voronoi diagram. In *Proc. 8th Discrete Geometry and Computational Imagery conference (DGCI99)*, volume 1568 of *Lecture Notes in Computer Science*, pages 375–387. Springer-Verlag, 1999.

[DP98] O. Devillers and F. P. Preparata. A probabilistic analysis of the power of arithmetic filters. *Discrete and Computational Geometry*, 20:523–547, 1998.

[FM00] Stefan Funke and Kurt Mehlhorn. Look: A lazy object-oriented kernel for geometric computation. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 156–165, 2000.

[For99] S. Fortune. Vertex-rounding a three-dimensional polyhedral subdivision. *Discrete Comput. Geom.*, 22(4):593–618, 1999.

[FV96] S. Fortune and C. J. Van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Trans. Graph.*, 15(3):223–248, July 1996.

[GGHT97] M. Goodrich, L. J. Guibas, J. Hershberger, and P. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 284–293, 1997.

[GM98] Leonidas Guibas and David Marimont. Rounding arrangements dynamically. *Internat. J. Comput. Geom. Appl.*, 8:157–176, 1998.

[Gol91] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991.

[Gra96] Torbjörn Granlund. *GMP, The GNU Multiple Precision Arithmetic Library*, 2.0.2 edition, 1996. http://www.swox.com/gmp/.

[LM98] Vincent Lefévre and Jean-Michel Muller. The table maker's dilemma. *IEEE Trans. Comput.*, 47(11), 1998.

[Mil00] V. J. Milenkovic. Shortest path geometric rounding. *Algorithmica*, 27(1):57–86, 2000.

[Pri92] D. Priest. *On properties of floating point arithmetics: numerical stability and the cost of accurate computations*. Ph.D. thesis, Dept. of mathematics, Univ. of California at Berkeley, 1992.

[She97] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete Comput. Geom.*, 18(3):305–363, 1997.

[YD95] C. K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition, 1995.

[Yu92] Jiaxun Yu. Exact arithmetic solid modeling. Technical Report CSD-TR-92-037, Comput. Sci. Dept., Purdue University, June 1992.

`do_2dsegments_intersect`(p_i, p_j,p_k, p_l) {
/* decides whether line segments p_ip_j and p_kp_l intersects.
Precondition: p_ip_j is xy-lexicographically less than or equal to p_kp_l, p_ip_j is not vertical */

if ($p_j.x < p_k.x$) then return **false**
/* Segments s_1 and s_2 can intersect only if their x-ranges do intersect */
if ($p_l.x < p_j.x$)
  if ($[p_i, p_j, p_k] \otimes [p_i, p_j, p_l] <= 0$) then return **true** else return **false**
else
  if ($[p_i, p_j, p_k] \otimes [p_k, p_l, p_j] >= 0$) then return **true** else return **false**
}

Algorithm 1: 2D segment-2D segment intersection test

double $C = 3 \otimes 2^{51}$;
double `round_to_int`(**double** x) { return $((C \oplus x) \ominus C)$;}

`segment2d_segment2d_intersection`(p_i, p_j, p_k, p_l) {
/* **Precondition:** p_ip_j and p_kp_l intersect at an unique point */

N = $[p_i, p_k, p_l]$;
D = $(x_j \ominus x_i) \otimes (y_k \ominus y_l) \oplus (x_l \ominus x_k) \otimes (y_j \ominus y_i)$;
$x = ((x_j \ominus x_i) \otimes N) \oslash D$;

$\tilde{x} =$ `round_to_int`(x);
/* 2 addition */

if $0.5 \ominus |x \ominus \tilde{x}| > 2\mathbf{u} \otimes |x|$
/* 2 absolute values, 2 additions, 1 product , 1 comparison */

then return `round_to_int`$(x_i \oplus x)$;
  /* 3 additions */

else /* Compute the exact sign of $E = \left(\frac{2(x_j - x_i)N}{D} - \texttt{round\_to\_int}(2x)\right)$
  with arbitrary precision arithmetic */
  return `round_to_int`$(x_i \oplus 0.5 \otimes (\texttt{round\_to\_int}(2 \otimes x) \oplus \texttt{sign}(E)))$;
  /* 12 multiplications, 20 additions, 2 signs */
}

Algorithm 2: 2D segment - 2D segment intersection point (x-coordinate only).

```
circumcenter2d(p_i, p_j, p_k) {

xji = p_j.x ⊖ p_i.x; yji = p_j.y ⊖ p_i.y;
xki = p_k.x ⊖ p_i.x; yki = p_k.y ⊖ p_i.y;
d = xji ⊗ yki ⊖ xki ⊗ yji;

if (d == 0) then /* The three points are collinear */
else
        n1 = yji ⊗ (xki ⊗ xki ⊕ yki ⊗ yki);
        n2 = yki ⊗ (xji ⊗ xji ⊕ yji ⊗ yji);
        c = (n1 ⊖ n2) ⊘ d;

        c̃ = round_to_int(c);
        /* 2 additions */

        if 0.5 ⊖ |c ⊖ c̃| > 4u ⊗ |(|n_1| ⊕ |n_2|) ⊘ d|
        /* 1 division, 4 absolute values, 3 additions, 1 product , 1 comparison */

        then return round_to_int(x_i ⊖ c)
                /* 3 additions */

        else /* Compute the exact sign of E = (2(n1−n2)/d − round_to_int(2c)) */
                return round_to_int(x_i ⊖ 0.5 ⊗ (round_to_int(2 ⊗ c) ⊕ sign(E)))}
}
```

Algorithm 3: 2D circumcenter (x-coordinate only)

`circumcenter3d`(p_i, p_j, p_k, p_l){

$xcross_{jk} = y_{ji} \otimes z_{ki} \ominus z_{ji} \otimes y_{ki}$;
$xcross_{lj} = y_{li} \otimes z_{ji} \ominus z_{li} \otimes y_{ji}$;
$xcross_{kl} = y_{ki} \otimes z_{li} \ominus z_{ki} \otimes y_{li}$;
// Computation of denominator d accurate to double precision number (24 operations)

if (d == 0) **then** /* The four points are coplanar. */
else
$n_1 = ||p_{li}||^2 \otimes xcross_{jk}$;
$n_2 = ||p_{ki}||^2 \otimes xcross_{lj}$;
$n_3 = ||p_{ji}||^2 \otimes xcross_{kl}$;
$n = n_1 \oplus n_2 \oplus n_3$;
$c = 0.5 \oslash (n \oslash d)$;

$\tilde{c} =$ `rount_to_int`(c);
$n_+ = 0.5 \otimes (|\tilde{n_1}| \oplus |\tilde{n_2}| \oplus |\tilde{n_3}|)$.
/* 3 absolute values, 4 additions, 1 product */

if $(0.5 \ominus |c \ominus \tilde{c}| > 8\mathbf{u} \otimes |n_+ \oslash d|)$
/* 1 division, 2 absolute values, 2 additions, 1 product */
then return `round_to_int`$(x_i \oplus c)$
/* 3 additions */
else
/* Compute the exact value of c (with GMP[Gra96] for example) */
return `round_to_int`$(x_i \oplus c)$
}

Algorithm 4: 3D Circumcenter (x-coordinate only)

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399