Supplement to: Accurate and Robust Geometric Algorithms for Regridding on the Sphere

Hongyu Chen[1], Paul A. Ullrich[2,3], Julian Panetta[4], David Marsico[5,6], Moritz Hanke[7], Rajeev Jain[8], Chengzhu Zhang[3], and Robert L. Jacob[8]

[1]Computer Science Graduate Group, University of California, Davis, Davis, CA, USA
[2]Department of Land, Air and Water Resources, University of California, Davis, Davis, CA, USA
[3]Physical and Life Sciences Directorate, Lawrence Livermore National Laboratory, Livermore, CA, USA
[4]Department of Computer Science, University of California, Davis, Davis, CA, USA
[5]NOAA Physical Sciences Laboratory, Boulder, CO, USA
[6]Cooperative Institute for Research in Environmental Sciences, University of Colorado Boulder, Boulder, CO, USA
[7]Application Support, Deutsches Klimarechenzentrum, Hamburg, Germany
[8]Argonne National Laboratory, Lemont, IL, USA

S1 Spherical mass calculation

Using *Wolfram Mathematica* (Wolfram Research), we obtain a closed-form expression for the inner moment $\mathbf{M}_{\text{inner}}(\lambda)$ in Eq. (19):

$$\mathbf{M}_{\text{inner}}(\lambda) = \int_{\phi_0}^{\tilde{\phi}(\lambda)} \mathbf{x}(\phi,\lambda)\cos\phi\, d\phi = \begin{pmatrix} M_{\text{inner}_x}(\lambda) \\ M_{\text{inner}_y}(\lambda) \\ M_{\text{inner}_z}(\lambda) \end{pmatrix}. \tag{S1}$$

$$\begin{aligned} M_{\text{inner}_x}(\lambda) = \frac{1}{2}\cos\lambda\Bigg(& \frac{\cos\left(\frac{\delta\lambda}{2}\right)\sin\phi_0\cos\phi_0\sec\left(\frac{\delta\lambda}{2}-\lambda\right)}{\left|\cos^2\phi_0\sec^2\left(\frac{\delta\lambda}{2}-\lambda\right)\sin(\delta\lambda-\lambda)\sin\lambda - 1\right|} \\ & + \csc^{-1}\Big(\csc\phi_0\sqrt{1-\sin\lambda\cos^2\phi_0\sin(\delta\lambda-\lambda)\sec^2\left(\tfrac{\delta\lambda}{2}-\lambda\right)}\Big) - \phi_0 - \sin\phi_0\cos\phi_0\Bigg). \end{aligned} \tag{S2}$$

$$\begin{aligned} M_{\text{inner}_y}(\lambda) = \frac{1}{2}\sin\lambda\Bigg(& \frac{\cos\left(\frac{\delta\lambda}{2}\right)\sin\phi_0\cos\phi_0\sec\left(\frac{\delta\lambda}{2}-\lambda\right)}{\left|\cos^2\phi_0\sec^2\left(\frac{\delta\lambda}{2}-\lambda\right)\sin(\delta\lambda-\lambda)\sin\lambda - 1\right|} \\ & + \csc^{-1}\Big(\csc\phi_0\sqrt{1-\sin\lambda\cos^2\phi_0\sin(\delta\lambda-\lambda)\sec^2\left(\tfrac{\delta\lambda}{2}-\lambda\right)}\Big) - \phi_0 - \sin\phi_0\cos\phi_0\Bigg). \end{aligned} \tag{S3}$$

$$M_{\text{inner}_z}(\lambda) = \frac{1}{2}\left(\cos^2\phi_0 - 1 + \frac{1}{\csc^2\phi_0 - \sin\lambda\cot^2\phi_0\sin(\delta\lambda-\lambda)\sec^2\left(\frac{\delta\lambda}{2}-\lambda\right)}\right). \tag{S4}$$

S2 Spherical area calculation

S2.1 Spherical triangle area calculation and splitting

A mathematically accurate way to calculate the spherical triangle area is to use the Gaussian Quadrature as firsted introduced in Dunavant (1985). It begins with the standard vertex fan but augments it with adaptive subdivision and high-order quadrature to retain accuracy on faces that are either large or highly elongated. To begin, define Ω as the spherical triangle with vertices $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ connected by great circle arcs. A barycentric coordinate system $(\zeta, \eta) \in [0,1]^2$ can be defined over the planar triangle T enclosed by these vertices via

$$\mathbf{r}(\zeta,\eta) = (1-\zeta-\eta)\mathbf{x}_1 + \zeta\mathbf{x}_2 + \eta\mathbf{x}_3, \tag{S5}$$

This point on T is subsequently projected to the sphere via

$$\mathbf{p}(\mathbf{r}) = \frac{\mathbf{r}}{||\mathbf{r}||}. \tag{S6}$$

Therefore, the area of Ω can be rewritten as

$$\int_{\Omega} dA = \int_0^1 \int_0^{\eta} \left|\left| \frac{\partial \mathbf{p}}{\partial \zeta} \times \frac{\partial \mathbf{p}}{\partial \eta} \right|\right| d\zeta d\eta. \tag{S7}$$

Using properties of the cross product, the following equation can be obtained

$$\left|\left| \frac{\partial \mathbf{p}}{\partial \zeta} \times \frac{\partial \mathbf{p}}{\partial \eta} \right|\right| = \frac{\mathbf{x}_1}{||\mathbf{r}^3||} \cdot (\mathbf{x}_2 \times \mathbf{x}_3). \tag{S8}$$

We can then use triangular quadrature to integrate over T_1, so that the area of Ω can be written as

$$\int_{\Omega} dA \approx \sum_{i=1}^{N_q} \frac{\mathbf{x}_1}{||\mathbf{r}^3||} \cdot (\mathbf{x}_2 \times \mathbf{x}_3)\, w_i, \tag{S9}$$

where N_q is the number of points of the quadrature rule, and w_i are the quadrature weights.

In the general case of a face with N vertices, we decompose the face into $N-2$ triangular sub-face. We then apply equation (S9) to each triangular sub-face, and add up the results, yielding the equation

$$\int_{\Omega} dA \approx \sum_{i=1}^{N-2} \sum_{j=1}^{N_q} \frac{\mathbf{x}_{1,i}}{||\mathbf{r}_i^3||} \cdot (\mathbf{x}_{2,i} \times \mathbf{x}_{3,i})\, w_{i,j}, \tag{S10}$$

where $\mathbf{x}_{1,i}$, $\mathbf{x}_{2,i}$, and $\mathbf{x}_{3,i}$ are the vertices of sub-triangle i,

$$\mathbf{r}_i = (1-\zeta-\eta)\mathbf{x}_{1,i} + \zeta\mathbf{x}_{2,i} + \eta\mathbf{x}_{3,i}, \tag{S11}$$

and $w_{i,j}$ is the quadrature weight at quadrature point j of sub-triangle i.

While the method above generally provides near machine truncation accuracy for computing face areas, for larger faces or for faces with a high aspect ratio, it may be necessary to use an adaptive algorithm for triangulation. Here we present the following way to split the given face. Let h be the maximum edge length of a face, where the edge length is defined as the Euclidean distance between two adjacent vertices. If $h < 0.004$, we use equation (S10) with $N_q = 4$ Gaussian quadrature, and if $h < 0.09$, equation (S10) is used with $N_q = 8$ quadrature. These thresholds, 0.004 and 0.09, are empirical choices, and we observe that they give good results in practice. Otherwise, we repeatedly subdivide the face into triangular sub-faces as illustrated in Fig. 5, which is detailed into the following steps:

1. Decompose the face into triangular sub-faces as in figure 5b.
2. For each triangular sub-face, find the midpoints of each of the three chords connecting adjacent vertices; call these points $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, and $\boldsymbol{v}_3$. Project these midpoints onto the sphere by using equation (S6) to compute $\boldsymbol{p}(\boldsymbol{v}_1)$, $\boldsymbol{p}(\boldsymbol{v}_2)$, and $\boldsymbol{p}(\boldsymbol{v}_3)$.
3. Four triangular sub-faces are then formed by connecting $\boldsymbol{p}(\boldsymbol{v}_1)$, $\boldsymbol{p}(\boldsymbol{v}_2)$, and $\boldsymbol{p}(\boldsymbol{v}_3)$, as in figure 5c
4. Repeat the second and third steps until the maximum edge length of each sub-face is less than a given tolerance, and then compute its area using order 8 quadrature.

S2.2 Area correction for constant latitude

Without loss of generality we rotate the coordinate system about the z–axis so that the two nodes lie on the parallel $\phi = \phi_0$ with longitudes $\lambda_1 = 0$ and $\lambda_2 = \Delta\lambda$. The nodes are

$$\mathbf{x}_1 = \big(\cos\phi_0, 0, \sin\phi_0\big), \qquad \mathbf{x}_2 = \big(\cos\phi_0\cos\Delta\lambda, \cos\phi_0\sin\Delta\lambda, \sin\phi_0\big).$$

We study the spherical region bounded by the line of constant latitude $\phi = \phi_0$ and the great-circle arc joining $\mathbf{x}_1$ and $\mathbf{x}_2$. The arc is parameterized along the chord as in (3): first perform a linear interpolation on the chord and then normalize the result to unit length. This coordinate choice serves only as a convenience. The subsequent formulas depend solely on the longitude difference $\Delta\lambda$ rather than on the absolute longitudes of the two vertices. Setting $y = 0$ at $\mathbf{x}_1$ therefore imposes no restriction.

$$\|\tilde{\mathbf{x}}\|_2 = \sqrt{1 + 2a(1-a)(\cos\Delta\lambda - 1)\cos^2\phi_0}. \tag{S12}$$

Thus the longitude and latitude at each point along the great circle arc are given by

$$\lambda = \arctan\left[\frac{a\sin\Delta\lambda}{(1-a) + a\cos\Delta\lambda}\right], \tag{S13}$$

$$\phi = \arcsin\left[\frac{\sin\phi_0}{\sqrt{1 + 2a(1-a)(\cos\Delta\lambda - 1)\cos^2\phi_0}}\right]. \tag{S14}$$

The area between these two boundaries is given by

$$A_{\text{corr}} = \int_0^{\Delta\lambda}\int_{\phi_0}^{\phi} \cos\phi\, d\phi\, d\lambda = \int_0^{\Delta\lambda} [\sin\phi - \sin\phi_0]\, d\lambda. \tag{S15}$$

Converting this to an integral in a using (S13) gives

$$A_{\mathrm{corr}} = -\Delta\lambda \sin\phi_0 + \sin\phi_0 \sin\Delta\lambda \int_0^1 \frac{1}{[1+2a(1-a)(\cos\Delta\lambda - 1)]\sqrt{1+2a(1-a)(\cos\Delta\lambda-1)\cos^2\phi_0}} da. \tag{S16}$$

Carrying out the remaining algebraic simplifications of this integral yields the closed-form expression used in the main text, namely Eq. (24):

$$A_{\mathrm{corr}} = 2\,\mathrm{atan2}\left(\sin\phi_0 \tan\left(\frac{\Delta\lambda}{2}\right), 1\right) - \sin\phi_0\,\Delta\lambda. \tag{S17}$$

S3 Experiments

This section presents two accuracy experiments evaluating the numerical performance of the formulas and algorithms described earlier: the great-circle arc angle computation (Section 2) and the spherical-triangle area computation (Section 5.10). All reference (baseline) values are computed with 17-digit arbitrary-precision using *Mathematica* (Wolfram Research). Inputs are in IEEE 754 double-precision (binary64), following the IEEE Floating-Point Standard (IEEE, 2008). These experiments isolate algorithmic error by keeping hardware-level floating-point format fixed, enabling a clear assessment of each computational method's numerical stability.

S3.1 Great circle arc angle calculation

The test dataset contains a $100{,}000 \times 3$ matrix, where each row stores the coordinates of two endpoints of a geodesic arc, $\mathbf{x}_1 = [x_1, y_1, z_1], \quad \mathbf{x}_2 = [x_2, y_2, z_2]$. The arcs span angles from 0° to 90°. This range is sufficient because $\arccos$ becomes ill-conditioned as the angle approaches 0°, and $\arcsin$ reaches its largest errors near 90°. Results for 90°–180° are symmetric to those for 0°–90°, consistent with many numerical linear algebra applications. For each pair, the arc angle is computed using four methods: $\arcsin$, $\arccos$, direct $\mathrm{atan2}$, and Kahan's formulation using $\mathrm{atan2}$. Baseline values are generated using Mathematica's arbitrary-precision framework, fixed at 17 decimal digits, ensuring agreement to full IEEE 754 double-precision accuracy. For visualization, the 0°–90° interval is partitioned into uniform 5° bins (with narrower 0.1° bins near the edges). Within each bin, we compute the maximum relative error over all sample angles falling in that interval. Each bin is then represented by a single point at the bin midpoint along the horizontal axis (e.g., the bin $[5^\circ, 10^\circ)$ is plotted at 7.5°). This choice makes the horizontal position reflect the central angle of the bin while the vertical coordinate reflects the worst-case error within that bin.

As shown in Fig. S1, Kahan's formula remains stable across the entire range. $\arcsin$ is ill-conditioned near 0°, $\arccos$ is ill-conditioned near 90°.

We benchmarked the angle-calculation kernels on the NERSC Perlmutter CPU partition (HPE Cray EX), using a single CPU node with two AMD EPYC 7763 ("Milan") processors (64 cores/socket, AVX2, NPS = 4) and 512 GB DDR4 RAM, connected via HPE Slingshot 11 (node-level memory bandwidth ∼204.8 GB/s per socket (National Energy Research Scientific Computing Center, 2025)). Kernels were compiled in release mode with `-fno-fast-math -ffp-contract=off` to

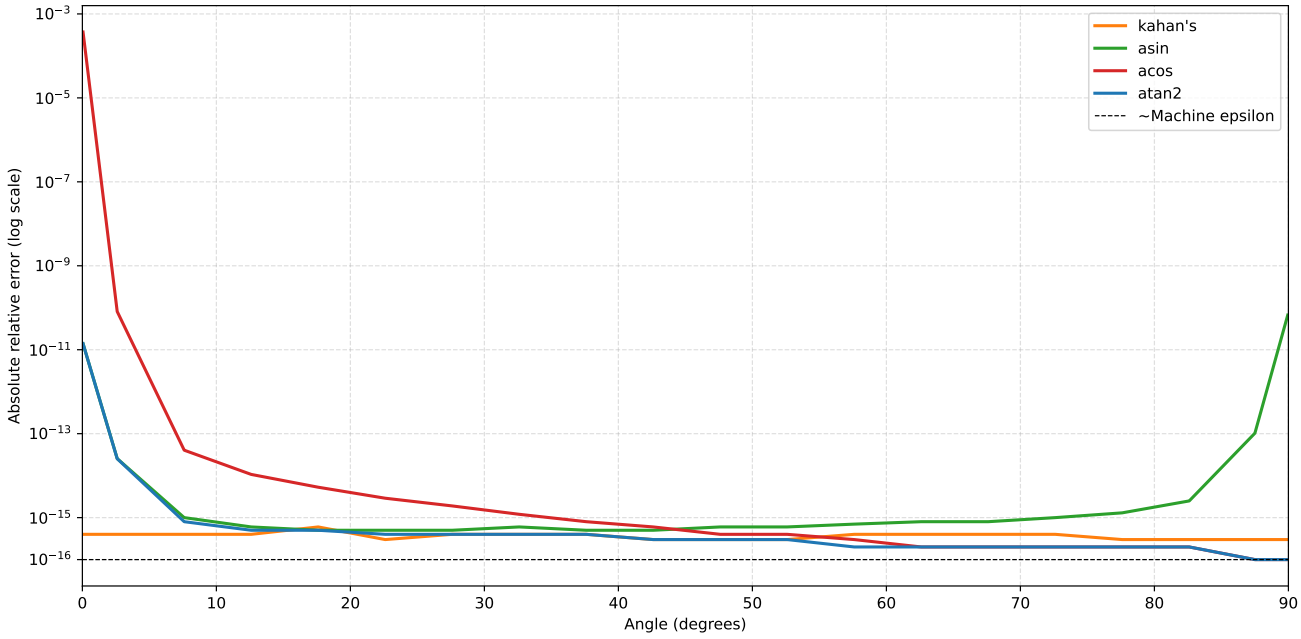

Figure S1. Relative error in computed great–circle arc length as a function of the baseline angle (log scale).

preserve IEEE-754 semantics, and results are reported as average execution time per angle (seconds). From the plot, we observe that the most accurate method—Kahan's angle formulation—is also the second fastest among all tested approaches. The seemingly counterintuitive result that asin outperforms acos, despite involving additional operations such as cross products and vector normalization (Eq. (9)), is likely attributable to compiler-level optimization effects.

S3.2 GCA–GCA intersection

We evaluate Algorithm 2, an EFT-enhanced floating-point kernel, against two commonly used float64 implementations: a direct implementation using standard operations and the same code with the cross product computed via Kahan's 2×2 determinant with fused multiply add (FMA). The objective is to measure numerical accuracy as configurations become ill-conditioned for nearly coincident great-circle arcs.

Test pairs are generated by first forming an arc $a = [\mathbf{a}_0, \mathbf{a}_1]$ on the unit sphere with length less than π. A point $\mathbf{p}$ is sampled uniformly along a, and a target separation angle θ is drawn from a logarithmically spaced set covering 10^{-8} to 10 degrees. The second arc $b = [\mathbf{b}_0, \mathbf{b}_1]$ is created by rotating $\mathbf{a}_0$ and $\mathbf{a}_1$ about the tangent at $\mathbf{p}$ by θ, with renormalization applied to the endpoints. Baseline intersections are computed in Mathematica with arbitrary precision chosen to guarantee at least 17 correct digits, which serves as ground truth. We report the absolute relative errors with respect to these references.

Fig. S3 shows that Algorithm 2 stays around or below machine epsilon for most angles, with only a slight increase in the most ill-conditioned regime $\theta \lesssim 10^{-6}\,\mathrm{deg}$. This follows from its use of EFT, which attains accuracy as if computed at roughly

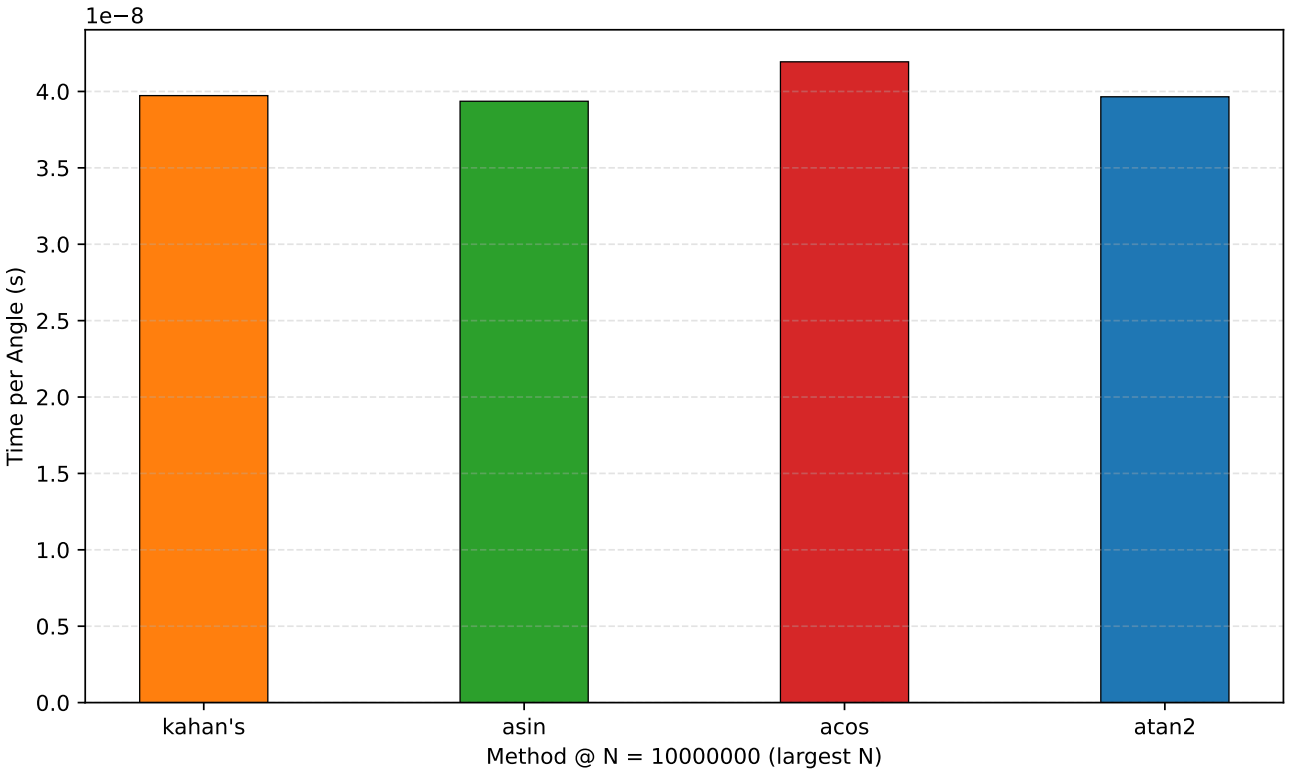

Figure S2. Performance comparison of different angle implementations across different numbers of angles.

twice the working precision and then rounded to double. Therefore, even in the worst cases where the float64 variants lose more than half of the available digits, Algorithm 2 still maintains effectively doubled-precision accuracy. By contrast, the two float64 variants deteriorate as θ decreases, with errors growing roughly linearly on a log–log scale and exceeding machine epsilon by $\theta \approx 10\,\mathrm{deg}$. Kahan's determinant with FMA provides a small, nearly uniform reduction but does not change this trend. Because the intersection kernel is compute-light, once it is properly vectorized and parallelized, the runtime becomes I/O-bound (Chen et al., 2025), so the EFT high-accuracy Algorithm 2 incurs effectively no additional compute time.

S3.3 Spherical triangle area calculation

We benchmark the improved Gaussian–quadrature formulation in *ARPIST* against (i) direct quadrature in *TempestRemap*, (ii) the L'Huilier-based implementation in *YAC*, and (iii) the Eq. (23) variant derived from Eriksson (1990). To evaluate numerical accuracy, we generated a dataset of ill-conditioned spherical triangles: $n = 60$ colatitudes were sampled in a geometric progression with ratio 0.8, and $m = 360$ azimuths were sampled uniformly over $[0, \pi]$, recreating the setup from Li and Jiao (2022). Each triangle was anchored at a random point on the unit sphere within a local orthonormal frame and perturbed by a small random azimuthal offset. And among these triangles, we picked only the triangle with the maximum edge length

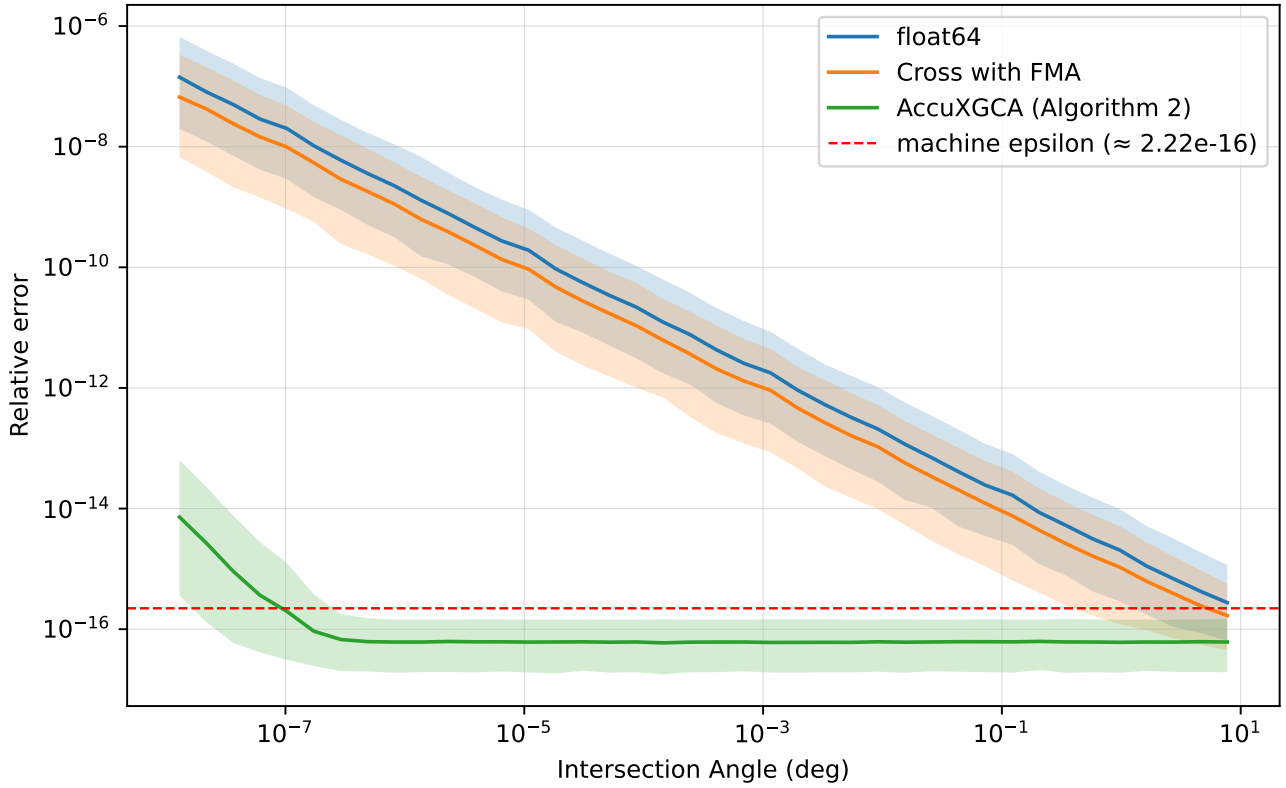

Figure S3. Accuracy of great-circle intersection versus separation angle in degrees. Solid lines show the mean absolute relative error; the shaded region shows the central spread from the 5th to the 95th percentiles of the samples. The dashed red line marks double precision machine epsilon ($\approx 2.22 \times 10^{-16}$).

$h \approx 0.26$, following Li and Jiao (2022). The reference baseline was computed with arbitrary-precision arithmetic using *Wolfram Mathematica*.We evaluate four configurations: `ARPIST` at fixed degree 8 (no splitting) and with adaptive quadrature (splitting enabled); `TempestRemap` likewise at fixed degree 8 (no splitting) and with adaptive quadrature (splitting enabled); the L'Huilier implementation in *YAC* (FMA enabled); and the Eq. (23) variant (FMA enabled to match L'Huilier).

Figure S4 reports absolute relative error versus the minimum interior angle for the pointy, ill-conditioned triangles described above. The L'Huilier implementation in *YAC* exhibits the largest errors ($\sim 10^{-7}$–10^{-9}) due to cancellation in the small-angle regime. Fixed degree-8 Gaussian quadrature in both `TempestRemap` and `ARPIST` clusters near 10^{-12}–10^{-11}, with accuracy limited when the h too large and requires further splittig. Enabling adaptive quadrature with splitting reduces error by one to three orders of magnitude; `ARPIST` is consistently tighter than `TempestRemap`, attributable to its anchored-step design. The Eq. (23) variant implemented with FMA attains the same accuracy as `ARPIST` adaptive quadrature with splitting, confirming that well-conditioned formulas paired with EFT/FMA arithmetic can achieve high accuracy while being efficient.

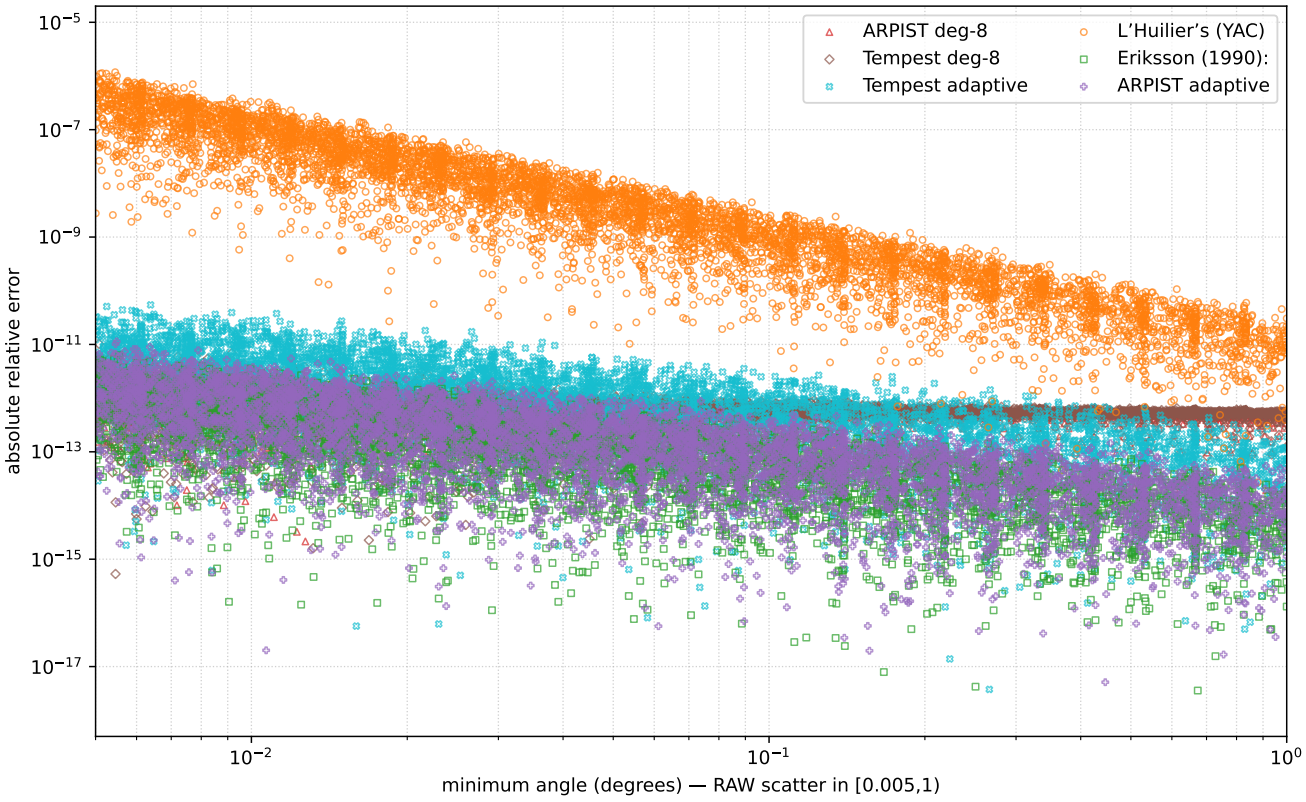

Figure S4. Absolute relative error vs. the triangle's minimum angle for the ill-conditioned set. The minimum angle ranges from 0° to 1°. All faces selected have $h \approx 0.26$ with tolerance 0.005.

In addition, Fig. S5 illustrates a more practical scenario: the area computation errors on icosahedral meshes of different resolutions covering the entire sphere. The input mesh triangles have minimum angles uniformly distributed between 48° and 60°, regardless of resolution, while the edge length decreases as resolution increases. With similar minimum angles, the L'Huilier formulation degrades as edges shorten, whereas the adaptive Gaussian quadrature in *ARPIST* remains stable across resolutions. For fixed degree-8, no-splitting methods (`TempestRemap` and `ARPIST`), degree 8 is insufficient for the large triangles at low resolution (large h), so accuracy drops. As resolution increases and h becomes small, the accuracy bottleneck is no longer h, and the fixed degree-8 trends overlap their corresponding adaptive trends, indicating both h and degree are not the bottleneck, and the remaining error is dominated by evaluation/roundoff effects rather than quadrature order, making degree 4 and degree 8 numerically indistinguishable at the plotted scale. `ARPIST` remains slightly more accurate than `TempestRemap`, consistent with its anchored-step design. The Eq. (23) variant with FMA attains the same level of accuracy as the Gaussian-quadrature implementations.

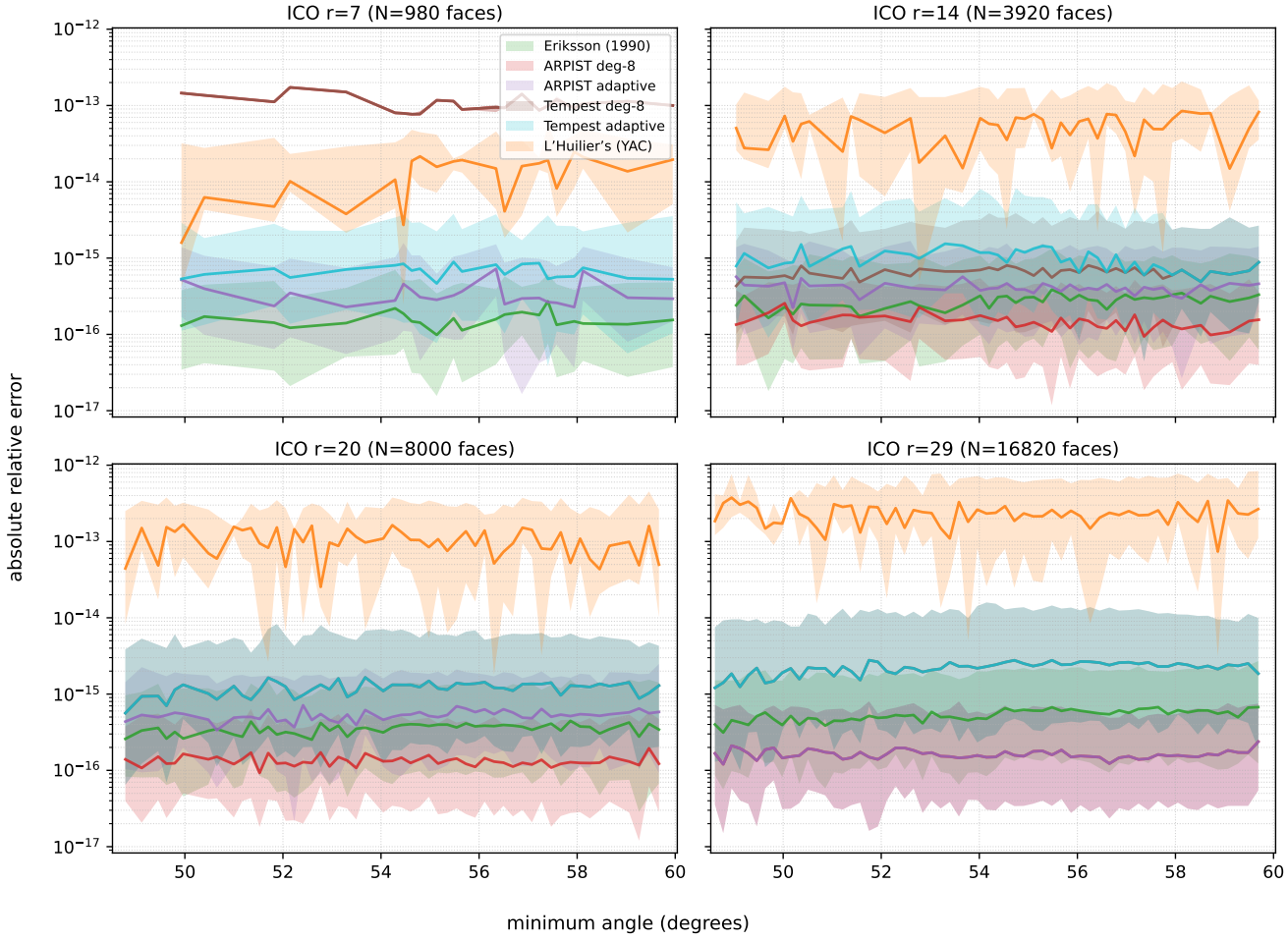

Figure S5. Four-panel polygon plots for icosahedral meshes at resolutions $r \in \{7, 14, 20, 29\}$. Shaded bands show the 5%–95% envelope within each minimum-angle bin; the center line is the bin-wise mean (computed in $\log_{10}$-error space). Angles are grouped into 60 log-spaced bins over the observed minimum-angle range. (60 balances resolution with per-bin sample size across all meshes)

S3.4 KD-Tree and Ball-Tree

We benchmarked nearest-neighbor construction and query performance using the `sklearn.neighbors.KDTree` and `sklearn.neighbors.BallTree` libraries (Pedregosa et al., 2011). Three configurations were tested: (1) KDTree with Euclidean chord distance, (2) BallTree with great-circle (haversine) distance, and (3) BallTree with Euclidean chord distance. The experiments were conducted on two types of grids: uniform meshes and regionally refined meshes. For each grid, we used the Cartesian coordinates of face centers as query points, and measured the total runtime of `tree.query(Q, 1, return_distance=False)` across all faces. Benchmarks were run on the NERSC Perlmutter CPU partition (HPE Cray EX), using a single CPU node equipped with two AMD EPYC 7763 ("Milan") processors (64 cores per socket, AVX2,

NPS = 4), 512 GB DDR4 RAM, and HPE Slingshot 11 interconnect (node-level memory bandwidth ∼204.8 GB/s per socket (National Energy Research Scientific Computing Center, 2025)). We exclude *GriSPy* because its recommended setting ($N_{\text{cell}} = 2^6$; (Chalela et al., 2021)) is not memory efficient for our problem sizes.

Figures S6 and S7 show that `sklearn.neighbors.KDTree` attains faster query times but incurs higher construction costs, whereas `sklearn.neighbors.BallTree` builds more quickly but queries more slowly. Leaf size also matters: smaller leaves yield faster queries. Although reducing the leaf size slightly increases build time, the discrepancy becomes negligible at large node counts.

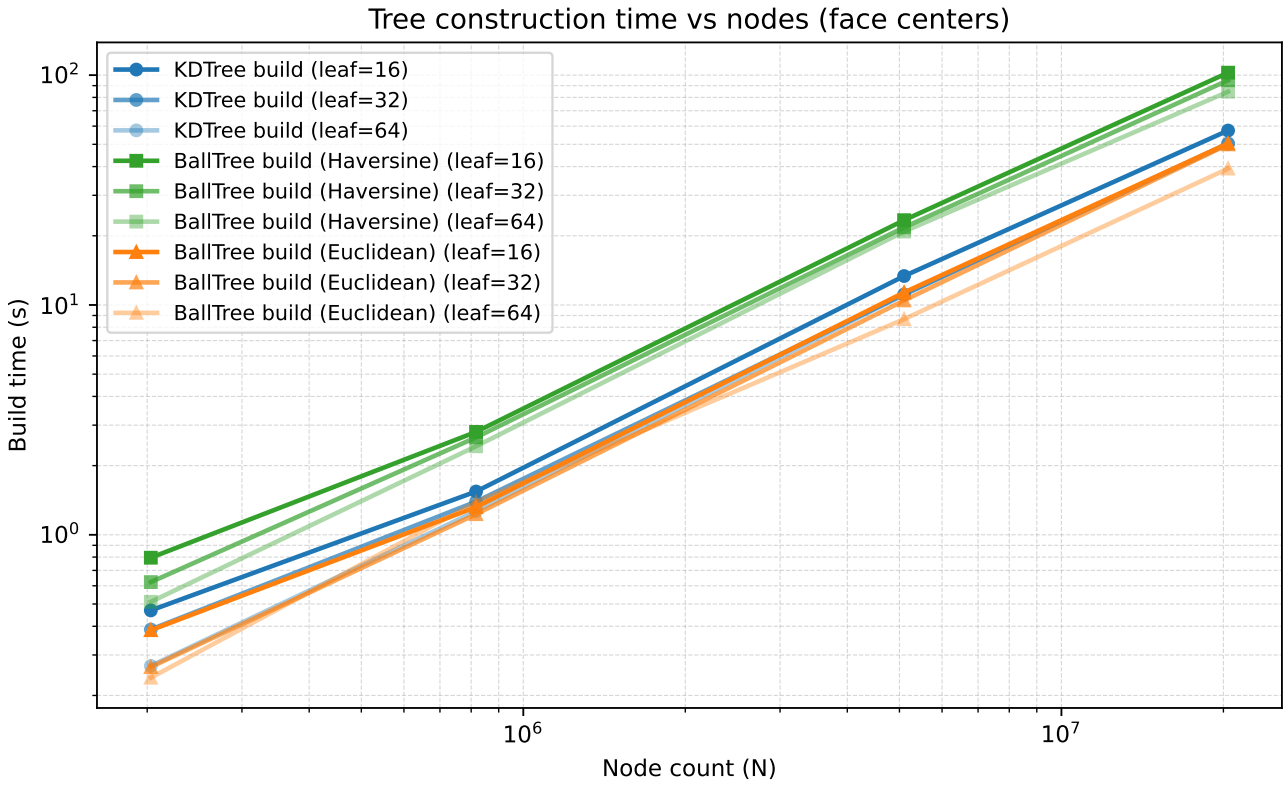

Figure S6. Construction performance of KDTree and BallTree for varying leaf sizes, measured as runtime versus number of face centers.

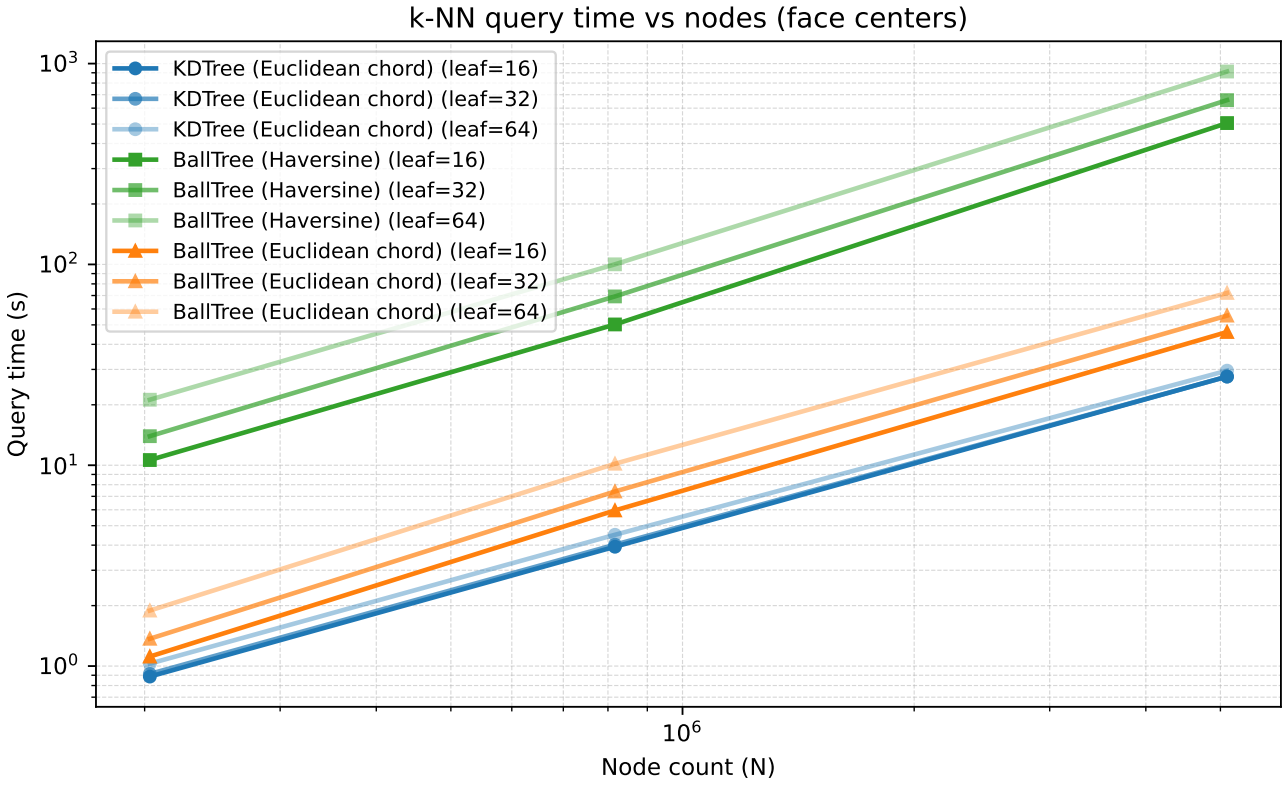

Figure S7. Query performance of KDTree and BallTree for varying leaf sizes, measured as runtime versus number of face centers.

References

Chalela, M., Sillero, E., Pereyra, L., Garcia, M. A., Cabral, J. B., Lares, M., and Merchán, M.: GriSPy: A Python package for fixed-radius nearest neighbors search, Astronomy and Computing, 34, 100443, https://doi.org/10.1016/j.ascom.2020.100443, 2021.

Chen, H., Ullrich, P. A., and Panetta, J.: Fast and Accurate Intersections on a Sphere, https://arxiv.org/abs/2510.09892, 2025.

Dunavant, D. A.: High degree efficient symmetrical Gaussian quadrature rules for the triangle, International Journal for Numerical Methods in Engineering, 21, 1129–1148, https://api.semanticscholar.org/CorpusID:120117894, 1985.

Eriksson, F.: On the Measure of Solid Angles, Mathematics Magazine, 63, 184–187, http://www.jstor.org/stable/2691141, 1990.

IEEE: IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008, pp. 1–70, https://doi.org/10.1109/IEEESTD.2008.4610935, 2008.

Li, Y. and Jiao, X.: ARPIST: Provably Accurate and Stable Numerical Integration over Spherical Triangles, arXiv preprint arXiv:2201.00261, 2022.

National Energy Research Scientific Computing Center: Perlmutter Architecture, https://docs.nersc.gov/systems/perlmutter/architecture/, accessed: 2025-08-09, 2025.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E.: Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825–2830, 2011.

Wolfram Research, I.: Mathematica, Version 14.3, https://www.wolfram.com/mathematica, champaign, IL, 2025.