Subgrid Marching Tetrahedra

HOSSEIN BAKTASH, Carnegie Mellon University, USA
MARK GILLESPIE, Inria, France and University of Utah, USA
KEENAN CRANE, Carnegie Mellon University, USA and Roblox, USA

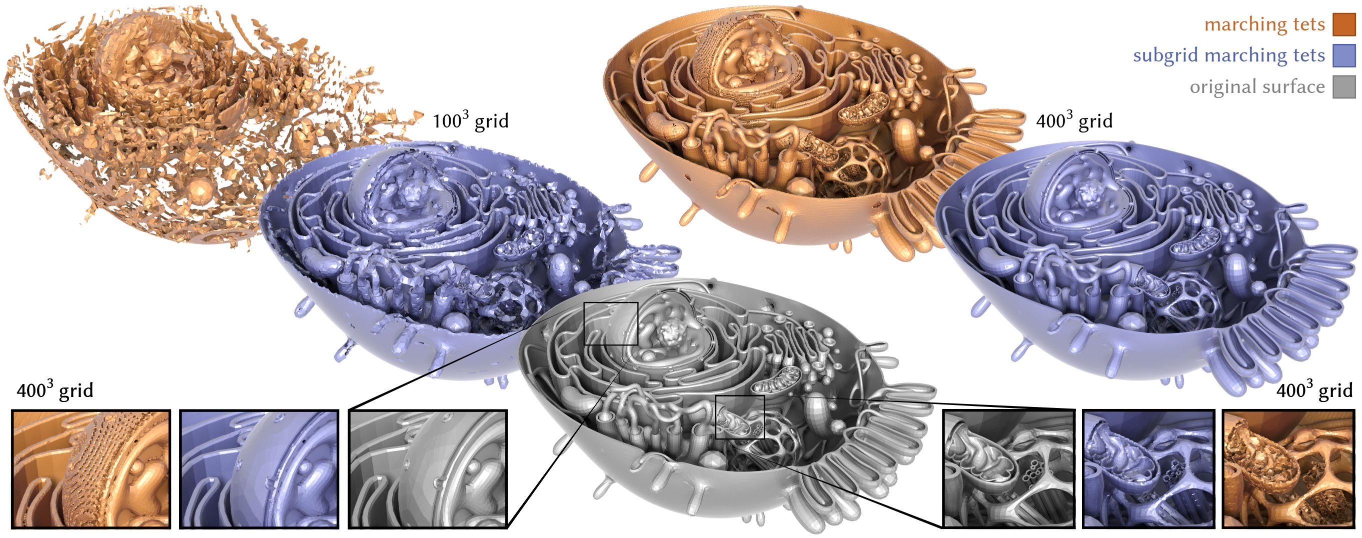


Fig. 1. Just as dual contouring extends classic marching algorithms to capture sharp geometric features, our *subgrid* extension enables marching algorithms to capture fine topological features below the grid resolution. Here we reconstruct a cell model (gray) using standard marching tetrahedra (orange) versus our subgrid marching tetrahedra (blue), using dual variants of both algorithms. Even on an extremely coarse grid *(top left)*, our subgrid method does a much better job of reconstructing thin sheets and fine features. At higher resolution *(top right)*, we better resolve small details—without adaptive sampling or refinement.

We describe a method for recovering a manifold, intersection-free triangle mesh from the points where edges of a tetrahedral grid pierce a continuous surface. Unlike classic marching cubes or tets, our *subgrid marching* scheme allows arbitrarily many surface patches within a single cell, capturing fine features and thin sheets. Moreover, it requires neither a well-defined inside/outside (allowing surfaces with boundary), nor consistently-oriented input geometry. Yet we retain the local, parallel nature of classic marching: reconstruction is performed independently per tet, yielding a conforming mesh across tet boundaries. Our key innovation is a generalization of *normal coordinates* from geometric topology, which encode surface connectivity via arbitrary integer intersection counts along each grid edge. This encoding sidesteps the usual Nyquist–Shannon limit, putting no lower bound on the size of features that can be resolved on a fixed grid. In practice, for similar compute time and equal grid resolution—or even an equal number of output triangles—meshes produced by subgrid marching are far more accurate than those from classic marching. Beyond standard contouring, our method can be used to convert polygon soup into a manifold, intersection-free mesh.

Authors' Contact Information: Hossein Baktash, hbaktash@andrew.cmu.edu, Carnegie Mellon University, Pittsburgh, PA, USA; Mark Gillespie, mark.gillespie81@gmail.com, Inria, Palaiseau, France and University of Utah, Salt Lake City, USA; Keenan Crane, keenanc@andrew.cmu.edu, Carnegie Mellon University, Pittsburgh, PA, USA and Roblox, San Mateo, CA, USA.

ACM 1557-7368/2026/7-ART57
https://doi.org/10.1145/3811358

CCS Concepts: • **Computing methodologies → Mesh geometry models**; *Shape representations*; Reconstruction.

Additional Key Words and Phrases: Marching Tetrahedra, Topology, Iso Surfaces, Surface Reconstruction

ACM Reference Format:
Hossein Baktash, Mark Gillespie, and Keenan Crane. 2026. Subgrid Marching Tetrahedra. *ACM Trans. Graph.* 45, 4, Article 57 (July 2026), 20 pages. https://doi.org/10.1145/3811358

1 Introduction and Related Work

Isosurface contouring is the three-dimensional analog of root finding: given a continuous function $f : \mathbb{R}^3 \to \mathbb{R}$, one seeks a polygonal approximation of the zero set

$$\mathcal{S} := \{x \in \mathbb{R}^3 \mid f(x) = 0\}. \quad (1)$$

This operation is fundamental to geometry processing, enabling conversion from implicit to explicit surface representations—*e.g.*, from a *signed distance function (SDF)* to a polygonal mesh.

We generalize this setup considerably: we do not require an SDF or any other implicit function, and devise a general strategy for contouring a surface from any collection of grid-surface intersection points. Our approach builds on classic *marching algorithms*, such as marching cubes [Lorensen and Cline 1987] and marching tetrahedra [Doi and Koide 1991], which take a "divide and conquer"

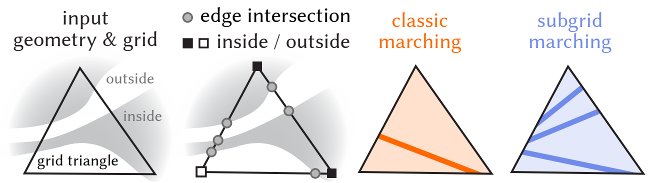


Fig. 2. Our subgrid method reconstructs geometry from arbitrarily many intersection points along each edge. As a result, it can capture features that are missed entirely by traditional marching algorithms.

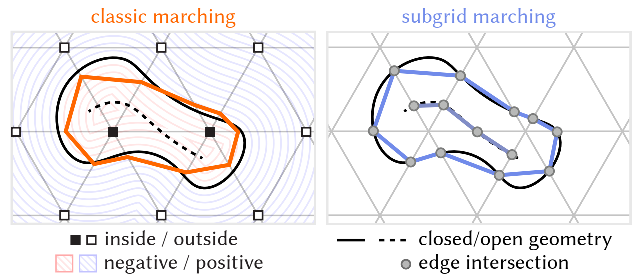


Fig. 3. *Left:* Classic marching relies on an implicit function to classify grid nodes as inside/outside, making it impossible to capture open curves (2D) or surfaces with boundary (3D). Moreover, using interpolated nodal values to estimate intersection locations leads to errors in the geometric approximation. *Right:* Subgrid marching avoids both of these issues by reconstructing geometry directly from edge-surface intersection points.

approach: divide space into cells, and build a polygonal approximation within each cell. Marching is popular in practice since it is amenable to parallelization, and guarantees closed manifold output.

However, classic marching methods exhibit two basic limitations. First, they require a *continuous* function f, relying on the transition between negative and positive values to identify intersections. Hence, they are suitable only for closed curves or surfaces (Figure 3). Second, output frequency is limited by grid resolution, since the sign test used by these methods produces at most one vertex per grid edge (Figure 2). Hence, when f has multiple zeros along an edge, one gets aliasing of fine geometric features, thin sheets, *etc.* One common remedy is to simply increase grid resolution—yielding over-tessellated output which must be aggressively decimated. Alternatively, spatially adaptive methods lose the simplicity and efficiency of regular marching, and may still fail to identify surface-edge intersections [Ju et al. 2002; Schaefer et al. 2007].

Our approach addresses these limitations head-on, making two changes to the standard marching strategy that enable us to resolve features below the scale of the grid (hence the name "subgrid"):

(1) We replace 0-dimensional sampling (evaluate f at each grid node), with 1-dimensional root finding (find all zeros of f along each grid edge). As detailed in Section 4.3.2, this task can be carried out efficiently and exactly for many common implicit surface types (*e.g.*, SDFs or neural implicits), or approximately for "black box" functions f (via regular sampling along edges)—providing a drop-in replacement for existing marching algorithms (Section 1.3).
(2) Rather than rely on a finite lookup table of output configurations, we develop a deterministic algorithm that reconstructs a local polygonal approximation given *any* number of intersections along the edges of a tetrahedron (Figure 4). This algorithm is $O(n)$ in the number of edge intersections n, and produces a globally conforming mesh even though it is executed independently on each tet. The existence of such an algorithm is not obvious *a priori*, and occupies the bulk of our exposition.

Reconstruction proceeds inductively on dimension (Section 3): we start with 0-dimensional intersection points along edges (Section 4.3.2), construct a collection of 1-dimensional curves on the boundary of each tetrahedron (Section 3.1), and then fill these tetrahedra with 2-dimensional polygons interpolating the boundary curve (Section 3.2 and Section 3.3). Since we need only intersection points along edges (and not an implicit function), the target surface need not have a well-defined inside and outside, nor a consistent orientation. Likewise, a dual variant of our algorithm (Section 3.3) uses only *unoriented normals* (*i.e.*, sign does not matter). This flexibility enables our method to be applied to broader tasks than contouring. For instance, by intersecting each edge with an inconsistently-oriented polygon soup, we can recover an oriented manifold mesh (Section 4).

1.1 Normal Surfaces

The starting point for our approach is *normal surface theory*, which is a central tool in the algorithmic study of 3-manifolds [Matveev 2007]. This topic was initiated by Kneser [1929] and Haken [1961], and later extended by others [Rubinstein 1994; Bachman 2003, 2012].

The basic idea is to encode a surface $\mathcal{S}$ by counting intersections with a tetrahedral grid $\mathcal{T}$ (Section 2). To make this encoding canonical, the surface must be *normal*, intersecting each tetrahedron in a collection of topological disks that either separate one vertex from the other three ("corner cuts"), or split vertices into two sets of two ("diagonal cuts"), as seen in Figure 4, *left.* A surface is hence reduced to a vector $\mathbf{x} \in \mathbb{Z}^{7n}_{\geq 0}$ of integer *normal coordinates*, since there are seven such patterns for each of n tets in the grid. (Classic marching tetrahedra is even more restrictive, producing a vector $\mathbf{x}$ where only one entry is 1 and the rest are 0.)

In mathematics, the restriction to a finite number of intersection types is intentional: it ensures the class of representable surfaces is rich enough to formulate topological questions, without introducing superfluous geometric details. Informally, any unnecessary "wrinkles" have already been "pulled tight." For isosurfacing, however, we want to capture as much detail as possible—including all the wrinkles.

We hence break with the standard philosophy of normal surfaces, adopting *edge coordinates* as our primary representation. Edge coordinates simply count how many times each grid edge pierces the surface; the only mild restriction is that intersections must occur at isolated points. Traditionally, edge coordinates are viewed as auxiliary data derived from normal coordinates: each triangle or quad increments the count for three or four edges, *resp.* Unlike normal coordinates, edge coordinates do not describe a finite number of disk

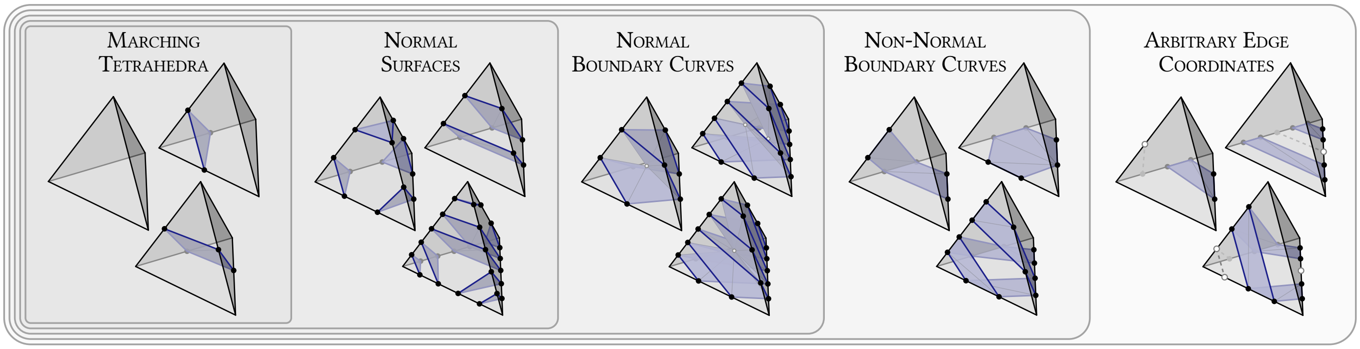
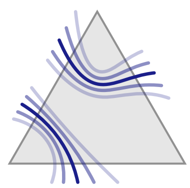


Fig. 4. Examples of surfaces reconstructed from edge intersections. Classic marching tetrahedra considers at most one intersection per edge *(far left)*, whereas we define connectivity for arbitrary patterns of edge intersections of increasing complexity *(left to right)*. See supplement for more comprehensive tables.

types (such as triangles and quads). Yet as we show in this paper, one can still uniquely reconstruct the surface from edge coordinates (in some cases up to a small set of symmetries), via a fast algorithm local to each tet (Section 3).



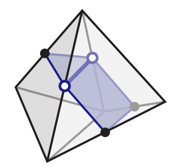
Moreover, intersection counts alone do not pin down the precise geometry of the surface, but rather an *isotopy class* of normal surfaces—*i.e.*, they tell us what the surface looks like only up to continuous deformations that do not cross grid vertices (see inset for a 2D example). For isosurfacing, we also seek the best geometric approximation of the target surface $\mathcal{S}$. We hence enrich integer intersection counts with locations and (optionally) normal vectors for each intersection point.

To date, normal surface theory is little used in geometry processing and visualization. The marching tetrahedra algorithm reinvented a small piece of this story [Doi and Koide 1991], but the isosurfacing literature makes no reference to the broader theory—apart from a brief mention by Hass and Trnkova [2020], whose *GradNormal* algorithm still constructs only a single disk per tetrahedron. More recently, normal coordinates were used to describe *curves* (rather than surfaces) in the context of *intrinsic triangulations* [Gillespie et al. 2021b,a]. Finally, in computational topology, normal coordinates have been used to reduce the asymptotic complexity of basic operations, relative to an explicit piecewise linear representation [Erickson and Nayyeri 2012; Chambers et al. 2023; Lackenby 2024]. However, we know of no past effort to adapt this theory to general surface processing, as we do in this paper.

1.2 Seifert Surfaces

The final step of our reconstruction algorithm is effectively a method for computing a discrete *spanning surface* or *Seifert surface*, *i.e.*, an embedded, oriented surface with a given boundary curve. This problem is fairly well-explored in visualization and geometry processing [Van Wijk and Cohen 2006], and often framed in terms of *Plateau's problem*: find the surface of minimal area with prescribed boundary—akin to a soap film bound by a loop of wire (Figure 5).

Several methods minimize the area of a fine surface triangulation [Renka and Neuberger 1995; Pinkall and Polthier 1993], but this Lagrangian approach does nothing to prevent intersections between mesh components—which would require an expensive collision potential or repulsive energy [Yu et al. 2021]. More recent, Eulerian methods use a fine regular grid [Wang and Chern 2021] or neural field [Palmer et al. 2022] that precludes intersections by construction. In our case, the topology is known *a priori* (just a union of disks), but the aforementioned solutions are far too heavy in both compute time and output resolution. Instead, our algorithm directly constructs a small spanning triangulation within each tet by carefully considering the taxonomy of possible boundary curves. Rather than minimize area, we aim first and foremost to guarantee the intersection-free property, via simple deterministic rules about how to triangulate polygons and place additional *Steiner points*.



The "soap bubble" picture also provides some intuition for how marching algorithms (including ours) guarantee output that is globally manifold and intersection-free: as long as the triangulation for each cell is intersection-free, contained entirely within the cell, and interpolates a boundary curve shared with neighboring cells (inset), we are simply gluing together embedded disks. Appendix A provides a more rigorous discussion.

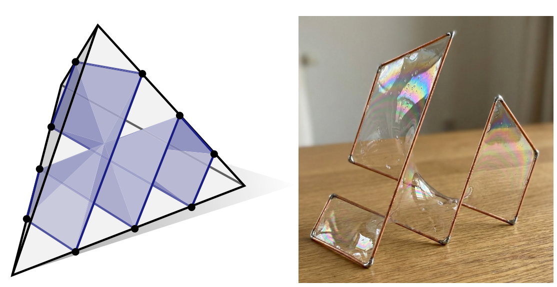


Fig. 5. A core piece of our method is an algorithm for filling a curve on a tet boundary with an interpolating surface—analogous to a minimal-area soap bubble formed by a closed loop of wire.

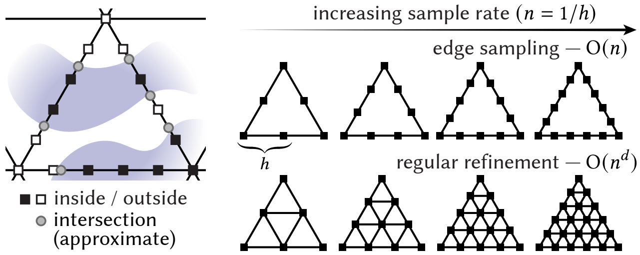


Fig. 6. Our method is a drop-in replacement for standard marching: when exact intersections are unavailable, we can still perform 1D marching along grid edges *(left)*. This approach retains the robustness of subgrid contouring (*e.g.*, capturing thin sheets) while asymptotically reducing the number of samples relative to regular grid refinement in dimension d *(right)*.

1.3 Isosurfacing

Beyond marching, isosurfacing methods build on Delaunay triangulation [Gelas et al. 2009; Binninger et al. 2025], particle simulation [Witkin and Heckbert 1994], and active contours [Cohen and Cohen 1993]; De Araújo et al. [2015] provide a survey. Marching sits at one end of the spectrum: reconstruction is highly localized in space, with rigid assumptions about the input samples. At the other end of the spectrum, one can reconstruct surfaces from arbitrary unstructured point sets [Huang et al. 2024]. Our subgrid method remains firmly in the domain of fixed-grid marching, while dramatically generalizing the sample patterns and output stencils in each cell. As a result, we capture complex surface topology where needed, without over-resolving ordinary smooth regions.

Some past marching methods permit multiple intersections per edge [Bloomenthal and Ferguson 1995; Wyvill and van Overveld 1996], but strongly restrict the output (*e.g.*, at most one additional vertex per cell), precluding robust resolution of features like thin sheets. The method closest in spirit to ours is perhaps Bloomenthal and Ferguson [1995], though its aim is different: contouring nonmanifold geometry—whereas our output is manifold by construction (Theorem A.1). Adaptive grid refinement helps resolve fine features [Bloomenthal 1988; Schaefer et al. 2007; Shen et al. 2023; Shu et al. 1995], but is notoriously difficult to implement without cracks between levels of the hierarchy, and breaks the fixed, regular layout needed for high-performance parallel evaluation. Our subgrid strategy *complements* adaptive methods, since it can be used without modification on a spatially adaptive grid. In essence, subgrid marching helps recover challenging topology (such as thin sheets); grid refinement helps improve geometric detail (*e.g.*, bumps and wrinkles on those sheets).

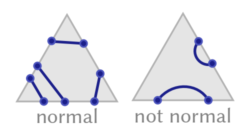
At first glance, one might think that a drawback of our method is the need to find all intersections along each edge—unlike classic marching, which simply evaluates f at each grid node. In reality, however, the subgrid approach is a true drop-in replacement for existing marching algorithms, since *root finding can always be approximated via point sampling*. For instance, we can uniformly sample f along an edge, and use sign changes to identify intersection points—essentially a 1D "marching segments" approach. While this approach can of course miss intersections, we are no worse off than classic marching (which can also miss intersections). At the same time, we asymptotically reduce the number of samples needed to capture fine features relative to regular grid refinement, since we refine only along one dimension, rather than two or three (see Figure 6). Most importantly, our robust reconstruction procedure ensures valid output, even if 1D marching misses some intersections.

In general, as with classic marching, we do not guarantee the output mesh has the same global topology as the surface $\mathcal{S}$. We do, however, ensure that it exhibits the same edge-surface intersections—a *necessary* condition for correct topology, not provided by classic marching. In practice the subgrid approach recovers surface topology far better than classic marching, as demonstrated in Section 4.

2 Background

We first establish some elements of normal surface theory. Though the standard theory is based on *normal coordinates*, a general-purpose contouring algorithm must assign an interpretation to arbitrary *edge coordinates* (Section 2.2). Patterns of edge coordinates arising in normal surfaces (Section 2.3) and *almost normal* surfaces (Section 2.4), namely, triangles, quads, and octagons, serve as important base cases for our more general reconstruction procedure. Before considering normal surfaces, it is helpful to understand *normal curves*, which play an important role in this procedure (Sharp et al. [2021, Section 3.5.1] provide a more thorough introduction).

2.1 Normal Curves



The basic idea of a normal curve or surface is that it is in some sense locally minimal: it cannot be "pulled tighter" to reduce the number of intersection points, without passing through vertices of the triangulation (Figure 7). Hence, a closed curve in a triangulated surface is *normal* if the arcs within each triangle are disjoint, simple, and separate the three vertices into two nonempty sets. It can "cut off corners" of a triangle, but cannot "scoop out edges" (see inset).

We can express normal curves via two kinds of coordinates. *Corner coordinates* $c_0, c_1, c_2 \in \mathbb{Z}_{\geq 0}$ give the number of segments separating each vertex i from the other two vertices j, k. *Edge coordinates* $e_{01}, e_{12}, e_{20} \in \mathbb{Z}_{\geq 0}$ give the number of intersections with each edge.

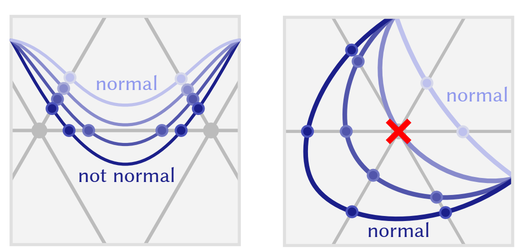


Fig. 7. A curve or surface is normal if the number of intersections cannot be reduced without pulling it across a vertex of the grid. On the left, for instance, the non-normal curve can be pulled tighter, yielding a normal curve with two rather than four intersections. On the right, we can only pull the dark blue normal curve tighter by crossing the marked vertex.

We can convert from corner to edge coordinates by summing up the number of segments crossing the corners at endpoints:

$$e_{01} = c_0 + c_1, \qquad e_{12} = c_1 + c_2, \qquad e_{20} = c_2 + c_0.$$

Inverting this map, we can recover corner coordinates via

$$c_0 = \tfrac{e_{01}+e_{20}-e_{12}}{2}, \quad c_1 = \tfrac{e_{12}+e_{01}-e_{20}}{2}, \quad c_2 = \tfrac{e_{20}+e_{12}-e_{01}}{2}. \tag{2}$$

However, for arbitrary edge coordinates, we can get fractional or negative values. To describe a valid collection of normal curves, edge coordinates must satisfy two conditions:

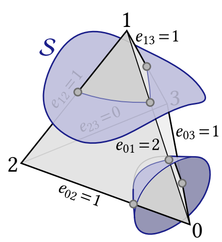
(1) **Even sum.** $e_{01} + e_{12} + e_{20} \equiv 0 \mod 2$
(2) **Triangle inequality.** $e_{ij} + e_{ki} \geq e_{jk}$ for all corners i.

The even sum condition captures the idea that "what goes in, must come out." The triangle inequality captures the idea that arcs entering one edge must exit a different edge.

In our algorithm we decompose arbitrary curves on the boundary of each tet into normal and non-normal parts (Section 3.1), rather than restrict ourselves to normal curves (as in the standard theory).

2.2 Edge Coordinates

Moving to surfaces, we assume that $\mathcal{S}$ meets each grid edge in a finite set of points, all interior to the edge. For a tetrahedron with vertices $(0, 1, 2, 3)$, edge coordinates $e_{ij} \in \mathbb{Z}_{\geq 0}$ count the number of intersections of the surface $\mathcal{S}$ with each of the six edges ij. We write the vector of edge coordinates as



$$\mathbf{e} := (e_{01}, e_{02}, e_{03}, e_{23}, e_{13}, e_{12}) \in \mathbb{Z}^6,$$

offsetting complementary edge pairs (*e.g.*, 01 and 23) by three entries.

In our algorithm, edge coordinates are augmented with locations and (optionally) normals at intersection points (Section 4.3.2), which are not part of classic normal surface theory.

2.3 Normal Surface Coordinates

Similar to curves, a normal surface intersects a tetrahedron in topological disks that separate its four vertices into sets of size 1 and 3 (making four possible *corner cuts*), or 2 and 2 (three *diagonal cuts*).

For a piecewise linear normal surface, where every disk is a planar polygon, corner and diagonal cuts become triangles and quads, *resp.* (Figure 4, *center left*). We use t_i to denote the number of triangles at corner i, and q_{ij} for the number of quads separating edge ij from the complementary edge. The *normal surface coordinates* within a tet are then given by the vector

$$\mathbf{n} := (t_0, t_1, t_2, t_3, q_{01}, q_{02}, q_{03}) \in \mathbb{Z}^7_{\geq 0}.$$

Importantly, for this set of polygons to be intersection-free, there can be at most one type of diagonal cut, *i.e.*, only one of the coordinates q_{ij} can be nonzero.

We can convert normal coordinates to edge coordinates by incrementing edge counts for each type of triangle or quad. *E.g.*, for each triangle at corner 0, we can increment edge coordinates e_{01}, e_{02}, and e_{03}, or equivalently, add the vector

$$\mathbf{v}_0 := (1, 1, 1, 0, 0, 0)$$

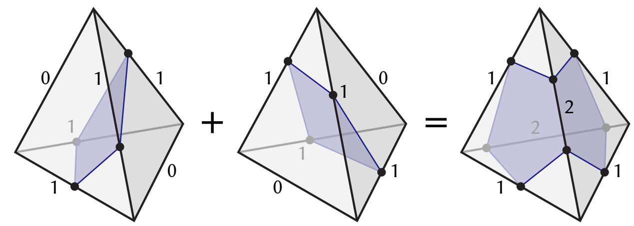


Fig. 8. The theory of *almost normal* surfaces slightly expands the set of edge coordinates that give rise to intersection-free surfaces. In particular, two quads that intersect in the standard theory *(left)* become a single octagon free of intersections *(right)*. We broaden this interpretation much further, recovering non-intersecting geometry from *any* set of edge coordinates.

to $\mathbf{e}$. In general, we can write this relationship as

$$\begin{array}{cccc|ccc} \mathbf{v}_0 & \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_{01} & \mathbf{v}_{02} & \mathbf{v}_{03} \end{array}$$
$$\left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{array}\right] \left[\begin{array}{c} t_0 \\ t_1 \\ t_2 \\ t_3 \\ \hline q_{01} \\ q_{02} \\ q_{03} \end{array}\right] = \left[\begin{array}{c} e_{01} \\ e_{02} \\ e_{03} \\ e_{23} \\ e_{13} \\ e_{12} \end{array}\right] \tag{3}$$

where the matrix $M \in \mathbb{Z}^{6\times 7}$ on the left hand side is the *incidence matrix*, whose columns encode the basic polygon types.

2.4 Almost Normal Surfaces

In contrast, we cannot always explain a given set of edge coordinates via intersections with normal triangles and quads: solutions to Equation 3 may yield negative or fractional coordinates, or describe intersecting polygons. Consider for example the edge coordinates $\mathbf{e} = (2, 1, 1, 2, 1, 1)$ (Figure 8, *right*). Here the solution to Equation 3 is $\mathbf{n} = (0, 0, 0, 0, 0, 1, 1)$, decomposing $\mathbf{e}$ into *intersecting* quads

$$\mathbf{e} = \mathbf{v}_{02} + \mathbf{v}_{03} = (1, 0, 1, 1, 0, 1) + (1, 1, 0, 1, 1, 0). \tag{4}$$

To reconstruct surfaces from general edge coordinates, we must hence broaden our interpretation beyond the basic normal polygons.

Almost normal surfaces [Rubinstein 1994] allow one additional non-normal piece: either an octagon, or a pair of normal disks joined by a tube [Hass 2012]. In particular, the pattern of edge coordinates in Equation 4 is now interpreted as a non-intersecting octagon (Figure 8). The original motivation behind this construction was not to add geometric detail, but rather to encode configurations needed for tasks like *3-sphere recognition* [Rubinstein 1995].

For isosurfacing, almost normal surfaces (and higher-index variants *à la* Bachman [2003, 2012]) remain limited: they still allow only edge intersection patterns $\mathbf{e}$ that are the image under M of vectors $\mathbf{n} \in \mathbb{Z}^7_{\geq 0}$. For instance, edge coordinates $\mathbf{e} = (1, 3, 0, 3, 3, 0)$ (pictured in Figure 5) cannot be encoded by any normal or almost normal surface. However, all such surfaces meet the tetrahedron boundary along a collection of normal curves (Section 2.1). This observation inspires our more general reconstruction procedure from arbitrary normal boundary curves—as well as non-normal curves.

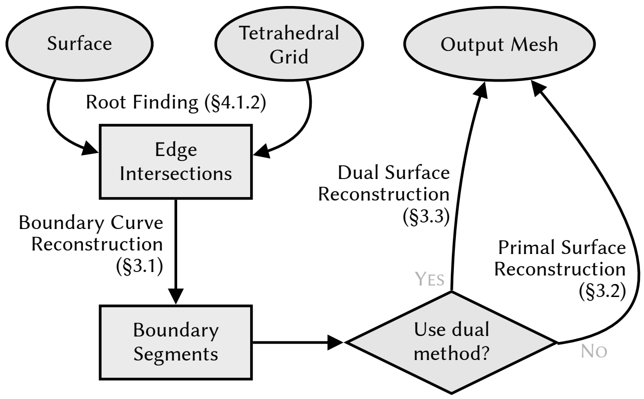


Fig. 9. High-level stages of our algorithm.

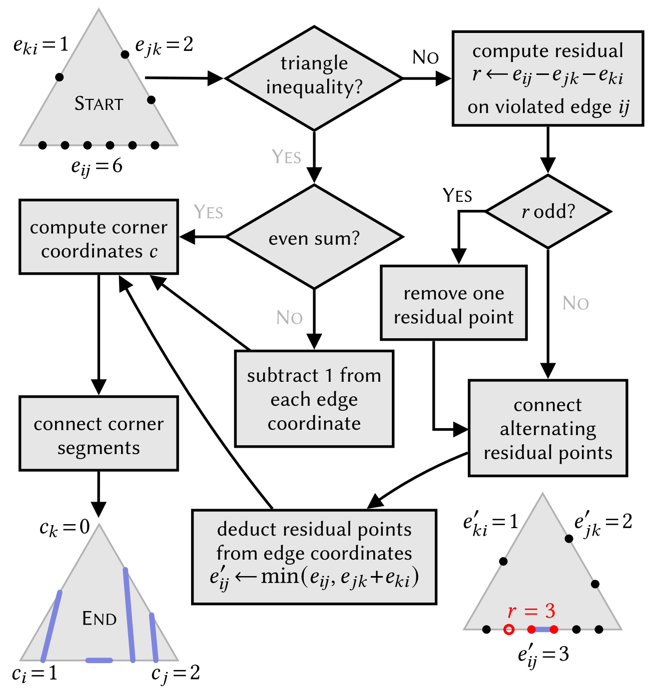


Fig. 10. Procedure for reconstructing a curve (blue) that interpolates an arbitrary set of edge intersection points (black) on each face of a tetrahedron.

3 Reconstruction Algorithms

Our reconstruction algorithm takes edge coordinates, intersection points, and (optionally) normals as input, and produces a triangle mesh approximating the surface $\mathcal{S}$. We first build a curve on each tet boundary interpolating the given intersection points (Section 3.1). We then construct triangles via one of two paths (Figure 9):

- **Primal reconstruction** (Section 3.2) — suitable when a no-intersection guarantee is desired.
- **Dual reconstruction** (Section 3.3) — suitable for geometry with sharp features. Requires normals, and lacks a no-intersection guarantee.

We put no conditions on input edge coordinates (apart from nonnegativity), and lift standard restrictions on the output: unlike marching tets, we can emit multiple polygons per tet (rather than just one); unlike normal and almost normal surfaces, polygons can have arbitrarily many sides (not just triangles, quads, and octagons).

Input. More precisely, the input consists of:

- An embedded tetrahedral mesh $\mathcal{T} = (V, E, F, T)$ with vertex positions $v : V \to \mathbb{R}^3$, where V, E, F, T denote vertices, edges, triangular faces, and tetrahedra.
- For each edge $ij \in E$ with $i < j$:
  - The number of intersections $e_{ij} \in \mathbb{Z}_{\geq 0}$.
  - Intersection locations as 1D barycentric coordinates $0 < s_1 < \ldots < s_{e_{ij}} < 1$, encoding points $(1-s)v_i + sv_j$.
  - *Dual method only:* Unit normals $n_1, \ldots, n_{e_{ij}} \in \mathbb{R}^3$ at each intersection point. The sign of each normal is ignored.

In practice we do not store an explicit tetrahedral mesh, and instead implicitly split the cubes of a regular grid (Section 4.3.1).

Output. The output is a manifold triangle mesh, possibly with boundary. It is closed and orientable if edge coordinates (i) are zero for all edges in the boundary of $\mathcal{T}$, and (ii) exhibit an even sum $e_{ij} + e_{jk} + e_{ki} \equiv 0 \mod 2$ for each triangle $ijk \in F$. These conditions are automatically satisfied when edge coordinates come from a closed surface $\mathcal{S}$ strictly contained in $\mathcal{T}$. If the primal algorithm is used, the mesh will also be free of self-intersections.

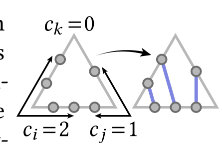
Proofs of these properties are found in the appendix.

3.1 Boundary Curve Reconstruction

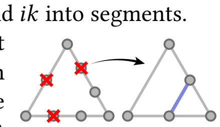
For each tetrahedron in $\mathcal{T}$, we first construct a collection of disjoint, simple curves $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$ on the tet boundary, by applying the procedure outlined in Figure 10 to each triangular face. We carefully define this procedure to produce identical curves on triangles shared by neighboring tetrahedra.

Explicitly, for each triangle ijk of the tet, we do the following:

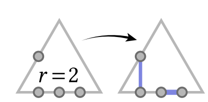
(1) If the edge coordinates satisfy the even sum and triangle inequality conditions (Section 2.1), we compute corner coordinates via Equation 2. For each corner i we then connect up the first c_i pairs of intersection points along oriented edges ij and ik into segments.



(2) If the triangle inequality is satisfied, but the even sum condition is violated, then we subtract 1 from each edge coordinate (*i.e.*, $\mathbf{e} \leftarrow \mathbf{e} - (1, 1, 1)$) and return to Step (1). Doing so ensures that the sum is now even, effectively creating three open endpoints.



(3) Finally, if the triangle inequality is violated, there will be exactly one edge ij with $r := e_{ij} - e_{jk} - e_{ki}$ *residual points*, *i.e.*, with more "incoming" segments than can be absorbed by the other two edges jk, ki. In this case, we first construct as many corner cuts as we can, by applying Step (1) to adjusted edge coordinates $e'_{ij} := \min(e_{ij}, e_{jk} + e_{ki})$. Then, to handle residual points:



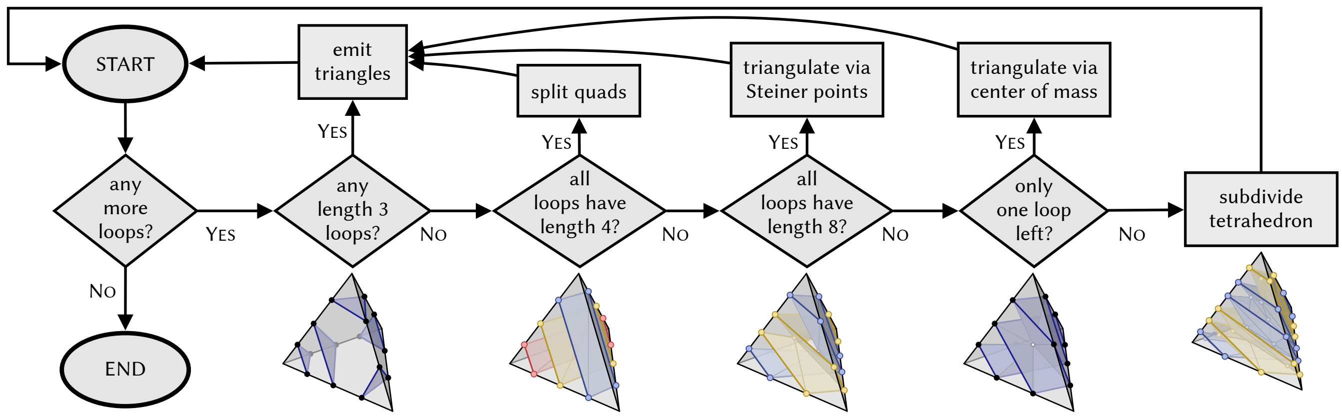


Fig. 11. Given normal loops on the boundary of a tetrahedron, we build spanning disks by first emitting triangles at corners, then considering a small number of mutually exclusive basic cases. Recursive subdivision occurs rarely, requiring a tetrahedron with at least 24 edge intersections ($\mathbf{e} = (2, 4, 6, 2, 4, 6)$).

(a) If r is even, we connect consecutive pairs of points along residual edge ij into segments. These segments correspond to "scoops" (*à la* Section 2.1) of least geometric length.

(b) If r is odd, we skip the first residual point along oriented edge ij (assuming a canonical orientation $i < j$), and connect the remaining consecutive pairs as in the even case.

Segments from each triangle meet at edge points, forming a collection of connected piecewise linear curves $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$ on the tetrahedron boundary. We classify these curves into three types:

- **Open curves.** If γ has a degree-1 vertex, it forms an open curve. Such curves are discarded, but their segments may still appear in neighboring tets as part of the mesh boundary.
- **Normal curves.** If all segments of γ connect two distinct edges, then γ is a normal curve, and will ultimately bound a disk on the tet interior (Figure 4, *center*).
- **Non-normal curves.** Otherwise, if any segment of γ runs along a tet edge, it is a *non-normal* curve, and its spanning disk can include pieces of the tet boundary (Figure 4, *center right*).

3.2 Primal Surface Reconstruction

For each tet in $\mathcal{T}$, we now have a collection Γ of closed loops, and must triangulate them such that the output mesh is intersection-free.

3.2.1 Normal Boundary Curves. We first consider the subset of normal curves $\Gamma_{\text{normal}} \subset \Gamma$, reducing the general case to four cases we can triangulate directly: multiple triangles, parallel quads or octagons, or a single closed loop (Figure 11).

We first handle corner cuts, *i.e.*, we emit a triangle for each loop $\gamma \in \Gamma_{\text{normal}}$ of length $\ell = 3$. We can then establish two properties:

I. All remaining loops have the same length $\ell > 3$ (Theorem B.6).

II. These loops have edge coordinates d_1, d_2, and d_1+d_2 on opposite pairs of edges, for some integers $0 \leq d_2 \leq d_1$ (Theorem B.3).

In other words, the remaining pattern of edge points is equivalent to one arising from a sum of normal quadrilaterals of at most two distinct types. But when $d_2 \neq 0$, we must give a different interpretation to this pattern to recover a nonintersecting surface. In particular:

- **Quads.** If $\ell = 4$, *i.e.*, if the remaining loops are quads, then we split all quads along the same, arbitrary diagonal, producing two triangles per loop.
- If $\ell > 4$, then the remaining m loops must all in fact have length $\ell \geq 8$, since the length of any non-triangular normal loop on a tet is a multiple of 4 (Property I, Theorem B.6). There are hence three possibilities to consider:

(1) **Octagons.** For $\ell = 8$ (octagons), some pair of opposite edges e, e' has $2m$ intersections, and the remaining edges each have m (Equation 4). To triangulate the octagons, we place m Steiner points $x_0, \ldots, x_{m-1}$ uniformly along the oriented segment from the midpoint of e to the midpoint of e'. Pairs of intersections on e are then connected to consecutive Steiner points, from the innermost to outermost pair (Figure 12). *I.e.*, if we enumerate intersections along e as $p_0, \ldots, p_{2m-1}$ (with either orientation), then all points on the loop passing through p_{m+i} are connected to Steiner point x_i.

(2) **Single loop.** If we have just $m = 1$ loop, we insert an additional *Steiner point* x at any point in the convex hull of the loop vertices, and triangulate the loop by connecting each of its edges to x. In practice we let x be the center of mass.

(3) **Subdivision.** Otherwise, $m > 1$ and $\ell > 8$. Noting Property (II) above, let i, j, k, l be vertices of the tetrahedron such that $e_{ij} = e_{kl} = d_1$, $e_{ik} = e_{jl} = d_2$, and $e_{il} = e_{jk} = d_1 + d_2$. (Due

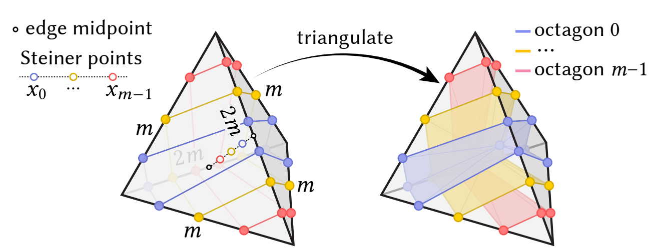


Fig. 12. To triangulate a collection of octagonal loops, we place m evenly-spaced Steiner points along the segment between midpoints of the two edges with $2m$ intersections.

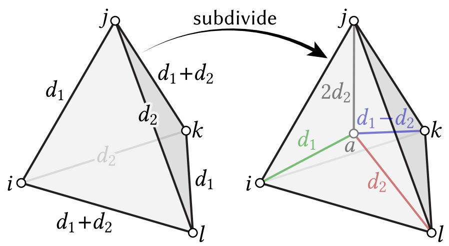


Fig. 13. We reduce arbitrary normal loops to a few simple base cases via subdivision of the edge coordinates, using the stencil shown above.

to the symmetry of edge coordinates, the vertex labeling is not unique; we use lexicographic ordering to break ties.) We subdivide the tet into four by connecting its vertices to the center of mass a (Figure 13), and assign edge coordinates

$$e_{ai} = 2d_2,\ e_{aj} = d_1,\ e_{ak} = d_2,\ e_{al} = d_1 - d_2. \tag{5}$$

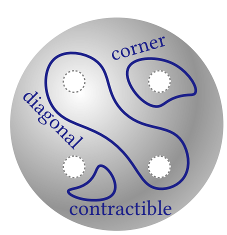
We then recursively process each of the four new tets. The new coordinates describe the simplest normal curves interpolating the observed intersection points, *i.e.*, the smallest coordinates satisfying curve normality conditions from Section 2.1.

Subdivision is needed since, in general, there may be no intersection-free triangulation using a single Steiner point. However, such configurations do not occur below 24 total edge intersections (Figure 11). Moreover, the number of subdivisions is finite, bounded by the sum of initial edge coordinates (Theorem C.1), and far fewer in practice. We do not maintain an explicit tet mesh data structure, but rather just recursively invoke reconstruction for each new tet.

3.2.2 *Non-Normal Boundary Curves.*

Loop type. We next consider the subset of non-normal loops $\Gamma_{\text{nonnormal}} \subset \Gamma$. Viewing a tetrahedron as a topological sphere punctured at its four vertices, each curve $\gamma \in \Gamma_{\text{nonnormal}}$ has one of three types (no matter how much it "spirals" around the tet):

- **Corner type.** Separates one vertex from the other three.
- **Diagonal type.** Separates two vertices from the other two.
- **Contractible.** Does not separate any vertices.



We evaluate the loop type using a parity bit $b_{ij} := \text{mod}\,(e^{\gamma}_{ij}, 2)$ for each edge ij, where e^{γ}_{ij} are edge coordinates for γ alone. This value is odd if γ separates vertices i and j, and even if they belong to the same connected component of the tet boundary. Letting $p := b_{01} + b_{02} + b_{03}$, γ is then contractible if $p = 0$, is of diagonal type if $p = 2$, and is of corner type if $p = 1$ or $p = 3$.

Vertex labels. Since γ is a closed simple curve, it partitions the tet boundary into two pieces. To define spanning disks, we must classify these pieces as "inside" or "outside" γ. When γ is of corner type, we mark the distinguished vertex as inside and all other vertices as outside. When γ is contractible, all vertices are "outside," and when γ is diagonal we do not require an inside/outside distinction.

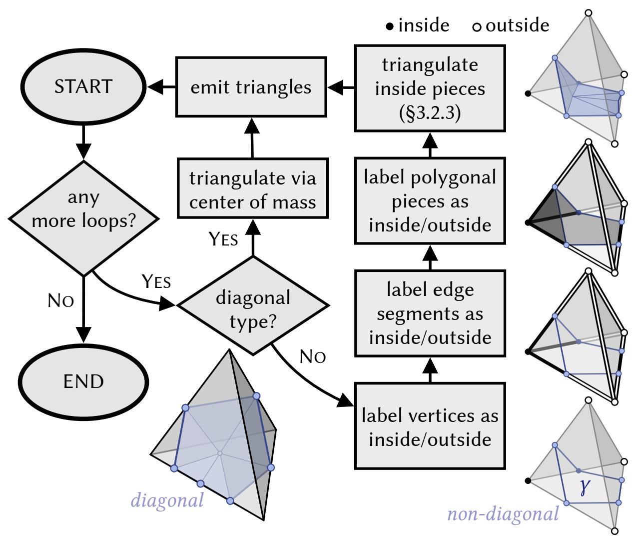


Fig. 14. For non-normal loops, we either insert a single additional point, or dice up the tet boundary along loop segments.

Spanning disks. The final spanning disk depends on loop type:

- **Diagonal.** If γ is of diagonal type, we triangulate it by connecting each of its segments to its center of mass a.
- **Corner and contractible.** Otherwise, we build a piecewise linear disk contained mostly in the tet boundary, rather than its interior. We first split each triangle σ of the tet boundary along the segments of γ, yielding a collection of planar polygons $\sigma \setminus \gamma$. Along each edge ij of σ, the first segment inherits the label of vertex i; subsequent segments alternate between "inside" and "outside" labels. Finally, we emit any polygon P bound by "inside" segments and segments of γ. If one of P's vertices coincides with the "inside" vertex of a corner loop, we omit this vertex from P. For corner-type loops we also emit a triangle at the corner.

3.2.3 *Simplicial Embedding.*

At this point, our mesh has manifold connectivity, but may be an immersed Δ-complex [Hatcher 2002, Section 2.1] rather than an embedded simplicial complex. In particular, we can get two oppositely-oriented copies of polygons bound by non-normal loops γ, associated with two adjacent tetrahedra sharing a face f (Figure 15, *left*). There are three polygon types: a quad, a hexagon, and a pentagon made at a corner. Topologically, these pairs form a manifold tube (quad, pentagon) or punctured sphere (hexagon), but for downstream applications one may require a standard, intersection-free triangle mesh.

We hence "push" polygons into the tet interior: we insert the midpoints of all polygon edges contained in any edge of f, and move them a small distance in the inward normal direction. For pentagons, we also insert the midpoint of the edge opposite the distinguished corner. The triangulation patterns in Figure 15, *right* ensure that inset regions stay within the tetrahedron.

This procedure is needed only when taking the union of the two polygons would yield a nonmanifold edge. If manifold connectivity is not required, one can also just discard one of the polygons.

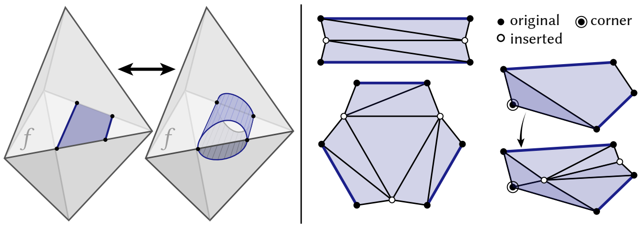


Fig. 15. *Left:* Pairs of identical polygons generated on triangles f shared by two tets define manifold connectivity, but degenerate geometry. *Right:* we obtain an intersection-free result by inserting a few points into each polygon, and pushing them slightly inside each tet.

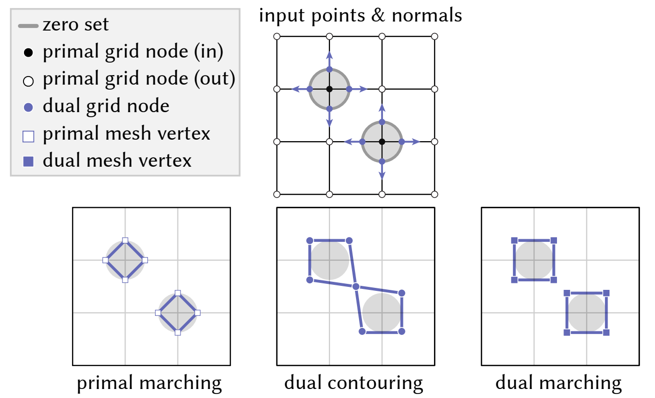


Fig. 16. Dual contouring produces a single vertex per grid cell, whereas dual marching yields a vertex for each component of the primal mesh within each cell—here shown in the context of 2D *marching squares*. Subgrid meshes can have many components per cell, making dual marching a more natural fit.

3.3 Dual Surface Reconstruction

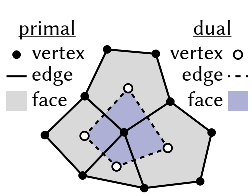
Inspired by *dual contouring* [Ju et al. 2002] and *dual marching* [Schaefer and Warren 2004], our dual subgrid method improves reconstruction accuracy (for equal grid resolution) by allowing output vertices to be placed freely, rather than restricting them to grid edges. Dual contouring associates a single vertex with each 3-dimensional cell of the volumetric grid, whereas dual marching associates a vertex with each 2-dimensional polygon of the primal surface mesh (Figure 16). Traditionally, these two strategies coincide in the tetrahedral case, since classic marching tets produces at most a single polygon per cell (Figure 4, *far left*). For subgrid marching, where we can have many polygons per tet, dual *marching* is the appropriate choice, so that we preserve topologically distinct features.

The greater accuracy of dual methods comes at the risk of self-intersection, since vertex placement is far less restricted. In our subgrid method, however, dual connectivity is much easier to build than in the primal version, since we no longer need a triangulation that supports an intersection-free embedding. Moreover, in situations where an intersection-free guarantee is required, one could still adopt conservative placement strategies, or revert to primal reconstruction for cells with mesh self-intersections (though we do not pursue such strategies here).

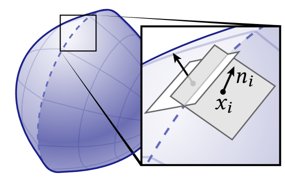
Our dual method proceeds in two steps: first construct the dual mesh connectivity M' (Section 3.3.1), then compute vertex coordinates for this dual mesh (Section 3.3.2).

3.3.1 Dual Connectivity. To obtain mesh connectivity for dual reconstruction, we first construct closed boundary loops Γ for each tet, exactly as in Section 3.1 (skipping the loop triangulation step from Section 3.2). These loops, viewed as (possibly nonplanar) polygonal faces, already define a primal mesh M.

To get the dual connectivity M', we build the usual *Poincaré dual*, replacing each primal polygon with a dual vertex, each interior primal edge with a dual edge connecting the dual vertices from adjacent polygons, and each interior primal vertex with a dual polygon. We then triangulate the polygons of M', without inserting any new vertices.



3.3.2 Dual Geometry. The only remaining question is where to place dual vertices. The basic observation of dual reconstruction methods is that points on sharp features sit (nearly) on the tangent planes of nearby sample points. *E.g.*, a vertex of a cube sits at the intersection of three adjacent faces, and even a curved crease (inset, dashed line) looks more and more like the intersection of two nearby tangent planes as we zoom in.



Quadratic Error Function. More explicitly, let $x_1, \ldots, x_k \in \mathbb{R}^3$ be the edge intersection points along γ, and let N_i be a unit normal at x_i. Ideally, we want a point $p \in \mathbb{R}^3$ at the intersection of the planes P_i passing through x_i and orthogonal to N_i. However, since this intersection may be empty, a standard approach is to find the best-fit point by minimizing the *quadratic error function (QEF)* (or *quadric error metric* [Garland and Heckbert 1997]):

$$Q(p) := \frac{1}{k}\sum_{i=1}^{k} \langle N_i, p - x_i \rangle^2, \qquad \text{(QEF)}$$

where term i equals zero if and only if p sits on plane P_i.

Regularization. When the planes P_i are nearly coplanar, the minimizer of Q can be far from the loop γ, and far outside the tetrahedron. Several corrective devices are explored in the isocontouring literature—we opt for a simple regularized energy

$$Q_\lambda := Q(p) + \lambda \|p - \bar{x}\|^2, \qquad \bar{x} := \frac{1}{k}\sum_{i=1}^{k} x_i,$$

which pulls the minimizer closer to the mean intersection point $\bar{x}$. We use $\lambda = 0.1$ for all examples in this paper.

Notice that input normals need not be consistently oriented, since this energy is invariant with respect to sign flips on the normals n_i. Also note that, in our setting, a loop γ always has at least three vertices x_i; degenerate cases $k = 1$ or $k = 2$ never arise. Moreover, the regularizer ensures a unique minimum even when Q is degenerate (*e.g.*, due to parallel normals).

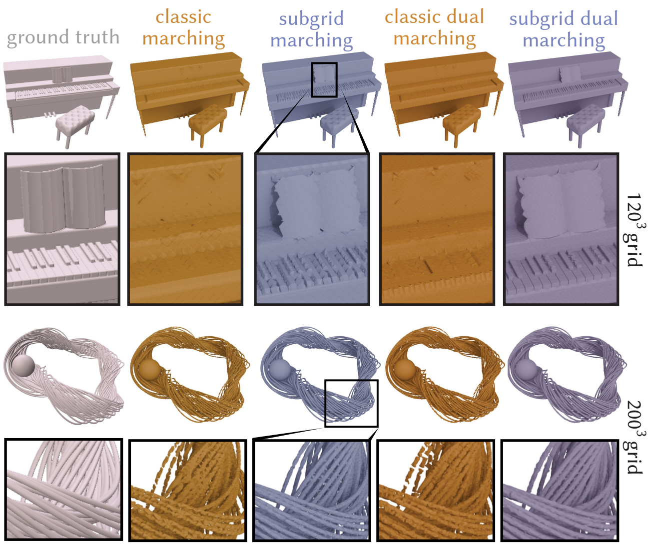


Fig. 17. Comparison of isosurfacing methods on two closed-form SDFs, rendered directly at *far left* via sphere tracing. Our subgrid method is able to preserve thin features like the sheet music and piano keys on the top row, or the woven cables on the bottom row, which are lost or broken by classic marching methods (both primal and dual).

4 Evaluation and Comparisons

To evaluate our method, we compared classic and subgrid marching tetrahedra algorithms on two tasks:

- **Isocontouring.** Given a closed-form SDF $f : \mathbb{R}^3 \to \mathbb{R}$, extract a mesh approximating the zero set $\mathcal{S}$ (Equation 1).
- **Volumetric mesh repair.** Given a polygon soup M_0, produce a manifold approximation M at a target sampling rate (determined by the grid spacing).

Sample results reconstructed from SDFs can be seen in Figure 17, and reconstructions from meshes can be seen in Figures 18, 23, and 24. As expected, even at equal grid resolution, subgrid marching captures small details and thin features lost by classic marching tets.

4.1 Data and Evaluation Metrics

4.1.1 Datasets. For isocontouring we used SDFs from Takikawa et al. [2022], ported to C++ [Baktash et al. 2026]. For mesh repair we used the 3200 model DORA benchmark [Chen et al. 2025], which contains data from Objaverse [Deitke et al. 2023], ABO [Collins et al. 2022], GSO [Downs et al. 2022], and Meta [Dong et al. 2025].

4.1.2 Evaluation Metric. We compute error in the reconstructed mesh M relative to the ground truth surface $\mathcal{S}$ using the *symmetric chamfer distance*

$$d(M, \mathcal{S}) := \frac{1}{2}\left(\frac{1}{|M|}\int_M \min_{y\in\mathcal{S}} \|x-y\|_2 \,\mathrm{d}x + \frac{1}{|\mathcal{S}|}\int_{\mathcal{S}} \min_{x\in M} \|y-x\|_2 \,\mathrm{d}y\right),$$

where $|\cdot|$ denotes surface area. In practice, we approximate each integral by sampling 10k points uniformly at random (with respect to surface area), and perform a closest point query for each sample [Sawhney et al. 2021].

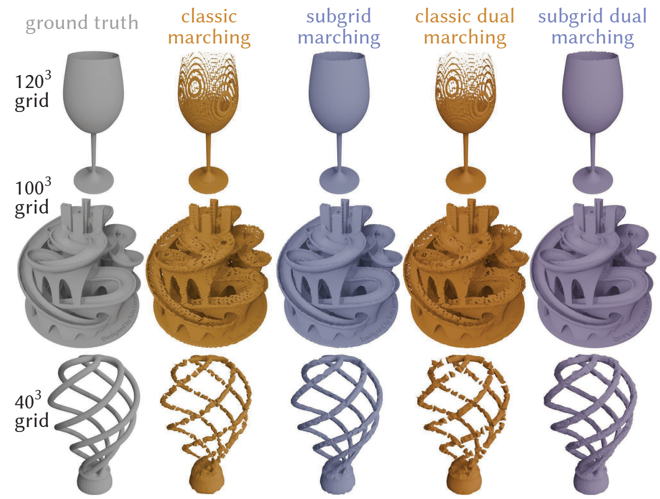


Fig. 18. Even at very low resolutions, our subgrid approach better preserves surface topology—which can have a profound impact on visual fidelity.

4.2 Comparisons

We compared our primal and dual subgrid methods to classic marching tetrahedra [Doi and Koide 1991], including a dual variant using the regularized QEF from Section 3.3.2. As expected, both classic and subgrid methods converge to zero error with increasing grid resolution (Figure 19, *left*).

Geometric accuracy. For equal grid resolution and comparable compute time, we achieve lower reconstruction error than classic methods across diverse inputs (Figure 19). However, better accuracy does not result from merely taking more samples of the input surface $\mathcal{S}$: for example, in Figure 20 we achieve the same accuracy as classic marching with far fewer samples (71M vs. 125M). Simultaneously, we get a dramatically smaller output mesh, with ~7x fewer triangles. In general, subgrid marching is often more sample efficient than classic marching, and never significantly worse. Improved per-triangle accuracy is also consistent across the large DORA dataset, as shown in Figure 21. Statistics for examples from individual figures are provided in Section 2 of the supplement.

Topology preservation. Chamfer distance alone does not tell the full story: even at very low resolutions (*e.g.*, 100^3) where neither method captures much *geometric* detail, the subgrid method does a far better job of preserving the *topology* of thin features—yielding a mesh that is still usable, rather than one with unacceptable holes. See for instance Figure 18, *top*, Figure 23, *top*, and Figure 24, *top*.

Exact intersections are not enough. Appendix E describes an ablation where we compare subgrid and classic marching to an intermediate method that (like ours) places vertices at exact edge-surface intersection points, but that (like classic marching) emits at most one polygon per tetrahedron. This approach improves accuracy slightly compared to classic marching (by around 10–20%), but the inability to produce subgrid-resolution output means it still suffers from the same large-scale topological artifacts—emphasizing the importance of using *all* intersections in the output mesh.

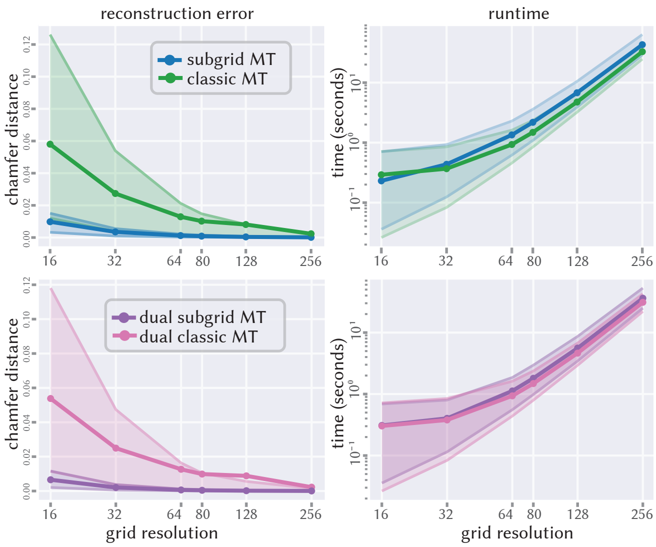


Fig. 19. Subgrid marching yields significantly lower error than classic marching *(left)* for nearly identical compute time *(right)*. Here the solid line shows the mean and the shaded region shows the 10th–90th percentile, across the 3200 models from the DORA dataset.

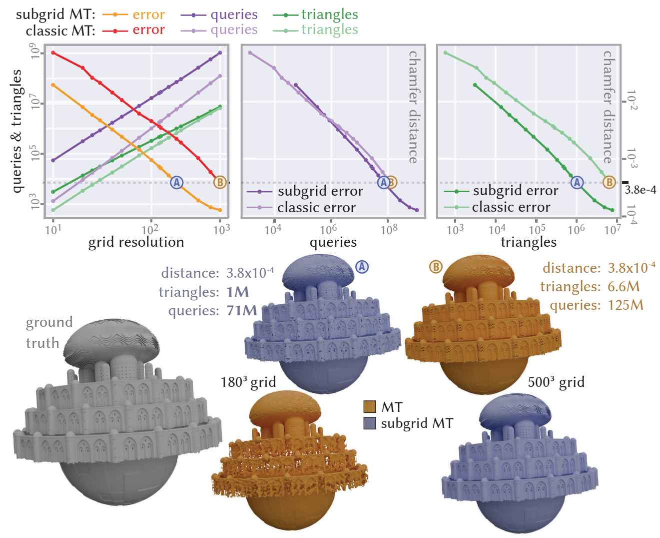


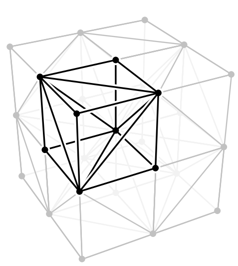
Fig. 20. Subgrid marching outperforms classic marching, independent of how we quantify the difference. Here for instance we achieve better approximation of the ground truth surface at the same grid resolution *(top left)*, the same number of input queries *(top center)*, and the same number of output triangles *(top right)*. Error is measured via chamfer distance, relative to ground truth. *Bottom*: The output of subgrid MT on a 180^3 grid has, in this case, comparable error to classic MT on a 500^3 grid.

4.3 Implementation

We implemented both our method and classic marching in a common C++ codebase, using identical data structures and code for all methods, apart from the core contouring routines. We used a single-threaded implementation, though the algorithm is straightforward to parallelize over grid cells using standard patterns for marching cubes/tets—we leave such acceleration to future work. All timings were measured on a Mac Studio with an M1 Ultra CPU and 128 GB of memory.

Attached to this PDF (and in supplemental material), we include a standalone HTML/JavaScript implementation and visualizer of our reconstruction algorithm for a single tet. This routine is the core of our method, analogous to the usual table of 256 stencils for marching cubes [Lorensen and Cline 1987]; all other steps (e.g., iterating over grid cells) are analogous to those from standard marching implementations.

4.3.1 Tetrahedral Grid. An attractive feature of simplicial marching algorithms, including our subgrid scheme, is that they apply to any simplicial mesh—whether regular or unstructured. For simplicity and efficiency, however, we split each cube of a regular grid into five tetrahedra (see inset), avoiding the need to build or store an explicit tetrahedral mesh. To obtain a conforming triangulation, the tessellation of each cube is reflected across faces shared by two cubes (variously known as an *alternating* or *checkerboard 5-tet cubic-grid*). Throughout we use N^3 to denote a $N \times N \times N$ grid.



4.3.2 Finding Intersections. For many classes of implicit surfaces, there are already algorithms that reliably find all intersections along a given ray $r(t)$ (developed for visualization via ray tracing):

algebraic surfaces	polynomial root finding [Hanrahan 1983]
SDF / Lipschitz	sphere tracing [Hart 1996]
neural implicits	interval analysis [Sharp and Jacobson 2022]
harmonic functions	Harnack tracing [Gillespie et al. 2024]

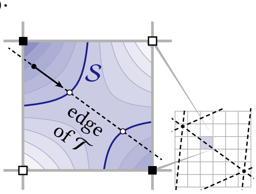
We can apply these same algorithms to find intersections between the surface $\mathcal{S}$ and each edge ij, using the ray

$$r(t) := v_i + t(v_j - v_i),$$

and keeping only those intersections found strictly on the edge interior $0 < t < 1$.

For dual marching, we must also evaluate unit normals at each intersection point. When isocontouring an implicit function f we estimate normals using a finite difference approximation of the gradient ∇f; for volumetric mesh repair we simply evaluate the normals of the input polygonal mesh M_0.

An important special case are grid-based functions f (such as those from CT scans or level set based simulation [Osher and Fedkiw 2003]), which define low-order piecewise polynomials per regular grid cell (*e.g.*, via multilinear interpolation). Here, roots are easily computed in closed form, making it possible to directly contour high-res data at a lower output resolution—without losing important topological features.



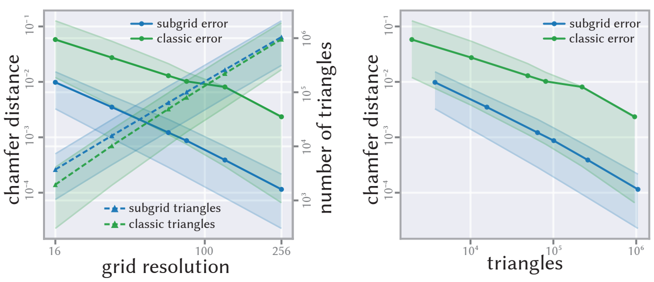


Fig. 21. Our subgrid method generally produces fewer polygons than classic marching for equal reconstruction error. Here we plot the relationship between error and triangle count on the DORA dataset. Center lines show mean performance; shaded regions indicate the 10th–90th percentile.

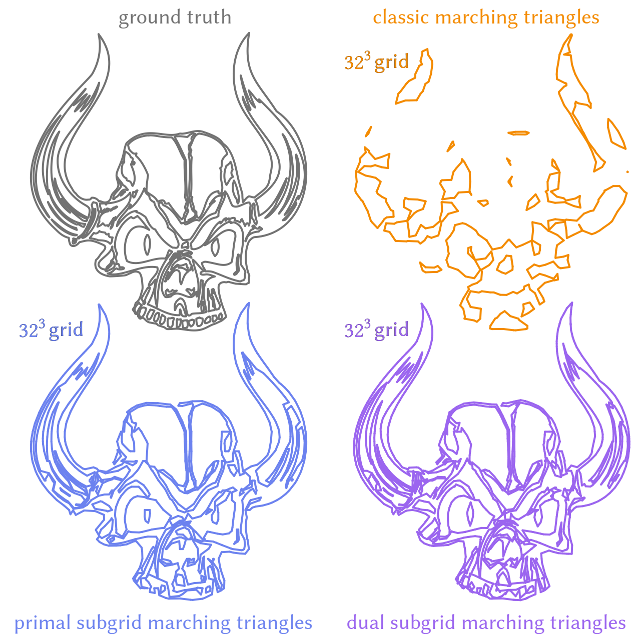


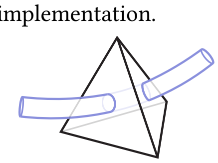
Fig. 22. The subgrid approach can also be applied on a triangular grid in 2D, using the same per-triangle procedure used to construct curve segments on a tetrahedron boundary. As in the 3D case, subgrid accuracy resolves complex geometry missed by ordinary marching triangles.

4.4 Subgrid Marching Triangles

Our procedure for reconstructing a curve within a single triangle also gives us a subgrid version of the *marching triangles* algorithm for contouring functions in two dimensions: we simply apply this subroutine to all triangles in a 2D triangular grid, given intersection counts e_{ij} on grid edges. Since we no longer need to worry about compatibility between neighboring tets, we can make one small modification: in the odd sum case, rather than subtracting 1 from all 3 edge coordinates in a face, we instead subtract 1 from the largest edge coordinate, breaking ties arbitrarily. For the dual method, we use the same regularized QEF as in Section 3.3.2, but in two dimensions. Figure 22 shows a comparison between this subgrid method and classic marching triangles. Note that, as in 3D, our reconstruction procedure makes no assumptions about the structure of the triangle grid, and can hence be applied to arbitrary unstructured triangulations (including surface meshes).

5 Limitations and Future Work

Speed Improvements. The *per-tetrahedron* cost of our algorithm can of course be greater than that of marching tets—since in general we may produce more polygons per tet. However, when considering the amortized cost *per output triangle*, the speeds are quite comparable: the main overhead is the additional logic of our local reconstruction procedure. In cases where speed (or thread synchronization) is critical, it may be beneficial to precompute a table of common stencils (say, for tets with edge coordinate sum below some fixed number), rather than computing them on the fly. We did not attempt these kinds of optimizations in our implementation.



Just as classic marching can easily miss thin 2-dimensional sheets not sampled by any node, our subgrid approach can easily miss thin 1-dimensional curve- or tube-like features not sampled by any edge (see inset). Further generalizing reconstruction to manifolds of codimension $m > 1$ is an interesting question for future work.

Although the mesh we construct in the primal case is formally manifold, we must introduce additional elements to support a simplicial embedding (Section 3.2.3). Another good question for future work is whether we can instead transform the edge coordinates such that they describe simpler spanning disks (*e.g.*, those bound by normal curves).

At the moment we do not carefully treat the question of how to split quads into triangles—as explored by Shen et al. [2023] in the context of the *FlexiCubes* algorithm, additional splitting weights might help our method to better capture sharp features. More generally, since our approach does not fundamentally change the overall structure of standard marching algorithms, it should be possible in the future to incorporate recent extensions to classic marching—such as differentiable optimization of grid nodes [Shen et al. 2021].

Acknowledgments

This work was funded by the National Science Foundation under awards 2212290 and 2504890. The authors wish to thank Dave Bachman, Saul Schleimer, and Eric Sedgwick for helpful conversations. The soap film image in Figure 5 was generated by Nano Banana Pro [Google 2025], conditioned on a render of a discrete minimal surface. The marble track model in Figure 18 is designed by Tulio Laanen, and the vase model in Figure 28 by Hiroaki Nishimura.

References

David Bachman. 2003. 2-normal surfaces. arXiv:math/0309437 [math.GT]
David Bachman. 2012. Normalizing Topologically Minimal Surfaces II: Disks. arXiv:1210.4574 [math.GT]
Hossein Baktash, Mark Gillespie, and Keenan Crane. 2026. *Conservative SDF Dataset.* https://github.com/geometrycollective/sdf-dataset
Alexandre Binninger, Ruben Wiersma, Philipp Herholz, and Olga Sorkine-Hornung. 2025. TetWeave: Isosurface Extraction using On-The-Fly Delaunay Tetrahedral Grids for Gradient-Based Mesh Optimization. *ACM Transactions on Graphics (TOG)* 44, 4 (2025), 1–19. doi:10.1145/3730851

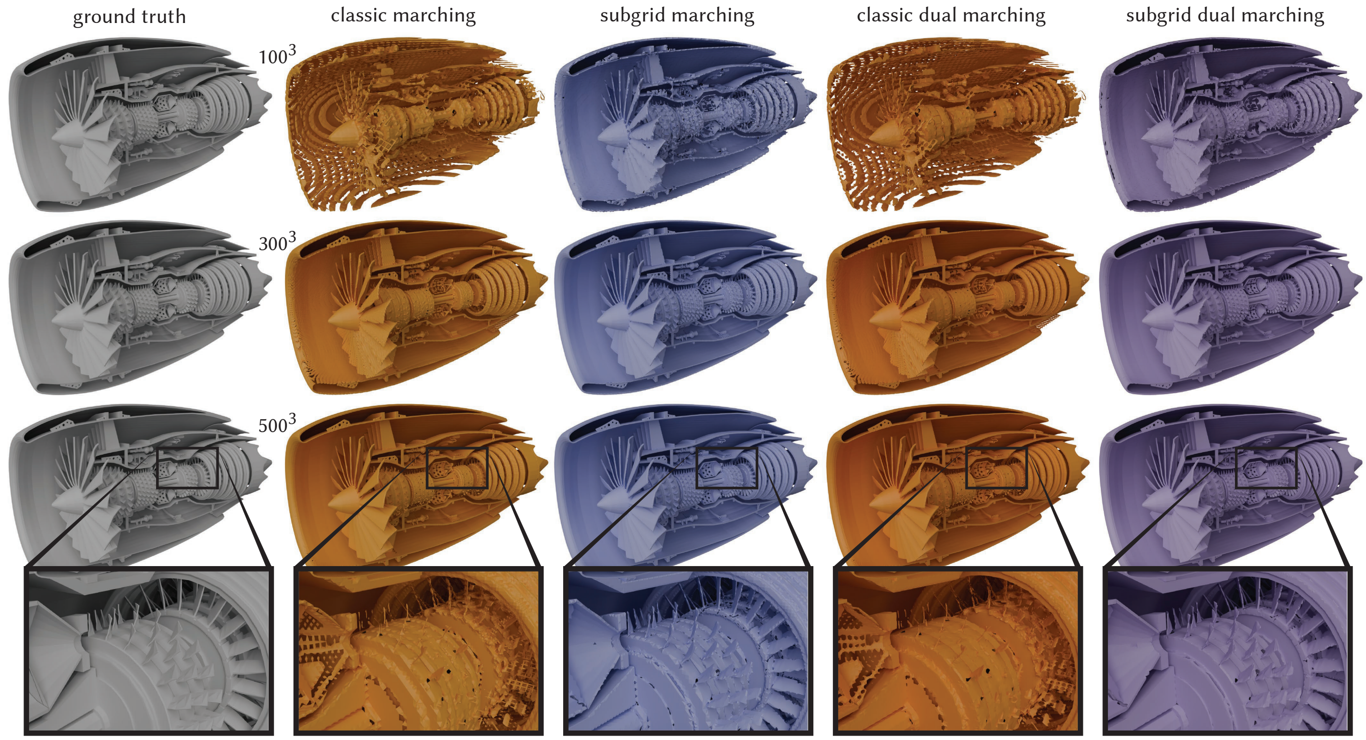


Fig. 23. In this complex jet engine model, even a resolution of 500^3 is insufficient for classic marching tets to capture all the small topological features. In contrast, the subgrid methods immediately capture most small details (at 100^3 resolution), and largely just improve their geometry as resolution increases.

Jules Bloomenthal. 1988. Polygonization of implicit surfaces. *Computer Aided Geometric Design* 5, 4 (1988), 341–355. doi:10.1016/0167-8396(88)90013-1

Jules Bloomenthal and Keith Ferguson. 1995. Polygonization of non-manifold implicit surfaces. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. ACM, New York, NY, USA, 309–316. doi:10.1145/218380.218462

Erin Wolf Chambers, Francis Lazarus, Arnaud de Mesmay, and Salman Parsa. 2023. Algorithms for Contractibility of Compressed Curves on 3-Manifold Boundaries. *Discrete & Computational Geometry* 70, 2 (2023), 323–354. doi:10.1007/s00454-022-00411-x

Rui Chen, Jianfeng Zhang, Yixun Liang, Guan Luo, Weiyu Li, Jiarui Liu, Xiu Li, Xiaoxiao Long, Jiashi Feng, and Ping Tan. 2025. Dora: Sampling and Benchmarking for 3D Shape Variational Auto-Encoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 16251–16261. doi:10.1109/CVPR52734.2025.01515

Laurent D Cohen and Isaac Cohen. 1993. Finite-element methods for active contour models and balloons for 2-D and 3-D images. *IEEE Transactions on Pattern Analysis and machine intelligence* 15, 11 (1993), 1131–1147. doi:10.1109/34.244675

Jasmine Collins, Shubham Goel, Kenan Deng, Achleshwar Luthra, Leon Xu, Erhan Gundogdu, Xi Zhang, Tomas F Yago Vicente, Thomas Dideriksen, Himanshu Arora, Matthieu Guillaumin, and Jitendra Malik. 2022. ABO: Dataset and Benchmarks for Real-World 3D Object Understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 21094–21104. doi:10.1109/CVPR52688.2022.02045

Bruno Rodrigues De Araújo, Daniel S Lopes, Pauline Jepp, Joaquim A Jorge, and Brian Wyvill. 2015. A survey on implicit surface polygonization. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 1–39. doi:10.1145/2732197

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. 2023. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 13142–13153. doi:10.1109/CVPR52729.2023.01263

Akio Doi and Akio Koide. 1991. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems* 74, 1 (1991), 214–224.

Zhao Dong, Ka Chen, Zhaoyang Lv, Hong-Xing Yu, Yunzhi Zhang, Cheng Zhang, Yufeng Zhu, Stephen Tian, Zhengqin Li, Geordie Moffatt, Sean Christofferson, James Fort, Xiaqing Pan, Mingfei Yan, Jiajun Wu, Carl Yuheng Ren, and Richard Newcombe. 2025. Digital Twin Catalog: A Large-Scale Photorealistic 3D Object Digital Twin Dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 753–763. doi:10.1109/CVPR52734.2025.00079

Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. 2022. Google scanned objects: A high-quality dataset of 3d scanned household items. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2553–2560. doi:10.1109/ICRA46639.2022.9811809

Jeff Erickson and Amir Nayyeri. 2012. Tracing Compressed Curves in Triangulated Surfaces. In *Proceedings of the twenty-eighth annual symposium on Computational geometry*. 131–140. doi:10.1145/2261250.2261270

Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 209–216. doi:10.1145/258734.258849

Arnaud Gelas, Sébastien Valette, Rémy Prost, and Wieslaw L Nowinski. 2009. Variational implicit surface meshing. *Computers & Graphics* 33, 3 (2009), 312–320. doi:10.1016/j.cag.2009.03.016

Mark Gillespie, Nicholas Sharp, and Keenan Crane. 2021a. Integer Coordinates for Intrinsic Geometry Processing. *ACM Transactions on Graphics (TOG)* 40, 6, Article 252 (2021), 13 pages. doi:10.1145/3478513.3480522

Mark Gillespie, Boris Springborn, and Keenan Crane. 2021b. Discrete Conformal Equivalence of Polyhedral Surfaces. *ACM Transactions on Graphics (TOG)* 40, 4, Article 103 (2021), 20 pages. doi:10.1145/3450626.3459763

Mark Gillespie, Denise Yang, Mario Botsch, and Keenan Crane. 2024. Ray Tracing Harmonic Functions. *ACM Trans. Graph.* 43, 4 (2024), 99–1.

Google. 2025. Nano Banana Pro: Gemini 3 Pro Image model from Google DeepMind. https://blog.google/innovation-and-ai/products/nano-banana-pro/. Accessed: 2026-04-30.

Wolfgang Haken. 1961. Theorie der Normalflächen: ein Isotopiekriterium für den Kreisknoten. *Acta Mathematica* 105, 3-4 (1961), 245–375. doi:10.1007/BF02559591

Pat Hanrahan. 1983. Ray tracing algebraic surfaces. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '83)*. ACM,

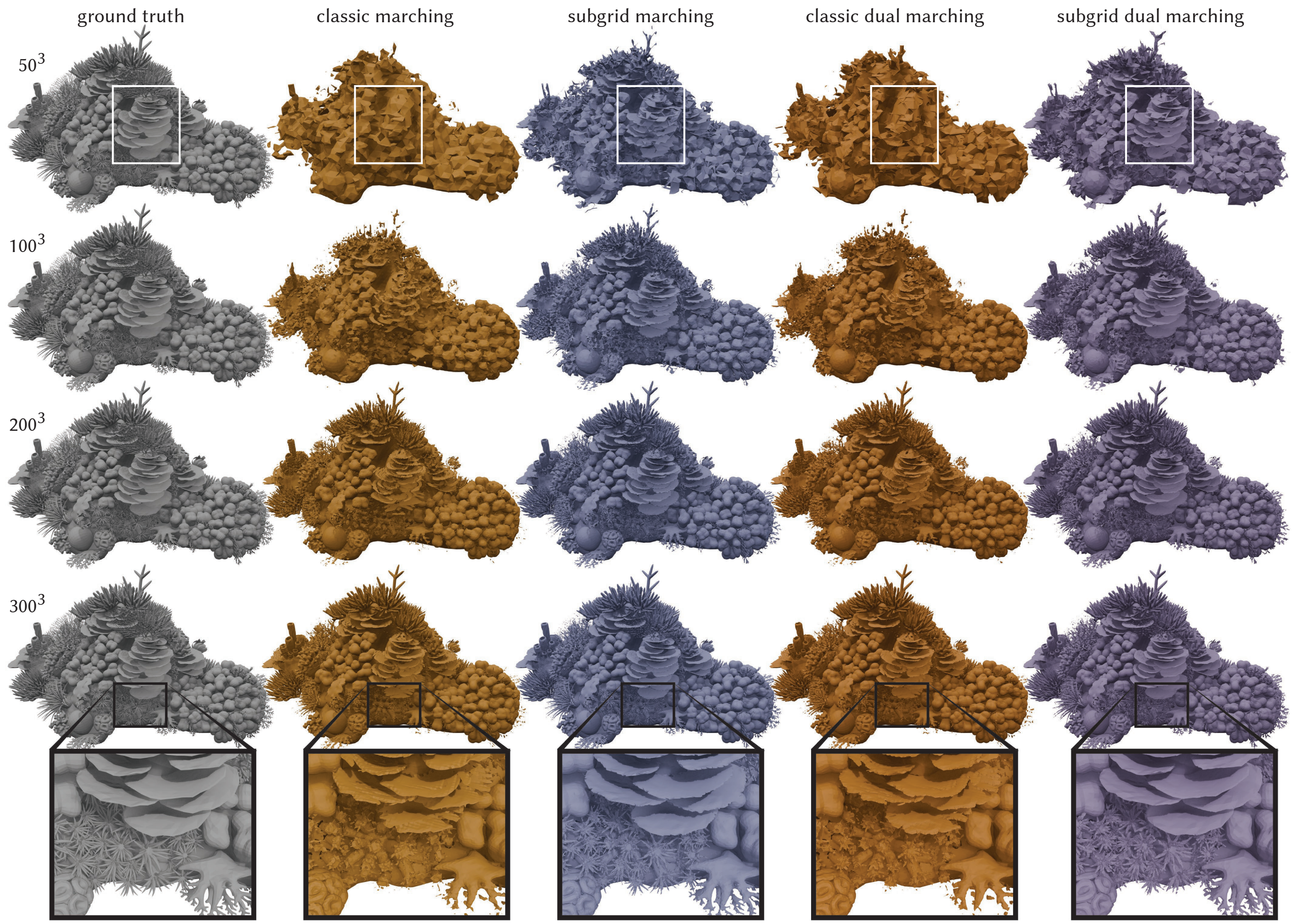


Fig. 24. Beyond geometric accuracy, resolving fine topological features is essential for capturing object semantics. Here, for instance, even at the coarsest resolution of 50^3, our subgrid method makes it clear that this coral reef contains leafy *foliose coral* (outlined in white), whereas classic marching tets conveys only the bulk distribution of this reef.

83–90. doi:10.1145/800059.801136

John C Hart. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10 (1996), 527–545. doi:10.1007/s003710050084

Joel Hass. 2012. What is an Almost Normal Surface. arXiv:1208.0568 [math.GT]

Joel Hass and Maria Trnkova. 2020. Approximating isosurfaces by guaranteed-quality triangular meshes. *Computer Graphics Forum* 39, 5 (2020), 29–40. doi:10.1111/cgf.14066

Allen Hatcher. 2002. *Algebraic Topology.* Cambridge University Press. https://pi.math.cornell.edu/~hatcher/AT/ATpage.html

ZhangJin Huang, Yuxin Wen, ZiHao Wang, Jinjuan Ren, and Kui Jia. 2024. Surface Reconstruction From Point Clouds: A Survey and a Benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 12 (Dec. 2024), 9727–9748. doi:10.1109/TPAMI.2024.3429209

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. *ACM Transactions on Graphics (TOG)* 21, 3 (July 2002), 339–346. doi:10.1145/566654.566586

Hellmuth Kneser. 1929. Geschlossene Flächen in dreidimensionalen Mannigfaltigkeiten. *Jahresbericht der Deutschen Mathematiker-Vereinigung* 38 (1929), 248–259. doi:10.1515/9783110894516.147

Marc Lackenby. 2024. Some fast algorithms for curves in surfaces. arXiv:2401.16056 [math.GT]

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87).* ACM, New York, NY, USA, 163–169. doi:10.1145/37401.37422

Sergei Matveev. 2007. *Algorithmic topology and classification of 3-manifolds.* Springer. doi:10.1007/978-3-540-45899-9

Stanley Osher and Ronald Fedkiw. 2003. *Level Set Methods and Dynamic Implicit Surfaces.* Vol. 153. Springer New York, New York, NY. doi:10.1007/b98879

David Palmer, Dmitriy Smirnov, Stephanie Wang, Albert Chern, and Justin Solomon. 2022. DeepCurrents: Learning Implicit Representations of Shapes with Boundaries. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 18665–18675. doi:10.1109/CVPR52688.2022.01811

Ulrich Pinkall and Konrad Polthier. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics* 2, 1 (1993), 15–36. doi:10.1080/10586458.1993.10504266

Robert J Renka and JW Neuberger. 1995. Minimal Surfaces and Sobolev Gradients. *SIAM Journal on Scientific Computing* 16, 6 (1995), 1412–1427. doi:10.1137/0916082

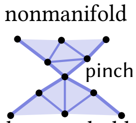
Joachim Hyam Rubinstein. 1994. *Polyhedral minimal surfaces, Heegaard splittings and decision problems for 3-dimensional manifolds.* Department of Mathematics, University Melbourne.

Joachim H Rubinstein. 1995. An algorithm to recognize the 3-sphere. In *Proceedings of the International Congress of Mathematicians: August 3–11, 1994 Zürich, Switzerland.*

Springer, 601–611. doi:10.1007/978-3-0348-9078-6_54
Rohan Sawhney, Ruihao Ye, Johann Korndoerfer, and Keenan Crane. 2021. FCPW: Fastest closest points in the west. https://github.com/rohan-sawhney/fcpw
Scott Schaefer, Tao Ju, and Joe Warren. 2007. Manifold Dual Contouring. *IEEE Transactions on Visualization and Computer Graphics* 13, 3 (2007), 610–619. doi:10.1109/TVCG.2007.1012
Scott Schaefer and Joe Warren. 2004. Dual marching cubes: Primal contouring of dual grids. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.* IEEE, 70–76. doi:10.1109/PCCGA.2004.1348336
Nicholas Sharp, Mark Gillespie, and Keenan Crane. 2021. Geometry Processing with Intrinsic Triangulations. In *ACM SIGGRAPH 2021 Courses (SIGGRAPH '21)*. ACM, 79 pages. https://nmwsharp.com/media/papers/int-tri-course/int_tri_course.pdf
Nicholas Sharp and Alec Jacobson. 2022. Spelunking the Deep: Guaranteed Queries on General Neural Implicit Surfaces via Range Analysis. *ACM Transactions on Graphics (TOG)* 41, 4, Article 107 (jul 2022), 16 pages. doi:10.1145/3528223.3530155
Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. 2021. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems* 34 (2021), 6087–6101.
Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. 2023. Flexible Isosurface Extraction for Gradient-Based Mesh Optimization. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–16. doi:10.1145/3592430
Renben Shu, Chen Zhou, and Mohan S Kankanhalli. 1995. Adaptive marching cubes. *The Visual Computer* 11, 4 (1995), 202–217. doi:10.1007/BF01901516
Towaki Takikawa, Andrew Glassner, and Morgan McGuire. 2022. A Dataset and Explorer for 3D Signed Distance Functions. *Journal of Computer Graphics Techniques (JCGT)* 11, 2 (27 April 2022), 1–29. http://jcgt.org/published/0011/02/01/
Jarke J Van Wijk and Arjeh M Cohen. 2006. Visualization of Seifert surfaces. *IEEE Transactions on Visualization and Computer Graphics* 12, 4 (2006), 485–496. doi:10.1109/TVCG.2006.83
Stephanie Wang and Albert Chern. 2021. Computing minimal surfaces with differential forms. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–14. doi:10.1145/3450626.3459781
Andrew P Witkin and Paul S Heckbert. 1994. Using particles to sample and control implicit surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. 269–277. doi:10.1145/192161.192227
Brian Wyvill and Kees van Overveld. 1996. Polygonization of implicit surfaces with constructive solid geometry. *International Journal of Shape Modeling* 02, 04 (1996), 257–274. doi:10.1142/S0218654396000142
Chris Yu, Caleb Brakensiek, Henrik Schumacher, and Keenan Crane. 2021. Repulsive surfaces. *ACM Transactions on Graphics (TOG)* 40, 6, Article 268 (Dec. 2021), 19 pages. doi:10.1145/3478513.3480521

A Manifold Property

In the even-sum case—when the edge coordinates for all faces of the grid obey the even sum condition of Section 2.1—both the primal and dual versions of our algorithm are guaranteed to produce a manifold output. If the even-sum condition is violated, the primal polygon mesh generated in Section 3.2 is still edge-manifold everywhere, and vertex-manifold at interior vertices, but it may contain nonmanifold "pinches" at the boundary. Note, though, that such pinches are easily split into several manifold vertices, allowing our primal and dual algorithms to produce manifold outputs even when the even-sum condition does not hold.

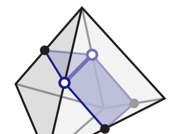


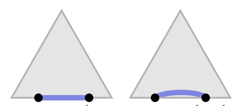
Theorem A.1. *In the even sum case, both the primal and dual algorithms yield manifold connectivity,* i.e.*, an abstract simplicial 2-complex homeomorphic to a manifold without boundary. In the odd sum case, the primal algorithm yields an edge-manifold simplicial complex, which only has non-manifold vertices on the boundary.*

Proof. The main difficulty lies in showing that the cell complex formed before Section 3.2.3 is manifold or edge-manifold as appropriate, which is done in Lemma A.2. Triangulating each polygon *à la* Section 3.2.3 turns the mesh into a simplicial complex (since each polygon's triangulation is itself a simplicial complex) homeomorphic to the original cell complex. Thus, the primal output is manifold or edge-manifold as appropriate. For the dual algorithm, note that triangulating the dual of a manifold simplicial complex (possibly with boundary) yields another manifold simplicial complex, which has boundary if and only if the original had boundary. □

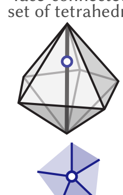
Lemma A.2. *The primal polygons constructed before Section 3.2.3 form an edge-manifold cell complex. In the even-sum case, the cell complex is also guaranteed to be vertex manifold, and in either case interior vertices are always manifold.*

Proof. At a high level, the cell complex inherits its manifold properties from the background tetrahedral grid. First, we show that the cell complex is edge manifold, meaning that no edge belongs to more than two polygons. The main idea is that we can associate each segment along a polygon's boundary to a face in the background tetrahedral grid that it runs through. Since each face of our background grid is incident to one or two tetrahedra, such an edge can only belong to one or two polygons. Note that this reasoning applies even to "scoop" edges (Section 2.1) which geometrically run along the edges of the tetrahedral grid. These scoop edges are constructed in Section 2.1 by constructing segments from their edge coordinates in some face, and even an infinitesimal perturbation is enough to push them into their associated face. Thus, when defining the mesh connectivity these scoop edges are associated to at most two polygons, and are thus manifold edges in the cell complex. But note that as depicted in Figure 15 (*left*) multiple scoop edges may connect the same pair of vertices.





To conclude, we show that in the even-sum case the cell complex is vertex manifold. Since every edge is manifold, it suffices to check that the faces neighboring each vertex i form a single edge-connected component. Note that every edge in the tetrahedral grid is contained in a single face-connected set of tetrahedra. In the even-sum case, each tetrahedron produces one polygon containing the vertex, and two such polygons share an edge if the corresponding tetrahedra share a face, so the neighboring polygons do indeed form a single edge-connected component.



If the even-sum condition does not hold, some neighboring tetrahedra may not produce a polygon incident on i. But if i is an interior vertex then each of its neighboring tetrahedra must have produced an incident polygon, and i must be vertex-manifold. □

B Tetrahedral Normal Curves

Here we establish some basic properties of normal curves on a tetrahedron. Note that we assume only that the *curves* obey the normality conditions from Section 2.1 on the boundary faces—they are not required to bound normal *surfaces* inside of the tetrahedron.

B.1 Normal Coordinate Conversion

We label vertices of our tetrahedron with i, j, k, l, and denote the edge coordinates with e_{ij}, e_{ik}, e_{il}, e_{jk}, e_{jl}, e_{kl}, and label the normal coordinates by t_i, t_j, t_k, t_l, q_{ij}, q_{ik}, and q_{il}. Note that q_{ij} is the number of diagonal cuts that separates vertices i, j from k, l and so we have $q_{ij} = q_{kl}$; and they both represent the same quantity. We use them interchangeably for ease of notation at various places.

First we show that Equation 3 has an integer solution if and only if edge coordinates obey the normality conditions (triangle inequality and even sum). We then use this fact to decompose the edge coordinates into corner and diagonal coordinates (t_i and q_{ij}).

Theorem B.1. *Equation 3 has a nonnegative integer solution if and only if normality conditions are satisfied. In this case, there is a unique nonnegative solution where one of the three q_{ij} values is zero.*

Proof. The RREF format of the linear system in Equation 3 is:

$$\left[\begin{array}{ccccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{e_{01}+e_{02}-e_{12}}{2} \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & \frac{e_{01}-e_{03}+e_{13}}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & \frac{e_{02}-e_{03}+e_{23}}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & \frac{-e_{12}+e_{13}+e_{23}}{2} \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & \frac{-e_{01}+e_{03}+e_{12}-e_{23}}{2} \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & \frac{-e_{02}+e_{03}+e_{12}-e_{13}}{2} \end{array}\right], \tag{6}$$

so the solutions are non-negative integers if and only if even-sum condition and triangle inequality is met on all faces ijk. We also see that the kernel of the constraint matrix is generated by the vector which adds 1 to each each corner coordinate t_i and subtracts 1 from each diagonal coordinate q_{ij}. So given any nonnegative solution, there exists a unique shift which sets the smallest q_{ij} to zero. □

Lemma B.2. *Let Γ be a collection of normal curves on the boundary of a tetrahedron. And let $\mathbf{n} = (t_i, t_j, t_k, t_l, q_{ij}, q_{ik}, q_{il})$ be the corresponding normal coordinates defined in Theorem B.1. Then at each vertex i we have exactly t_i triangles (size 3 polygons).*

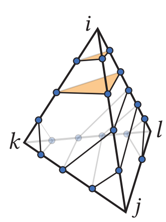
Proof. Since normality is satisfied, we and the integer solution $\mathbf{n}$ to Equation 3, then for every edge ij we can write:

$$e_{ij} = t_i + t_j + q_{il} + q_{ik} \tag{7}$$

Consider a face ijk of the tetrahedron. The number of normal segments at corner i, denoted by c^i_{jk} is given by:

$$c^i_{jk} = \frac{e_{ij} + e_{ik} - e_{jk}}{2} \tag{8}$$

$$= t_i + q_{jk} \tag{9}$$



Thus we have at least t_i normal segments at every face incident to vertex i. These segments circling vertex i get connected up to triangles. So we have at exactly $\min\{t_i + q_{jk}, t_j + q_{kl}, t_i + q_{lj}\}$ triangles around vertex i; for example, the inset shows a vertex i with $t_i = 2$, and $q_{il} = q_{ij} = 0$. Since at least one of q_{jk}, q_{jl}, q_{kl} is zero (by Theorem B.1), we have exactly t_i triangles around vertex i. □

Theorem B.3. *Let Γ be a triangle-free collection of normal curves on the boundary of a tetrahedron,* i.e. *a set of normal curves with no curve of length 3. Then there are pairs of edges with edge coordinates d_1, d_2, and $d_1 + d_2$ for integers $0 \le d_2 \le d_1$.*

Proof. This is a direct consequence of theorem B.2. After extracting and removing t_i triangles from every vertex i, the new edge coordinates of a face ijk can be written as:

$$e_{ij} = q_{ik} + q_{il} \tag{10}$$

$$e_{jk} = q_{ik} + q_{ij} \tag{11}$$

$$e_{ki} = q_{ij} + q_{il} \tag{12}$$

At least one of the q values is zero (due to B.1). Denoting the other two by d_1 and d_2, we can see that the edge coordinates above are exactly the values d_1, d_2, and $d_1 + d_2$ in some order. □

From here on we call this configuration of curves the (d_1, d_2) pattern. See Figure 25 *(top left)* for an example.

B.2 Connected Components of Normal Curves

Here we analyze the connected components a triangle-free set of normal curves on a tetrahedron. By Theorem B.3, we know that the curves must form a (d_1, d_2) pattern.

Theorem B.4. *The number of connected components of a (d_1, d_2) pattern is exactly* $\gcd(d_1, d_2)$.

Proof. We show this by constructing an operation that shortens the curves without breaking or merging them, turning the (d_1, d_2) pattern into a $(d_1 - d_2, d_2)$ pattern.

To better illustrate this process, we imagine cutting the tetrahedron open and collapsing it onto a line segment, as shown in Figure 25. The flattening does not change the connectivity of our curves, and the connectivity reduces to the following pattern:

- We have $2(d_1 + d_2)$ points on a line segment.
- The first (leftmost) $2d_2$ points are connected along the top of the segment in a radial pattern, *i.e.*, the first one to the last one and the two middle ones to each other.
- The next $2d_1$ points are connected similarly along the top.
- The first $d_1 + d_2$ points are connected to the last $d_1 + d_2$ points along the bottom of the segment in the same radial fashion.

If two points are connected from the top of the segment, we call the top-pairs and likewise we call pairs of points connected on the bottom of the segment bottom-pairs.

Our shortening operation is as follows. Take the first $2d_2$ points and move them towards their corresponding bottom-pairs, from the bottom of the segment until they merge.

This operation removes the leftmost $2d_2$ points and yields a new connectivity for the $2d_1$ remaining points:

- All the $2d_1$ points are connected above the segment.
- The rightmost $2d_2$ points are connected below the segment.
- The leftmost $2(d_1 - d_2)$ points are connected below the segment.

All connections are radial as before. Hence the new connectivity is identical to before, with new values $(d_1 - d_2, d_2)$ instead of (d_1, d_2).

Repeating this process performs the Euclidean algorithm on the pair (d_1, d_2), eventually terminating at a pair $(d, 0)$, where $d = \gcd(d_1, d_2)$. Since the diagram for the pair $(d, 0)$ is a set of d concentric circles, there are exactly d connected components. And since we never changed the topology of the curves, the initial curves also must have had d components. □

Theorem B.5. *All loops in a (d_1, d_2) pattern have the same number of segments, which we call the length ℓ.*

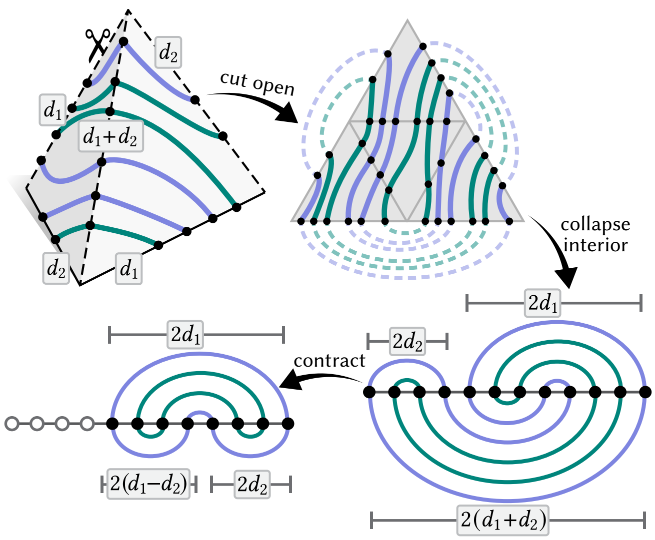


Fig. 25. Here we illustrate our proof of Theorem B.4, that the number of curves on the boundary of a tetrahedron with two non-zero diagonal cuts d_1, d_2 is exactly $\gcd(d_1, d_2)$. We begin by cutting the tetrahedron open and collapsing the resulting triangle to a line segment, creating a topologically-equivalent set of curves which cross the line segment at $2(d_1 + d_2)$ points. Contracting the leftmost d_2 loops yields the diagram for the pair $(d_1 - d_2, d_2)$. Iterating this procedure reveals a set of $\gcd(d_1, d_2)$ connected curves.

Proof. Consider the terminal state of the process outlined in the proof of Theorem B.4, and color the points with $d = \gcd(d_1, d_2)$ colors. The color of every point determines its connected component. When reversing the operation, the difference between the number of points of different colors remains fixed. There is an equal number of points of each color in the terminal state, so in the initial flattened stage, each connected component has an equal contribution on the flattened edge. The choice of which edge to collapse the triangle on was arbitrary in the proof of Theorem B.4, so the every component has equal contribution at every edge of the tetrahedron. Hence all components must have equal length. □

Corollary B.6. *The length of every component in a (d_1, d_2) pattern is exactly $4\frac{d_1+d_2}{\gcd(d_1,d_2)}$. Consequently, when $d_1, d_2 \geq 1$, the number of segments is at least 8.*

Proof. Each of the tetrahedron's four faces contains exactly $d_1 + d_2$ segments. Since these segments make up $\gcd(d_1, d_2)$ curves (Theorem B.4) and each curve has the same length (Theorem B.5), we conclude that each curve has length $4\frac{d_1+d_2}{\gcd(d_1,d_2)}$. □

C Algorithm Termination

Theorem C.1. *For a tetrahedron with $\sum_{ij} e_{ij} = n$, the primal reconstruction algorithm in Section 3.2.1 terminates after $O(n)$ subdivisions.*

Proof. First, note that due to our choice for the interior edge coordinates, normality conditions are satisfied for all the new tetrahedra, allowing the splitting procedure defined in Equation 5 to continue recursively until termination.

Now we bound the recursion depth. Before splitting the tetrahedron, the edge coordinates adjacent to each vertex are d_1, d_2, and $d_1 + d_2$ (assume $d_1 > d_2$). Splitting creates a new vertex with edge coordinates $d_1, d_2, 2d_2$, and $d_1 - d_2$ on its incident edges. So long as $d_1 \neq d_2$, the new edge coordinates are all strictly less than $d_1 + d_2$ (since $d_2 \leq d_1$), and so the maximum edge coordinate of at least one vertex must decrease. And if $d_1 = d_2$, we are in the octagon case which is handled explicitly without requiring further subdivision.

Thus, after at most $d_1 + d_2$ subdivision steps all interior tetrahedra either have a 0 edge coordinate, or they are at the base case, where $d_1 = d_2$. Hence the process terminates in linear time. □

D No Intersection Property

Our main theorem about the geometry of the primal reconstruction algorithm is the following:

Theorem D.1. *The piecewise linear surface produced via the primal reconstruction algorithm in Section 3.2 is globally embedded in $\mathbb{R}^3$,* i.e., *it exhibits no self-intersections.*

We will show that these surfaces have no intersections *locally* inside each tetrahedron. Since the surfaces are always contained in their tetrahedra by construction (up to the ε-perturbation in Section 3.2.3), the global surface will also be intersection free.

To prove this theorem, we must first establish several key lemmas about our primal reconstruction algorithm in Section 3.2.

Lemma D.2. *Every primal disc we construct, whether bounded by a normal or non-normal boundary curve γ, lies in the convex hull of γ.*

Proof. If γ is a triangle or quad, we triangulate γ, which always produces a disk lying in the convex hull. For the surfaces that we construct via inserting a Steiner point, the Steiner point is contained in the convex hull of γ and so the surface also is. The remaining surfaces are constructed on the boundary faces of the tetrahedron (smeared against the tet faces); these regions are enclosed between segments of γ and are thus contained within its convex hull. □

Next we go through the different types of surfaces that we construct and prove that they cannot intersect. Note that after analyzing a class of surfaces, we can disregard that class during later analyses.

Theorem D.3. *Triangles do not intersect any of our surfaces.*

Proof. Consider a vertex i of the tetrahedron. The triangles $T_1, \ldots, T_k$ at i do not intersect each other. And they lie in the convex hull of the final triangle T_k and the vertex i so they cannot intersect any other surfaces either. □

Now we are free to assume that no corner triangles exist in our later intersection-free arguments.

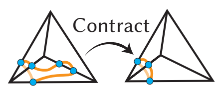
Theorem D.4. *Surfaces constructed for contractible type non-normal curves do not intersect any of our surfaces.*

Proof. A non-normal curve γ of contractible type is not contained in another curve of the same kind (Lemma D.7). All the surface polygons emitted for this type of curve are constructed on the tet boundary. We call these the *smeared* polygons. These smeared polygons are strictly within γ's region on the boundary of the tet and since γ is not contained nor contains another curve, these polygons do not intersect other surfaces. Hence the constructed surface does not intersect itself or other surfaces. ◻

Now we are free to assume that no such contractible non-normal curves exist in our later intersection-free arguments.

Before considering other non-normal curves, we define a **contraction** operation that can be applied to such curves. We use this operation in several proofs throughout this section.

Definition D.5. For a non-normal boundary curve γ, the contraction process pulls γ tight at non-normal segments until it becomes normal. The discrete version of this operation is as follows: on an edge of the tetrahedron where γ has a scoop (segment running along an edge), reduce the edge coordinate by 2; this terminates when γ is normal.



Note that during the contraction process the smeared polygons remain smeared and or are removed entirely, and hence no new intersections arise, and no existing intersections are removed. And so if the surface for γ does not intersect other surfaces, then it remains intersection-free at every stage of the contraction process.

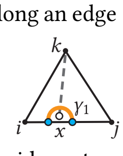
Theorem D.6. *Surfaces constructed for corner-type non-normal curves do not intersect any of our surfaces.*

Proof. A corner-type curve γ also contains some smeared polygons similar to those in contractible type curves (Theorem D.4). With the same argument as before, we see that these polygons do not intersect themselves or others. Now consider the vertex i of the tetrahedron that γ loops around. The final polygon used to fill in γ is a corner triangle at vertex i. To show that this triangle also cannot intersect any other surfaces, we contract (see D.5) the non-normal curve until it becomes a normal corner curve at vertex i. Once the contraction is done, we only have a corner triangle at vertex i. This triangle does not intersect other surfaces due to Theorem B.2, and so our original surface must also not have intersected any other surfaces in the contraction process. ◻

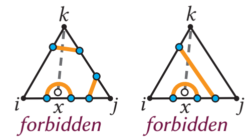
Before proceeding to the next class of surfaces, we prove the key lemma on smeared polygons which we used above in Theorem D.4.

Lemma D.7. *A curve γ of contractible or corner-type can not be contained in the interior of another contractible or corner-type curve.*

Proof. We prove this by contradiction. Consider two non-normal curves γ_1 and γ_2 either of contractible or corner types. Without loss of generality, suppose that γ_1 is contained inside of γ_2. Since γ_1 is non-normal, it must have a scoop (a segment running along an edge of the tetrahedron) on some edge ij of some triangle ijk. Consider a point x in on the face ijk and inside the scoop (almost on the edge ij). Since γ_1 is contained in γ_2, then x is contained in both γ_1 and γ_2. If we ignore all the other curves then every path from a point inside γ_1 to a vertex of the tetrahedron must intersect these two curves an even number of times. But the path from x to vertex k only intersects γ_1 once (only at the scoop that it was a part of), due to Lemma D.8, providing our desired contradiction. ◻



Lemma D.8. *If a scoop exists on triangle ijk along edge ij, then the path connecting any point x inside the scoop to the opposite vertex k cannot cross any other segments. In particular, no 2D corner segment can exist at vertex k on triangle ijk.*



Proof. Our construction for boundary curves (Section 3.1) is an as-normal-as-possible construction. A scoop on edge ij at triangle ijk is drawn if an only if there are residual points on edge ij and so the triangle inequality is violated: $e_{ik} + e_{jk} \leq e_{ij}$. So no normal segment at corner k is constructed. Similarly, all the corner segments of vertex i are on one side of the scoop, closer to i, so the corner cut depicted on the right of the inset is also impossible. ◻

The lemmas and theorems above prove that the no-intersection property holds for surfaces generated for corner triangles, contractible non-normals and corner-type non-normals. As usual, we now discard these curves and show that the other types of surfaces that we construct do not intersect each other.

The remaining curves are (a) normal curves corresponding to sum of two diagonal cuts (d_1, d_2) and (b) non-normal curves of diagonal type. These two types of curves can exist along with all the previously mentioned curves. Also note that both of these types are identified as diagonal types; in the sense that they separate two vertices of the tetrahedron from the other two.

However, we will show in Theorem D.10 that no two diagonal type curves can co-exist if one of them is non-normal.

As a direct result of this theorem, we can show that our surface construction for diagonal-type non-normal curves, and for single component normal curves do not intersect any of our surfaces.

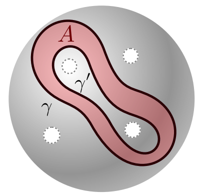
Theorem D.9. *If γ is a diagonal-type non-normal curve, or a normal diagonal curve with a single component ($k = 1$ from Section 3.2), then its corresponding surface does not intersect itself or other surfaces.*

Proof. In both cases, the curve exists as a single component (see Theorem D.10 for the non-normal case). We construct a surface by inserting a single Steiner point and connecting it up with all the segment of γ to make a fan of triangles. So the surface does not intersect itself since γ does not intersect itself. ◻

Next we show that diagonal-type non-normal curves cannot co-exist with the other diagonal-type curves.

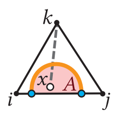
Theorem D.10. *If we have non-normal diagonal-type curve γ on the tetrahedron, no other diagonal-type curve γ', normal or non-normal, can exist on the tetrahedron.*

Proof. Suppose for contradiction that we have a second diagonal-type curve γ' which does not intersect γ. We make two simplifying assumptions without loss of generality. First, we assume that γ and γ' are the only curves on our tetrahedron. If another curve exists (of any type), we can remove it without affecting the rest of the construction: removing a whole curve amounts to removing corner segments and scoops, none of which change the as-normal-as-possible construction of the other curves. Second, γ and γ' must share the same diagonal type (*i.e.* separate the same pair of vertices), since curves of different diagonal types always intersect.



Now consider the simplified picture of γ and γ' in the inset, where the tet is seen as a sphere with four punctures at its vertices (also referenced in Section 3.2.2). Since γ and γ' are of the same diagonal type they separate the same vertices, creating three regions on the sphere. The middle region, which we call A, does not contain any punctures.

We will first show that the inside of any scoop must belong to region A. Then we will show that there must exist some scoop belonging to a different region, providing the desired contradiction.



Consider a face ijk of the tetrahedron with a scoop on edge ij, and take a point x inside the scoop. By Lemma D.8, x is separated from vertex k by a single curve. Since all the vertices lie outside region A, this means that x must lie in region A (since any path leaving a vertex ends up in region A if and only if it intersects γ and γ' an odd number of times.) Hence all scoops on all faces must be in region A.

Now we derive our contradiction by showing that there must be some scoop which is not in region A. Recall that one can always contract both curves (see D.5) until they are both normal. We will show that the reverse of this contraction operation must produce a scoop that is not in region A. Contraction produces two diagonal-type normal curves, both of which have the same diagonal type. Note that all edges must have an even number of intersections (since region A contains no vertices), so our normal curves must form a $(2d_1, 2d_2)$ pattern; the corner segments in this pattern can be seen as a collection of stripes that enclose region A (see Figure 26). Starting from this configuration, Lemma D.11 shows that the reverse of the contraction operation must produce a scoop that is not in region A, providing our desired contradiction. □

Lemma D.11. *If we start from a $(2d_1, 2d_2)$ pattern and perform reversed contractions until reaching an as-normal-as-possible configuration, then we must produce a scoop that is not in region A.*

Proof. First note that a reversed contraction operation can only create new scoops or move an existing scoop to a new face, but it can never remove a scoop or change the region of an existing scoop. So if we ever produce a scoop that is not in region A, we are done.

As depicted in Figure 26, a reverse contraction operation will either pull a corner segment towards one of its neighboring edges of the face ijk, or towards the opposite edge. In the first case, we create a new scoop on each of the neighboring faces, and we can go from the interior of one scoop to the interior of the other by crossing a single curve. Thus the two scoops are in different regions, so at least one must be outside of region A.

In the second case (where a corner segment is pulled towards the opposite edge), our corner segment it must be one of the segments in the middle section of the face ijk (exposed to vertex i). Without loss of generality, take the segment to be a corner segment of vertex k and pulled towards edge ij (Figure 26, *right*). This produces a scoop on edge ij on face ijl which lies between vertex i and any corner segments. So long as d_1 and d_2 are both greater than zero, vertex i

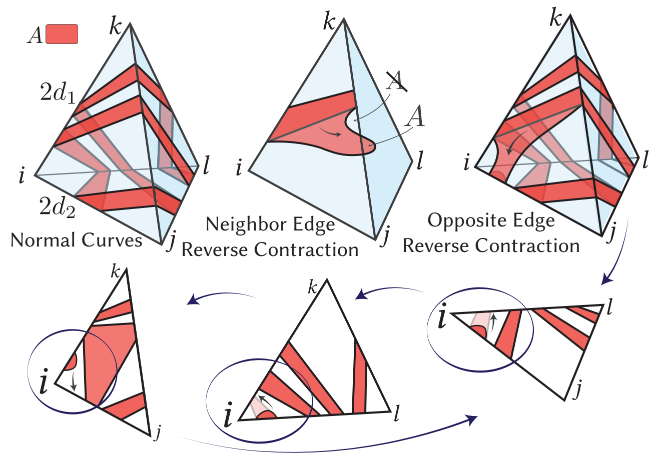
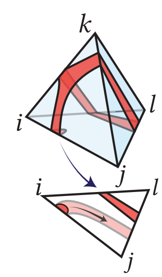


Fig. 26. Illustration of reverse contraction in the proof of Theorem D.10. The initial configuration has no scoops. Reverse contraction either pulls a corner segment towards its neighboring edge (middle), which produces a double-scoop pattern and a scoop that is not in region A; or pulls it towards the opposite edge (right), which produces a scoop that is trapped between corner i and the closest corner segment of vertex i on triangle ijl; this pattern repeats without ever reaching an as-normal-as-possible configuration.

now has at least one corner segment in each face. So even if we continue pulling the scoop into new faces, it will always stay between vertex i and the closest corner segment. Such a configuration can never be as normal as possible. Thus, these configurations can be neglected, as we only care about sequences of reversed contractions that yield an as-normal-as-possible configuration at the end.



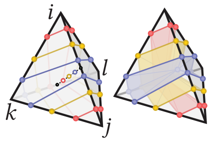
The only remaining case to consider is when $d_2 = 0$. In this case we must also have $d_1 = 2$, since we have exactly two curves on the tetrahedron. Again, consider pulling a scoop down across edge ij. As before, continuing to pull the scoop across edge il leaves the scoop trapped between vertex i and a corner cut, so it is impossible to reach an as-normal-as-possible configuration. We can also pull the scoop across edge jl, but the scoop is then trapped between vertex j and a corner cut, so it is still impossible to reach an as-normal-as-possible configuration.

Thus it is impossible to start from a $(2d_1, 2d_2)$ pattern and reach an as-normal-as-possible configuration whose scoops are all in A. □

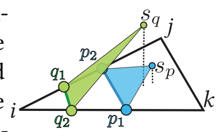
The only remaining case is the octagon case ($k > 1$, $\ell = 8$, see Section 3.2 and Figure 12). In this case, we triangulate k octagonal curves γ_i (akin to almost normal surfaces). The Steiner points we insert for each octagon are placed in a particular order as explained in 3.2 and shown in Figure 12. We show that this choice of Steiner points guarantees no-intersection between different octagons. Note that a single surface we construct for an octagon does not intersect itself with the same argument as D.9.

Theorem D.12. *Consider k octagonal curves γ_i on the boundary of the tetrahedron. Their constructed surfaces from Section 3.2 do not intersect each other.*

Proof. In the octagons configuration, our edge coordinates have a (d, d) pattern; *i.e.*, edges have the coordinates $d, d, 2d$, with opposite edges having the same value (see inset). We select a segment inside the tetrahedron to place the Steiner points. Consider edges ij and kl to have $2d$ intersections. We choose this segment to connect a point between the d'th and $d + 1$'th points (out of $2d$ ordered points) on edge ij to the similar point on edge kl. We prove that within every face ijk, triangles that are made with segments in ijk do not intersect each other. We only need to prove this for one face, and the rest are proved by symmetry.



Take two segments on face ijk, with endpoints p_1, p_2 and q_1, q_2 both at corner i, the latter being closer to corner i, and connected to their Steiner points s_p and s_q that is above the face ijk (see inset); also note that by construction, both Steiner points are on the same side of the segment q_1q_2. For the two triangles p_1, p_2, s_p and q_1, q_2, s_q to not intersect, it is sufficient to have s_q be higher than s_p. Our placement of Steiner points with respect to every face exactly satisfies this condition: if a segment is closer to a corner, its Steiner point has a bigger distance to the face. □



E Ablation: the Subgrid Mod 2 Algorithm

Subgrid marching tetrahedra uses two kinds of information that are unavailable in classic marching: the *number* of intersections e_{ij} along each edge, and the *locations* of the intersections. It is natural to wonder whether both are essential to the improved accuracy of our method.

Here we perform an ablation where we provide the exact input locations, but still limit output to at most one intersection per edge (as in classic marching). In particular, the parity of the intersection count, mod $(e_{ij}, 2)$, determines whether the two endpoints of edge ij are on different sides of a closed surface $\mathcal{S}$. This is enough information for detecting *sign changes* across an edge, as in marching tetrahedra. We then extract an aggregated intersection location along the edge by averaging all of the true intersection positions. Note that if there is only one intersection, the average equals the exact location of the zero set. In short, this modification improves the accuracy of the primal vertex positions, but produces the same mesh connectivity as classic marching tetrahedra.

We plot the performance of this modified marching tetrahedra, which we call "subgrid mod 2," in Figure 27 and show some examples in Figure 28. Subgrid mod 2 improves slightly over classic marching tetrahedra, but produces much more distortion than the full subgrid marching tetrahedra algorithm.

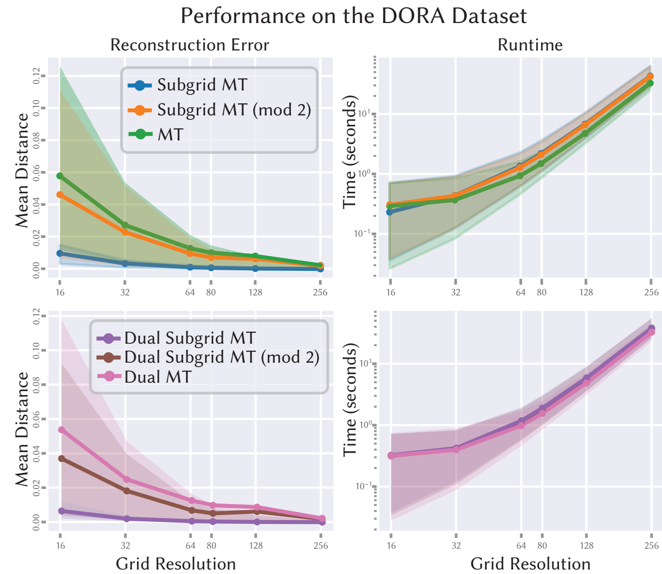


Fig. 27. Here we show the accuracy and performance of the subgrid mod 2 ablation. Its error profile is much closer to classic marching tetrahedra than to subgrid marching tetrahedra, emphasizing the importance of using *all* intersections along each edge.

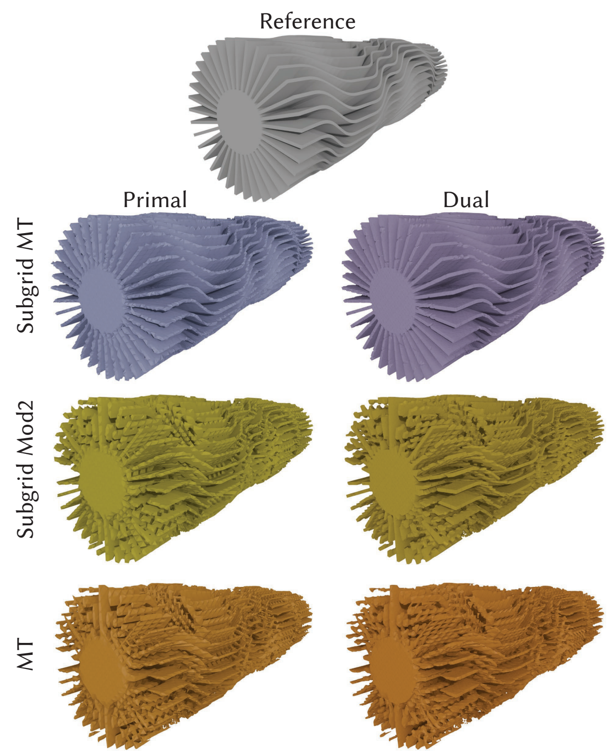


Fig. 28. Comparison of our subgrid approach against its mod 2 version and marching tetrahedra, in a 100^3 grid. Subgrid mod 2 has slightly better geometric quality than marching tetrahedra but still cannot produce more than one disc per tetrahedron.