



A Robust Predicate Library for Delaunay Triangulation on Mixed-Precision Hardware

Anahit Ellery^{1*}, Neha Venkatesan², Fernanda Yamamoto³

¹Department of Mathematics, Thames Graduate Institute of Mathematics, Serbia

²Department of Mathematics, Bharathapuzha Graduate Institute of Mathematics, Netherlands

³Department of Mathematics, Chaliyar Graduate Institute of Mathematics, Bulgaria

Article Info

Volume 1, Issue 1, 2026

[doi: 10.62715/ijcmg.2026.v1.i1.pp13-18](https://doi.org/10.62715/ijcmg.2026.v1.i1.pp13-18)

Received: 27 April 2026

Accepted: 25 June 2026

Published: 17 August 2026

Abstract

Delaunay triangulation depends on two sign computations, the orientation test and the in-circle or in-sphere test, and an incorrect sign can make an otherwise correct algorithm loop, crash or return an invalid mesh. Robust predicates for conventional double-precision processors have been available since the 1990s. They assume that binary64 arithmetic with round-to-nearest is the fast native format, an assumption that no longer holds on graphics processors and accelerators where single or half precision is the efficient format, double precision is slow or emulated, and fused multiply-add and subnormal handling vary between devices. This article proposes the design of a predicate library for that setting. It consists of an explicit input contract and a cascade of four stages: a semi-static filter in the narrow native format, a filter in double precision that exploits exact products of narrow operands, adaptive evaluation with floating-point expansions built from error-free transformations, and an exact fallback, followed by symbolic perturbation for degenerate inputs. Each stage is described with the condition under which it certifies a sign and the hardware property it relies on. A verification plan is included. The design is a proposal assembled from published techniques and has not been implemented or benchmarked here.

Keywords: Geometric predicates, Delaunay triangulation, Floating-point filters, Mixed-precision arithmetic, Error-free transformations, Exact geometric computation

© 2026 Anahit Ellery et al. This is an open access article under the CC BY license (<https://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

1. Introduction

A triangulation of a point set is Delaunay when no point lies inside the circumscribing circle, or sphere, of any simplex, the empty sphere condition introduced by [Delaunay \(1934\)](#). Algorithms that construct such triangulations, whether incremental, divide-and-conquer or flip-based, take all their geometric decisions through two predicates ([Guibas and Stolfi, 1985](#)). The orientation test asks on which side of a line or plane a point lies. The in-circle or in-sphere test asks whether a point lies inside, on or outside the circle or sphere through the others. Both are the sign of a small determinant of the input coordinates.

Evaluated in floating-point arithmetic, such a determinant can come out with the wrong sign when its

* Corresponding author: Anahit Ellery, Department of Mathematics, Thames Graduate Institute of Mathematics, Serbia. E-mail: anahit.ellery@thames.edu

© 2026 Anahit Ellery et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

true value is close to zero. The consequences are out of proportion to the size of the error. [Kettner et al. \(2008\)](#) exhibit small inputs on which a textbook incremental algorithm, run with a naive floating-point orientation test, produces a wrong result or fails to terminate. The accepted cure is the exact geometric computation paradigm: the predicates must return the sign of the exact determinant, whatever arithmetic is used internally ([Yap, 1997](#)).

The best known implementation of that paradigm is the set of adaptive predicates of [Shewchuk \(1997\)](#), written for processors on which double precision is the native format. Triangulation codes now also run on graphics processors and other accelerators, where the fast format is narrower and the arithmetic environment is less uniform. This article proposes a design for a predicate library suited to such mixed-precision hardware. It describes a method and the reasoning behind it. No implementation is evaluated, and no performance claims are made.

2. Background

The design combines three bodies of established work: the analysis of rounding error in the predicates, the filtering techniques that avoid exact arithmetic in easy cases, and the error-free transformations from which exact arithmetic can be built out of ordinary floating-point operations.

2.1. Rounding and the IEEE formats

The IEEE 754 standard defines binary formats with significands of 11, 24 and 53 bits, known as binary16, binary32 and binary64, and requires that each basic operation return the correctly rounded result ([IEEE, 2019](#)). Correct rounding is what makes rigorous error bounds possible: the computed result of one operation differs from the exact one by at most half a unit in the last place, and bounds for a whole expression follow by the standard techniques described by [Higham \(2002\)](#). [Goldberg \(1991\)](#) remains a clear account of where the guarantees end, for example when a compiler keeps intermediate values in a wider register or reorders operations.

2.2. Filters

A filter evaluates the determinant in fast arithmetic together with a bound on the accumulated error, and accepts the sign when the magnitude of the computed value exceeds the bound. Static filters derive the bound before run time from assumed limits on the coordinates, as in the integer-coordinate analysis of [Fortune and Van Wyk \(1996\)](#). Semi-static filters scale a precomputed constant by the magnitude of the actual operands. Dynamic filters compute the bound alongside the value, usually with interval arithmetic ([Brönnimann et al., 2001](#)). [Dewillers and Pion \(2003\)](#) show how these are layered for Delaunay predicates so that almost all calls are settled by the cheapest test. The general principle, cheap tests first and costly ones only on failure, is often called cascading.

2.3. Error-free transformations and expansions

The sum of two floating-point numbers can be written exactly as a floating-point result plus a floating-point error term, and the error term can be computed with a few further additions. One such algorithm is due to Knuth ([Knuth, 1997](#)), and a faster one that requires the operands to be ordered by magnitude, together with a splitting technique that gives the analogous result for products, is due to [Dekker \(1971\)](#). Chaining these transformations yields expansions, sums of floating-point components that represent a number exactly, an idea developed by [Priest \(1991\)](#) and made adaptive by [Shewchuk \(1997\)](#). Where a fused multiply-add is available, the error of a product costs a single extra operation ([Muller et al., 2018](#)).

3. What Changes on Mixed-Precision Hardware

Four assumptions behind the classical predicates need to be revisited. The first is the working format. Coordinates on an accelerator are commonly stored in binary32, sometimes in binary16 for compressed geometry, and double-precision operations may be markedly slower or absent. The second is the fused multiply-add. It changes the rounding of expressions that a compiler contracts silently, which can

invalidate an error bound derived for separate multiplication and addition. The third is the treatment of subnormal numbers, which some devices flush to zero by default, breaking the exactness of error-free transformations near the underflow threshold. The fourth is control flow: data-parallel execution penalizes branches, so that a cascade in which different inputs leave at different stages has to be organized differently from a scalar routine.

Mixed precision also offers an opportunity that uniform double precision lacks. The product of two binary32 numbers has at most 48 significant bits and is therefore exactly representable in binary64, barring overflow and underflow. Narrow inputs evaluated in a wider format thus lose nothing in their first multiplications. This is the principle on which mixed-precision methods in numerical linear algebra are built ([Higham and Mary, 2022](#)), and it can be used in geometric predicates in the same way.

4. The Proposed Design

The library is organized as an input contract followed by four evaluation stages and a tie-breaking rule. Figure 1 shows the arrangement, and Table 1 states for each stage the arithmetic it uses, the condition under which it returns a certified sign, and the hardware property on which its correctness depends. A call descends to the next stage only when the current one cannot certify the sign.

4.1. Input contract

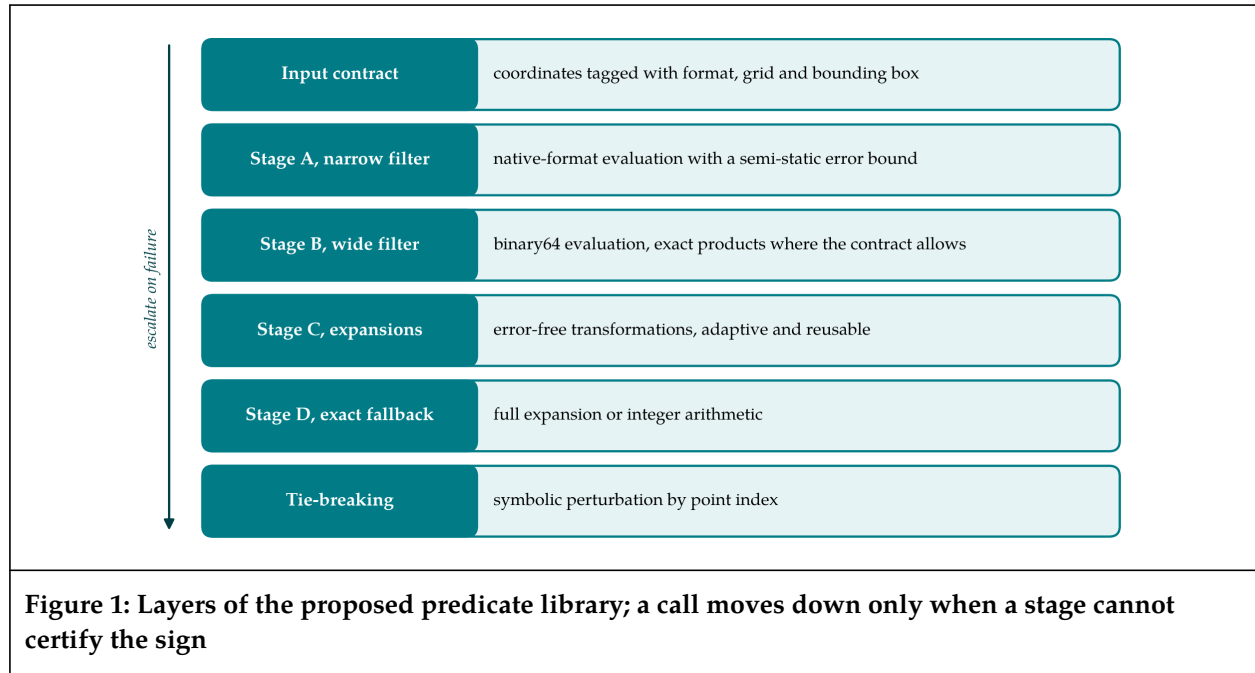
Each point set passed to the library carries a descriptor stating the format of its coordinates and, optionally, a bounding box and a grid spacing. The descriptor selects the constants used by the filters. If the coordinates are known to be multiples of a grid spacing within a bounded box, they are scaled integers of known bit length, and the bit length required by each determinant is known in advance: roughly twice the coordinate length for the planar orientation test and four times for the in-circle test, plus a few guard bits. Inputs without such guarantees are treated as arbitrary values of their format. Not-a-number and infinite coordinates are rejected at this point and never inside a predicate.

Table 1: Stages of the proposed predicate cascade

Stage	Arithmetic	Certifies the sign when	Relies on
A. Narrow filter	Native format, straight-line code	Computed magnitude exceeds a semi-static bound	Correct rounding; no contraction
B. Wide filter	binary64, or double-word arithmetic where binary64 is absent	Magnitude exceeds a tighter bound	Exact products of narrow operands
C. Adaptive expansions	Error-free transformations in the widest native format	Partial expansion exceeds its remaining error bound	Round-to-nearest; subnormals kept
D. Exact fallback	Full expansions or multiword integers	Always, including a true zero	Integer or exact floating-point operations
Tie-breaking	Comparison of point indices	Exact determinant is zero	Consistent global indexing

4.2. Stage A: semi-static filter in the native format

The determinant is evaluated in the fast narrow format by branch-free code, after translating the points so that one of them is the origin. In parallel the same expression is evaluated on absolute values, and the result is multiplied by a constant derived from a forward error analysis of the expression in that format. The sign is accepted if the magnitude of the determinant exceeds this bound. The stage is written so that it can be applied to a whole batch of simplices at once, with the undecided ones compacted into a list for the next stage. Contraction into fused multiply-adds must be disabled for this code, or else the constant must be derived for the contracted form.



4.3. Stage B: filter in a wider format

Undecided calls are re-evaluated in binary64. For binary32 inputs the coordinate differences are formed in binary64, where they are exact whenever the operands are close enough in exponent for the result to fit in 53 bits, a condition the bounding box and grid in the input contract can guarantee. The first products are then nearly exact, and the error bound shrinks by many orders of magnitude compared with Stage A. On devices without binary64 the same stage is implemented with double-word arithmetic, in which each quantity is held as an unevaluated sum of two binary32 numbers and manipulated by error-free transformations (Muller et al., 2018), and the bound is adjusted accordingly.

4.4. Stage C: adaptive expansion arithmetic

The few calls that remain are close to degenerate. They are evaluated with expansions in the widest native format, following the adaptive scheme of Shewchuk (1997): the determinant is computed in successively more accurate approximations, each reusing the components already obtained, and evaluation stops as soon as the approximation exceeds its remaining error bound. Products use the fused multiply-add when the device provides a correctly rounded one and Dekker's splitting otherwise. The stage requires round-to-nearest and gradual underflow, and the library should verify both when it is initialized instead of assuming them.

4.5. Stage D and tie-breaking

If the adaptive stage runs to completion without certifying a nonzero sign, the full expansion has been computed and its sign is the exact answer. For grid inputs an alternative is multiword integer arithmetic of the length fixed by the input contract. A result of exactly zero means that the points are collinear, cocircular or the higher-dimensional equivalent. The library then applies a symbolic perturbation in the manner of the simulation of simplicity of Edelsbrunner and Mücke (1990), resolving the tie by comparing the global indices of the points, so that the triangulation algorithm never sees a degenerate case and the output is uniquely defined. Lévy (2016) describes a toolkit that generates filters, expansions and perturbation code from one formula, and the same generative approach is recommended here, since hand-written variants of one determinant are a known source of errors.

5. Verification Plan

A predicate library is useful only if its claim of exactness can be checked. The proposal is to test at three levels. Each error-free transformation is tested for exactness against multiword integer arithmetic on

operands chosen to stress carries, cancellation and the underflow boundary. Each predicate is then compared with an exact rational evaluation on inputs that are deliberately near-degenerate: points on a common line or circle perturbed by one unit in the last place, in the style of the failure-inducing examples of Kettner et al. (2008). Finally the device itself is probed at start-up for rounding mode, subnormal support, fused multiply-add behaviour and contraction, and the library refuses to run, or selects a slower safe path, when a required property is absent.

6. Limitations and What Remains to Be Shown

The design set out here is a proposal and has not been validated. Its correctness rests on error bounds that have to be derived, and preferably machine-checked, for each predicate in each format, and a mistake in a single constant would defeat the purpose. Its efficiency rests on the expectation that the narrow filter settles the large majority of calls. That is plausible for well-distributed inputs, since the narrow format leaves fewer bits of margin but most configurations are far from degenerate, yet for inputs with many nearly cocircular points, such as samples of spheres or regular grids, the proportion of calls reaching the later stages may be high. Only measurement on real devices can settle this.

Two further limits should be stated. The library guarantees correct predicates on the coordinates as stored, and says nothing about the error committed when those coordinates were rounded from other data. And it covers predicates only. Constructions such as circumcentres, which Voronoi diagrams and mesh refinement require, cannot be made exact in fixed-size floating-point formats and need separate treatment.

7. Conclusion

Exact predicates for Delaunay triangulation need not be confined to double-precision processors. A cascade that starts in the narrow native format, widens where widening makes products exact, and falls back on expansion arithmetic and symbolic perturbation can in principle deliver exact signs on hardware whose fast arithmetic is single precision. The components are all established. What this article adds is their arrangement around an explicit input contract and an explicit list of the hardware properties each stage depends on. Implementation, formal verification of the error bounds and measurement on current accelerators are the necessary next steps.

Conflict of interest

To be declared by the author(s) before publication.

Funding

To be declared by the author(s) before publication.

References

- Brönnimann, H., Burnikel, C. and Pion, S. (2001). [Interval arithmetic yields efficient dynamic filters for computational geometry](#). *Discrete Applied Mathematics*, 109(1-2): 25-47.
- Dekker, T.J. (1971). [A floating-point technique for extending the available precision](#). *Numerische Mathematik*, 18(3): 224-242.
- Delaunay, B. (1934). [Sur la sphère vide](#). *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7: 793-800.
- Devillers, O. and Pion, S. (2003). [Efficient exact geometric predicates for Delaunay triangulations](#). *Proceedings of the 5th Workshop on Algorithm Engineering and Experiments (ALENEX)*.
- Edelsbrunner, H. and Mücke, E.P. (1990). [Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms](#). *ACM Transactions on Graphics*, 9(1): 66-104.
<https://doi.org/10.1145/77635.77639>
- Fortune, S. and Van Wyk, C.J. (1996). [Static analysis yields efficient exact integer arithmetic for](#)

- computational geometry. *ACM Transactions on Graphics*, 15(3): 223-248.
- Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1): 5-48. <https://doi.org/10.1145/103162.103163>
- Guibas, L. and Stolfi, J. (1985). Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2): 74-123. <https://doi.org/10.1145/282918.282923>
- Higham, N.J. (2002). *Accuracy and Stability of Numerical Algorithms*. 2nd edition. Society for Industrial and Applied Mathematics, Philadelphia.
- Higham, N.J. and Mary, T. (2022). Mixed precision algorithms in numerical linear algebra. *Acta Numerica*, 31: 347-414. <https://doi.org/10.1017/S0962492922000022>
- IEEE (2019). *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019*. Institute of Electrical and Electronics Engineers, New York.
- Kettner, L., Mehlhorn, K., Pion, S., Schirra, S. and Yap, C. (2008). Classroom examples of robustness problems in geometric computations. *Computational Geometry*, 40(1): 61-78.
- Knuth, D.E. (1997). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 3rd edition. Addison-Wesley, Reading, MA.
- Lévy, B. (2016). Robustness and efficiency of geometric programs: The Predicate Construction Kit (PCK). *Computer-Aided Design*, 72: 3-12.
- Muller, J.-M., Brunie, N., de Dinechin, F., Jeannerod, C.-P., Joldes, M., Lefèvre, V., Melquiond, G., Revol, N. and Torres, S. (2018). *Handbook of Floating-Point Arithmetic*. 2nd edition. Birkhäuser, Cham.
- Priest, D.M. (1991). Algorithms for arbitrary precision floating point arithmetic. *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, 132-143.
- Shewchuk, J.R. (1997). Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3): 305-363. <https://doi.org/10.1007/PL00009321>
- Yap, C.-K. (1997). Towards exact geometric computation. *Computational Geometry*, 7(1-2): 3-23.