Subgrid Marching Tetrahedra

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

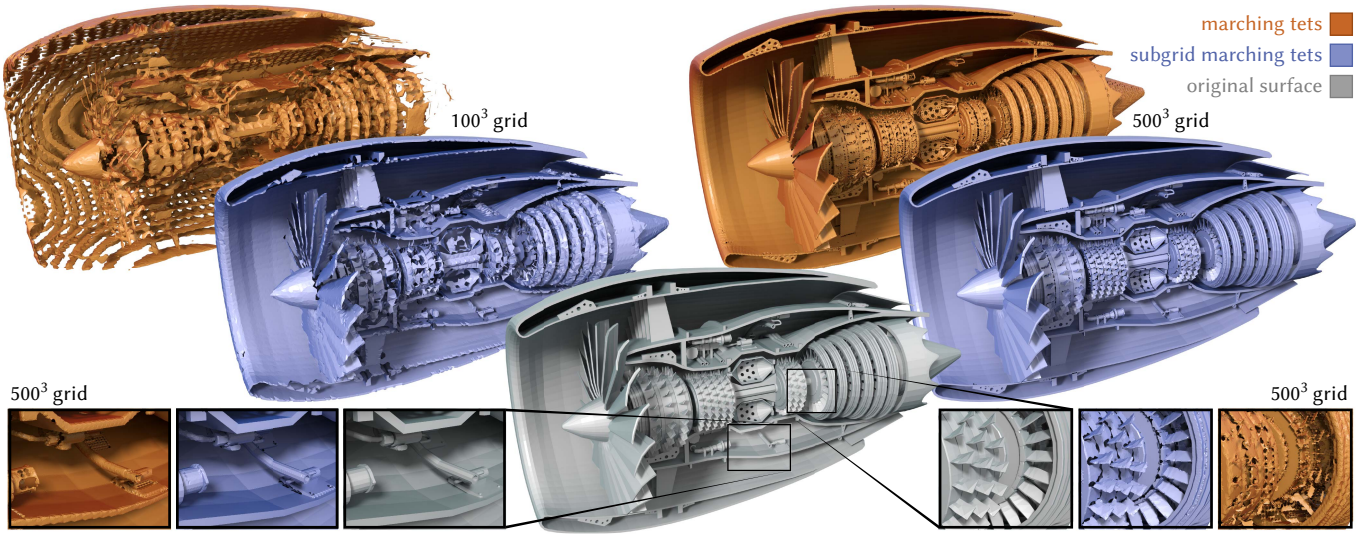


Fig. 1. Just as dual contouring extends classic marching algorithms to capture sharp geometric features, our *subgrid* extension enables marching algorithms to capture fine topological features, below the grid resolution. Here we reconstruct a jet engine (gray) using standard marching tetrahedra (orange) 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.

Contouring algorithms like marching cubes and marching tetrahedra are a fundamental tool in geometry processing, needed to convert from implicit representations (like signed distance functions) to explicit representations (namely, polygon meshes). However, they suffer from an equally fundamental challenge of signal processing, namely *aliasing*: unless the implicit function is sufficiently-well sampled onto a grid, the extracted mesh will have missing features, holes, etc. Moreover, these methods assume that the surface has a definite inside and outside—making them unsuitable for surfaces with boundary. The conventional solution is simply to increase grid resolution, but due to Nyquist-Shannon, this resolution may have to be made prohibitively large in order to capture very fine features in an implicit function; moreover, one most often does not even have an a priori bound on the smallest feature size. A common workaround is to use spatially adaptive marching methods, but here one encounters significant complexity in data structures and implementation—and still suffers from the lack of an a priori bound. We adopt a fundamentally different approach: rather than adapt the grid to the feature size, we change the information stored on the grid to capture (in spirit) information about the *frequency* of geometric features within each cell. More precisely, we encode geometry using a generalization of *Haken's normal coordinates* from geometric topology, which count the number of surface intersections for each edge, plus the intersection locations (as 1D barycentric coordinates). Our key contribution, then, is a procedure for reconstructing manifold, intersection-free geometry from these generalized normal coordinates. Unlike other marching procedures, we do not require a distinct inside/outside—nor do we even require consistently-oriented surface patches. This procedure is a strict generalization of the one used for marching tetrahedra, and shares all of its attractive properties. E.g., reconstruction can be performed in parallel across all cells, while still guaranteeing compatibility with neighbors. Moreover, unlike classic marching algorithms, we do not need to build a large table of possibilities, but can instead just run a simple deterministic algorithm. We evaluate our method on two applications: contouring signed distance functions with thin features, and converting polygon soup to manifold, intersection-free meshes. In practice, we see that, for equal grid resolution, the surface fidelity achieved via contouring is dramatically closer to the true surface. More importantly, with our method there is no lower bound on the feature size that we can encode on a fixed grid (apart from floating precision), meaning that we do better even than an adaptive method, for any fixed maximal degree of refinement.

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

Authors' addresses: Hossein Baktash, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213; Mark Gillespie, Inria, Palaiseau, France, University of Utah, Salt Lake City, USA; Keenan Crane, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, Roblox, San Mateo, CA, 94403.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© XXXX Association for Computing Machinery.
0730-0301/XXXX/-1-ARTXX $15.00
https://doi.org/XX

ACM Reference Format:
Hossein Baktash, Mark Gillespie, and Keenan Crane. XXXX. Subgrid Marching Tetrahedra. *ACM Trans. Graph.* X, X, Article XX (XXXX), 19 pages. https://doi.org/XX

1 INTRODUCTION AND RELATED WORK

Isosurface contouring is the three-dimensional analog of root finding: given a continuous function $f : \mathbb{R}^3 \to \mathbb{R}$, such as a *signed distance function (SDF)*, we seek a polygonal approximation of its zero level set

$$S \coloneqq \{x \in \mathbb{R}^3 \mid f(x) = 0\}.$$

This operation is fundamental to geometry processing, enabling conversion from implicit to explicit surface representations.

Marching algorithms, such as marching cubes [Lorensen and Cline 1987] and marching tetrahedra [Doi and Koide 1991] take a "divide and conquer" approach to isosurface contouring, by dividing space into cells and building an approximation within each cell. Since these local approximations are (by construction) compatible along cell boundaries, they globally form a single coherent mesh. Marching algorithms are widely used in practice, since they can be efficiently implemented in parallel, and are guaranteed to produce a closed manifold mesh.

However, existing marching algorithms (including those using adaptive grids [Shu et al. 1995]) exhibit a fundamental limitation: the output frequency is bounded by the resolution of the grid cells. Hence, standard marching effectively applies a low-pass filter to the implicit surface—resulting in aliasing of fine geometric features, thin sheets, *etc.*. The basic reason is that these algorithms generate at most one output vertex per grid edge. In classic marching cubes, for instance, one linearly interpolates the value of f at edge endpoints, and finds where this linear interpolant passes through zero. If the true function f has multiple zeros, this strategy cannot possibly yield the correct output, and may skip the edge altogether.

Our *subgrid* extension to the marching paradigm addresses this limitation head-on, by permitting multiple surface intersection points per grid edge. Here, there are two main changes to the standard approach:

- First, 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 discussed in Section 4.1.2, this task can be carried out efficiently and accurately for many common implicit surface types (e.g., sphere tracing for SDFs; interval analysis for neural implicits). More generally, this setup reduces 3-dimensional surface contouring to the more well-studied problem of ordinary 1-dimensional root finding. Moreover, because of the robustness of our reconstruction procedure, *it is fine in practice to simply take uniform samples along the edge*, reducing the requirements on our function evaluation to those of ordinary marching cubes/tets.
- Second, rather than rely on a finite lookup table of output configurations, we develop a closed-form deterministic algorithm that reconstructs a local polygonal approximation given *any* number of intersections along the edges of a tetrahedron (Figure 2). This algorithm takes time at most $O(n)$ in the number of edge intersections n. Like traditional marching algorithms, these local reconstructions are automatically compatible with those from neighboring tets, yielding a globally manifold mesh. Hence, we can still perform isosurfacing independently and in parallel across all cells of the grid. The existence of such a reconstruction is not obvious *a priori*; developing our reconstruction algorithm occupies the bulk of our exposition.

Since the only input needed by our method is a collection of input points per grid edge, the surface being contoured does not need to have a well-defined inside and outside—unlike classic marching algorithms, which are based on detecting a sign change. Even for our dual reconstruction algorithm, which uses normals to improve reconstruction quality, we need only *unoriented normals* (*i.e.*, sign does not matter). Hence, beyond standard "inside-outside" isosurfacing, we can also apply our method to broader geometry processing tasks like mesh repair—*e.g.*, converting inconsistently-oriented polygon soup into a manifold oriented mesh. Such conversion is widely sought after in modern mesh processing and learning pipelines [Wu et al. 2025].

1.1 Normal Surface Theory

The starting point for our approach is the *normal surface theory* initiated by Kneser [1929] and Haken [1961], which in mathematics is a central tool in the algorithmic study of 3-manifolds [Matveev 2007]. The basic idea is that a surface can be encoded via the way it intersects a tetrahedral grid (Section 2.1). By assumption, surfaces must intersect the grid in a canonical or *normal* way, in order to keep this encoding simple. Namely, within each tetrahedron, each connected component of the surface must resemble the standard cases from the marching tets algorithm: four triangular "corner cuts," and three quadrilateral "diagonal cuts," corresponding to the seven generic ways a plane can intersect a tet. A surface is hence reduced to a vector $\mathbf{x} \in \mathbb{Z}^{7n}_{\geq 0}$ of integer *normal coordinates* counting the number of intersections of each type, for each of n tets in the grid. (E.g., for standard marching tets, entries of $\mathbf{x}$ are only zero or one.) Importantly, this encoding does not pin down the precise geometry of the surface—rather, it identifies only an *isotopy class* of normal surfaces that exhibit the same coordinates.

In mathematics, this 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 any superfluous geometric details. Informally, any unnecessary "wrinkles" have already been "pulled tight." For practical isosurfacing, however, we wish to represent as much detail as possible, including all the wrinkles. In this work we hence make a break with the usual philosophy of normal surface theory and instead adopt *edge coordinates* as our primary representation. Edge coordinates simply count how many times each edge of the grid pierces the surface; the only restriction is that intersections must occur at isolated points (*e.g.*, an edge can not lay flat along the surface). Traditionally, edge coordinates are typically viewed as auxiliary data derived from the normal coordinates: each type of surface patch increments the count on several edges. Unlike normal coordinates, edge coordinates do not restrict the way a surface can intersect the grid to a finite

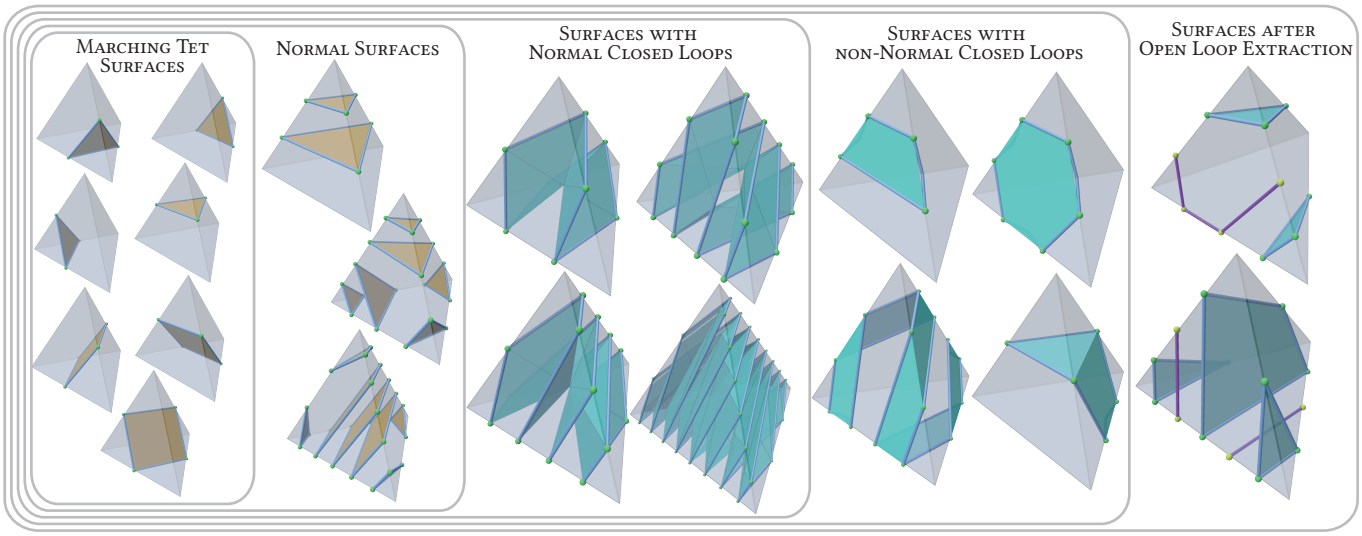


Fig. 2. Some examples of surfaces that we generate inside a single tetrahedron, organized by increasing complexity from the simplest versions (limited to one disc per tetrahedron) that are used classically in marching tetrahedra, to more general surfaces that we construct in this paper, which may have arbitrarily many components or boundary segments.

number of types. However, as we show in this paper, one can still uniquely reconstruct the surface within each tetrahedron (up to a small set of symmetries), via a simple deterministic algorithm.

To date, normal surface theory is little used in geometry processing and visualization. The marching tetrahedra algorithm [Doi and Koide 1991] reinvented a small piece of this story, 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

For isosurfacing, it is essential that we not only determine the mesh topology (*e.g.*, the isotopy class), but that we also recover a specific geometry approximating the original surface. For this reason, we enrich the integer intersection counts (*i.e.*, edge coordinates) with locations and (optionally) normals for each intersection point. From this point data, we must then produce a high-quality interpolating surface. Our reconstruction algorithms (Section 3) proceed inductively on dimension: we start with 0-dimensional intersection points along edges (Section 4.1.2), then construct a collection of 1-dimensional normal curves on the boundary of each tetrahedron (Section 3.1), and finally fill these tetrahedra with 2-dimensional polygons interpolating the boundary curve (Section 3.2 and Section 3.3). This final step effectively builds a discrete spanning surface

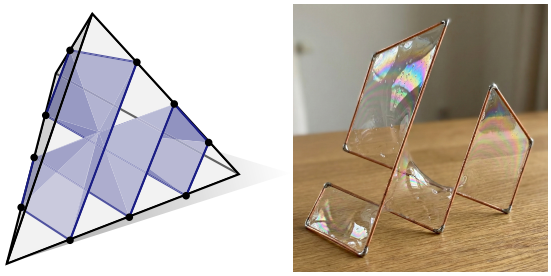


Fig. 3. The core of our method is an algorithm for filling a curve on a tet boundary with an interpolating surface—analogous to a minimal-area soap bubble formed by a closed loop of wire.

or *Seifert surface*, *i.e.*, an embedded, orientable surface with given boundary.

The problem of computing spanning surfaces is reasonably well-explored in visualization and geometry processing [Van Wijk and Cohen 2006]. A common approach is to cast it as an instance of *Plateau's problem*: find the surface of minimal area with prescribed boundary—often realized physically via a soap film attached to a wire boundary (Figure 3). Several methods minimize a mesh-based energy [Renka and Neuberger 1995; Pinkall and Polthier 1993], which requires a fine surface triangulation. However, for surfaces with multiple components this approach does nothing to prevent self-intersections—in this Lagrangian setting, one would require a more expensive collision potential or repulsive energy [Yu et al. 2021]. More recent methods have explored a Eulerian representation using a fine regular grid [Wang and Chern 2021] or neural implicit [Palmer et al. 2022], where surfaces are intersection-free 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, we develop discrete algorithm that produces a triangulated disk with the minimal number of elements needed to span the given curve. Moreover, rather than minimize area, we aim for the best geometric reconstruction, *e.g.*, by incorporating information about surface normals (Section 3.5).

The "soap bubble" picture also provides some intuition for how marching algorithms, including ours, can easily guarantee manifold output: as long as the geometry constructed for each grid cell is manifold, restricted to the cell, and shares its boundary with the geometry from neighboring cells, we are simply gluing together manifold pieces. See Appendix A for a more rigorous discussion.

1.3 Isosurfacing

Relative to classic marching algorithms, which need only perform point evaluation of an implicit function, an apparent drawback of our method—at least at first glance—is the need to find all intersection points along each edge. This shift from 0-dimensional sampling to 1-dimensional root finding is a fundamental trade off in the design of contouring algorithms, which offers several clear benefits (e.g., better approximation of fine features, and the ability to contour surfaces that are not closed). Importantly, however, it does not change the basic requirements of a system that performs contouring, because *root finding can always be approximated via point sampling*, *e.g.*, by uniformly sampling points along an edge (effectively performing classic marching in 1D rather than 3D). Although a sampling-based approach can miss intersections, (i) the robustness of our reconstruction procedure means that we still always get a manifold, intersection-free mesh, and moreover (ii) we are in no worse shape than with classical marching algorithms, which can also miss intersections. In this sense, subgrid marching tetrahedra is a true drop-in replacement for existing marching algorithms. Conversely, since our approach does not fundamentally change the overall structure of standard marching algorithms, it should be possible in the future to incorporate any recent extension to classical marching—such as differentiable optimization of node geometry [Shen et al. 2021] (though we do not pursue such extensions here).

More broadly, there are a wide variety of methods for isosurfacing, such as those based on Delaunay triangulation [Gelas et al. 2009; Binninger et al. 2025], particle simulation [Witkin and Heckbert 1994], or active contours [Cohen and Cohen 2002]—De Araújo et al. [2015] provide a survey. Our subgrid approach stays firmly in the domain of fixed-grid marching algorithms, maintaining all the same benefits: easy parallel implementation via regular arrays, manifold connectivity, intersection-free geometry (in the primal case), and so on. Simultaneously, we overcome one major drawback of classic marching algorithms: the ability to resolve features below the scale of the grid—hence the name "subgrid." Another approach to resolving fine features is to use a spatially adaptive marching algorithm [Schaefer et al. 2007; Shen et al. 2023; Shu et al. 1995], but such methods are notoriously difficult to implement without cracks between levels of the hierarchy, and break the fixed, regular memory layout often needed for efficient parallel or GPU execution. In any case, our subgrid strategy should be viewed as *complementary* to, rather than competitive with, adaptive methods, since it can be used without modification on a spatially adaptive tet mesh (e.g., obtained via Delaunay refinement). In essence, subgrid marching helps to faithfully recover topology (e.g., extremely thin sheets), whereas adaptive grid refinement helps to improve geometric detail (e.g., bumps and wrinkles on those sheets). On the whole, marching algorithms remain a vital part of the isosurfacing landscape—especially due to their recent resurgence in the context of geometric learning, differential rendering and field-based 3D generative models [Shen et al. 2021, 2023; Wu et al. 2025].

2 BACKGROUND

We first establish some elementary pieces of standard normal surface theory, needed to describe our more general algorithm for reconstructing a surface from arbitrary edge coordinates. In particular, the triangles and quads from standard normal coordinates (Section 2.1) and octagons from the *almost normal* theory (Section 2.2) serve as "base cases" for our more general reconstruction algorithms (Section 3).

2.1 Normal Coordinates

2.1.1 Surface Normal Coordinates. Within a tetrahedron, a piecewise linear normal surface can have seven distinct types of polygons, corresponding to the seven (nonempty) cases in the classic marching tetrahedra algorithm. In particular, we can separate the tetrahedron vertices into sets of size 1 and 3, yielding four triangular *corner cuts*, or into sets of size 2 and 2, yielding three quadrilateral *diagonal cuts*. 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 *surface normal coordinates* within a tet are then given by the vector

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

Importantly, in order for these 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.

2.1.2 Edge Coordinates. The *edge coordinates* e_{ij} give the number of intersections between each edge ij of a tetrahedron, and the normal polygons within that tetrahedron. We denote the vector of edge coordinates by

$$\mathbf{e} \coloneqq (e_{01}, e_{02}, e_{03}, e_{23}, e_{13}, e_{12}).$$

(By convention, we offset pairs of opposite edges by three entires.) Given a set of normal coordinates, we can easily determine the number of intersections along each edge of the tetrahedron. E.g., for each copy of the triangle at corner 0, we increment the edge coordinates e_{01}, e_{02}, and e_{03} (where j, k, l are the complementary vertices), or equivalently, we can add the vector

$$\mathbf{v}_0 \coloneqq (1, 1, 1, 0, 0, 0).$$

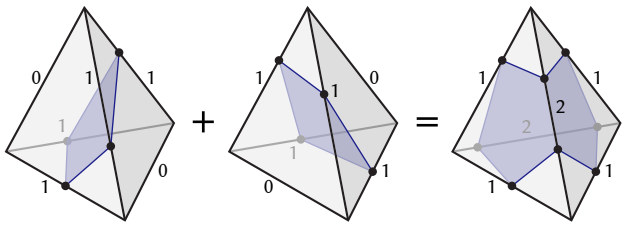


Fig. 4. The theory of *almost normal* surfaces slightly expands the set of edge coordinates that can be interpreted as intersection-free surfaces. In particular, two quads that would intersect in the standard theory *(left)* become a single octagon free of intersections *(right)*. We broaden this interpretation much further, providing a way to recover non-intersecting geometry from *any* set of edge coordinates.

In general, we can write this relationship as

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

where the matrix $M \in \mathbb{Z}^{6\times 7}$ is the *incidence matrix*, and the columns are annotated with the names of the vectors encoding each basic polygon type. We use this linear system for our proofs in Section B.1.

2.2 Almost Normal Coordinates

2.2.1 Not all edge coordinates define normal surfaces. For a normal surface, we can reconstruct polygons from edge coordinates $\mathbf{e}$ by simply solving Equation 1. Since at most one of the coordinates q_{ij} is nonzero, we effectively get a matrix with at most five linearly independent columns, and the solution is unique. In general, however, a given vector of edge coordinates may not describe any valid, intersection-free surface via standard normal coordinates. For instance, consider a set of edge coordinates $\mathbf{e} = (2, 1, 1, 2, 1, 1)$ where two opposite edges have value 2, and the remaining edges have value 1 (Figure 4, *right*). The only way to decompose this vector into normal polygons is as a sum

$$\mathbf{e} = \mathbf{v}_{13} + \mathbf{v}_{14} = (1, 0, 1, 1, 0, 1) + (1, 1, 0, 1, 1, 0), \tag{2}$$

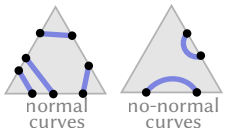
corresponding to two *intersecting* quads. Hence, if we want to reconstruct surfaces from general edge coordinates, we must broaden our approach beyond the seven basic normal polygons, and the simple linear relationship in Equation 1.

2.2.2 Almost Normal Surfaces. The theory of *almost normal surfaces* [Rubinstein 1994] introduces one additional possibility, namely that there can be one non-normal disk: either an octagon, or an annulus obtained by connecting two normal disks via a tube [Hass 2012]. The goal of this setup was not to increase geometric complexity, but rather to add just enough representational flexibility to address problems like *3-sphere recognition* [Rubinstein 1995].

At a more elementary level, however, the almost normal theory provides inspiration for how edge coordinates can be used to encode more general surfaces. In particular, the sum in Equation 2 can now be seen as giving rise to an embedded octagon, which has the same edge intersection pattern as two intersecting quads (Figure 4): two intersections on one pair of opposite edges; one intersection on all remaining edges. Our goal is to extend this kind of interpretation (much) further, to all possible edge intersection patterns.

2.3 Normal Curves

So far we have discussed coordinates that encode surfaces in space, but we can also use normal coordinates to describe curves running along a surface (see [Sharp et al. 2021, Section 3.5.1] for a more thorough introduction). In particular, a closed curve in a triangulated surface is *normal* if its intersection with any triangle is a disjoint union of simple arcs, each connecting interior points of two distinct edges. For instance, a normal curve can "cut off corners" of a triangle, but it cannot "scoop out edges" (see inset).



As in the surface case, 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. We can convert corner coordinates into edge coordinates by simply adding up the number of segments crossing each endpoint:

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

Unlike the surface case, this linear map is always invertible:

$$\begin{array}{rcl} c_0 & = & \frac{e_{01}+e_{20}-e_{12}}{2}, \\ c_1 & = & \frac{e_{12}+e_{01}-e_{20}}{2}, \\ c_2 & = & \frac{e_{20}+e_{12}-e_{01}}{2}. \end{array} \tag{3}$$

However, for non-normal curves, the resulting corner coordinates be negative: in order to describe a valid collection of normal curves, the edge coordinates must satisfy two conditions:

(1) **Even sum.** $e_{01} + e_{12} + e_{20} \equiv 0 \mod 2$
(2) **Triangle inequality.** $e_{ij} + e_{jk} \geq e_{ki}$ 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 we cannot have more arcs "leaving" the triangle from one edge than the total number of arcs "entering" the other two edges (though note that curves need not be oriented).

In our case, we will largely consider curves running along the boundary of a tetrahedron. Unlike the standard theory, we will be interested in the *decomposition* of an arbitrary curve into its normal and non-normal part—rather than restricting ourselves strictly to normal curves (Section 3.1).

3 RECONSTRUCTION ALGORITHMS

Our main *reconstruction algorithm* takes the intersection points (and possibly normals) along each edge as input, and produces a piecewise linear approximation of the surface as output. We describe two different variants, corresponding to the primal/dual variants developed for past marching algorithms:

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

For both of these algorithms, the first step is to construct a curve on the tetrahedron boundary, interpolating the intersection points (Section 3.1).

Importantly, unlike classical normal surface theory (as well as the almost normal theory), we put no conditions on edge coordinates, apart from nonnegativity. As a result, within a given tetrahedron there may be arbitrarily many non-normal spanning disks, i.e., multiple disks that are neither triangles nor quadrilaterals (nor even octagons, as in the almost normal case). Moreover, these disks can have arbitrarily many sides, corresponding to saddles of arbitrarily high index.

More precisely, our algorithms have the following pre- and post-conditions:

Input. The input is triangulation $\mathcal{T} = (V, E, F, T)$ of a solid region in $\mathbb{R}^3$, where V, E, F and T denote the set of vertices, edges, triangular faces, and tetrahedra, along with a nonnegative integer $e_{ij} \in \mathbb{Z}_{\geq 0}$ giving the number of intersections along each edge $ij \in E$, and 1D barycentric coordinates $s_1, \ldots, s_{e_{ij}} \in (0, 1)$ giving the locations of these intersection points, relative to the canonical orientation from i to j where $i < j$. In the case of the dual reconstruction algorithm, the input also includes unit normals $N_1, \ldots, N_{e_{ij}} \in S^2 \subset \mathbb{R}^3$ at each intersection point.

Output. The output is a triangle mesh, with vertex positions in $\mathbb{R}^3$. This mesh is guaranteed to be manifold, possibly with boundary. The output of the primal algorithm will be free of self-intersections (Theorem D.1). The output of the dual algorithm can exhibit self-intersections, but will still have manifold connectivity. The output will be closed and orientable if the input edge coordinates (i) are zero for all edges contained in the boundary of $\mathcal{T}$, and (ii) they satisfy the even sum condition on all triangles. In particular, these conditions will be satisfied when the edge coordinates come from a closed surface $S \subset \mathbb{R}^3$ strictly contained in $\mathcal{T}$.

Proofs of these properties are given in the appendix. For clarity and brevity, this section aims only to *state* our reconstruction algorithms, without immediately justifying each claim along the way.

3.1 Boundary Curve Reconstruction

The first step in our procedure is to connect up the edge intersections into a collection of disjoint curves $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$ on the tetrahedron boundary. Our algorithm is summarized in Figure 5. We are careful to define this procedure in a canonical way, so that we obtain the same curve on the intersection of any two tetrahedra sharing a triangle. The output of this procedure is a collection of closed loops and open curves. In the primal reconstruction algorithm (Section 3.2) the closed loops become disks interior to the tetrahedron. The open curves are simply discarded—but may later appear as boundaries of disks in adjacent tetrahedra.

Our procedure for curve reconstruction can be performed independently on each tetrahedron in $\mathcal{T}$, and from there, independently

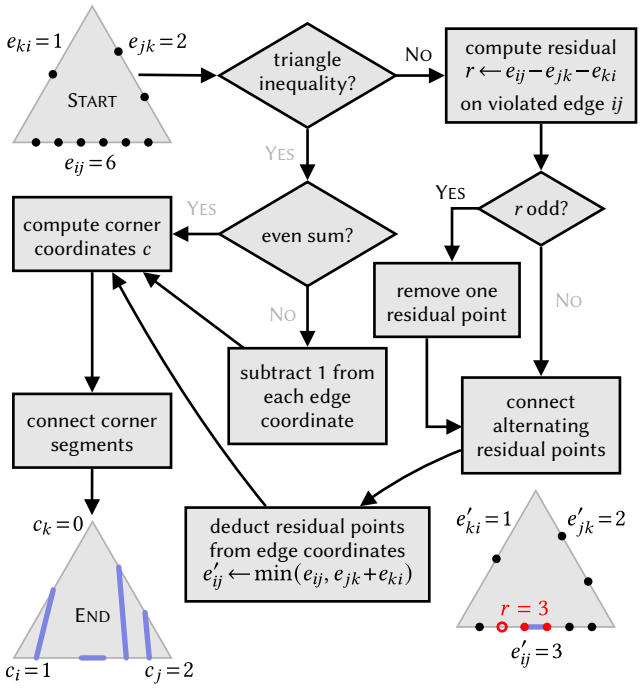


Fig. 5. Procedure for reconstructing a curve (blue) that interpolates an arbitrary set of edge intersection points (black) on each face of a tetrahedron. These curves are later "filled in" to obtain the polygons within the tetrahedron.

for each triangular face of the tetrahedron boundary. In particular, given three edge coordinates e_{01}, e_{12}, e_{20}, we do the following:

(1) If the edge coordinates satisfy the even sum and triangle inequality conditions (Section 2.3), we compute the corner coordinates via Equation 3. For each corner i we then connect up the first c_i pairs of intersection points along oriented edges ij and jk 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 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 the residual points:
   (a) If r is even, we connect consecutive pairs of residual points into segments. These segments correspond to the "scoops" discussed in Section 2.3, where we effectively pick the configuration with the 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.

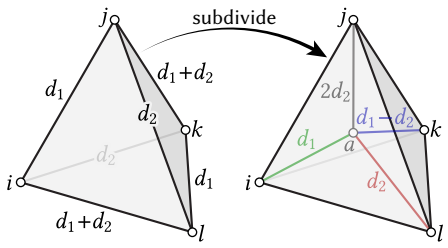


Fig. 6. We reduce the general case of normal loops to a few simple base cases via subdivision of the edge coordinates, via the stencil shown above.

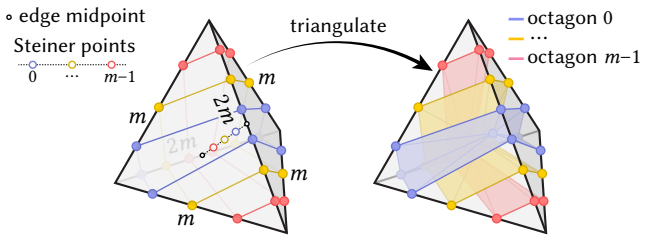


Fig. 7. To triangulate a collection of octagonal loops, we place m evenly-spaced Steiner points along the segment between midpoints of the two edges with $2m$ intersections.

Segments constructed individual triangles share endpoints, forming longer, piecewise linear curves $\gamma_1, \ldots, \gamma_m$ on the tetrahedron boundary. We classify these curves as follows:

- (Open curves.) If γ has a degree-1 vertex, it forms an open curve. Such curves are discarded, though segments from these curves may still appear as boundaries of disks constructed in neighboring tetrahedra, contributing to the boundary of the output mesh.
- (Normal curves.) If all segments of γ connect two distinct edges, then γ is a normal curve.
- (Non-normal curves.) Otherwise, if some segment of γ runs along an edge of the tetrahedron, it is a *non-normal* curve.

This classification will be used to define our primal surface reconstruction algorithm, in Section 3.2.

Note that although some of the curves we construct look quite complicated, they are essentially the simplest curves on the tetrahedron boundary that interpolate the given *geometric* intersection points—even if *topologically* they are homotopic to simpler piecewise linear representatives. In our task, and unlike traditional normal surface theory, it is the geometry of the surface (and not just its topology) that we wish to represent.

3.2 Primal Surface Reconstruction

Suppose we now have a collection of closed loops Γ on the boundary of our tetrahedron. We must now triangulate these loops in such a way that the resulting surface is free of self-intersections. We will treat different types of loops case by case.

3.2.1 Normal Boundary Curves. We first consider the subset of Γ consisting purely of normal curves γ (as identified in Section 3.1). Recall that all curves in Γ are closed loops on the tetrahedron boundary. Our high-level strategy is to reduce the general case to one of four configurations we can handle directly: triangles, quads, octagons, or a single closed loop.

The first step is to construct a triangle for each curve of length $\ell = 3$, and remove these curves γ from our set. After processing the triangles, we can establish the following two properties:

(1) All remaining loops have the same length $\ell > 3$ (Theorem B.6).
(2) These loops have edge coordinates d_1, d_2, and $d_1 + d_2$ on opposite pairs of edges, for some pair of integers $0 \le d_2 \le d_1$ (Theorem B.3)

In other words, the remaining curves can be viewed as a sum of normal quadrilaterals of at most two distinct types. But when $d_1 \neq 0$, we must give a different interpretation to these edge coordinates in order to recover a nonintersecting surface.

In particular, we take the following approach:

- If $\ell = 4$, *i.e.*, if the remaining loops are quads, then we split all quads along the same diagonal, producing two triangles per loop. The choice of diagonal is arbitrary (though see Section 5 for further discussion of split choices).
- If $\ell > 4$, then there are two possibilities to consider:
  (1) 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.
  (2) If we have $m > 1$ loops, then the shortest they can possibly be is length $\ell = 8$ (octagons), since the length of a normal loop on a tetrahedron can only be a multiple of 4, except for triangles (Theorem B.6). In this case, some pair of opposite edges e, e' will have $2m$ intersections, and the remaining edges will each have m intersections. To triangulate the octagons, we place m Steiner points $x_0, \ldots, x_{m-1}$ uniformly along the oriented segment from e's midpoint to e''s midpoint. Pairs of intersections on edge e are then connected to consecutive Steiner points, from the innermost to outermost pair (Figure 7). More explicitly, if we enumerate the intersections along edge 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.
  (3) Otherwise, noting property (2) above, let i, j, k, l be labels for the vertices of the tetrahedron such that $e_{ij} = d_1$, $e_{ik} = d_2$, and $e_{il} = d_1 + d_2$ (with equal values on the opposite edges). We subdivide the tetrahedron into four smaller tets, by connecting its vertices to the barycenter a, and assigning edge coordinates

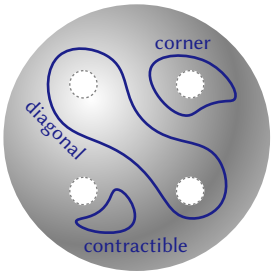
$$e_{ai} = 2d_2,\ e_{aj} = d_1,\ e_{ak} = d_2,\ e_{al} = d_1 - d_2, \tag{4}$$

  as shown in Figure 6. We then recursively process each of the four new tets.

Importantly, this subdivision process is finite: the number of subdivisions is bounded from above by the sum of the original edge coordinates (Theorem C.1), though this bound is not tight, and in practice we require far fewer subdivisions. The edge coordinates

assigned during the subdivision step effectively describe the simplest normal curve that agrees with the observed intersection points. More precisely, they are the smallest edge coordinates that satisfy the curve normality conditions on all inserted triangles. Note that due to symmetries of the edge coordinates, the assignment of vertex labels i, j, k, l is not unique, but any such assignment is sufficient to provide all the properties we need. (To get a canonical definition, one can use lexicographic ordering to break ties.) Finally, note that we do not need to maintain an explicit tetrahedral mesh data structure during the subdivision process—we simply perform ordinary recursion (e.g., on the call stack, or using a queue local to the original tetrahedron).

3.2.2 Non-Normal Boundary Curves. Finally, consider non-normal curves γ (as defined in Section 3.1). Recall that these curves are all simple closed loops on the boundary of the tetrahedron.



Loop type. If we view a tetrahedron as a topological sphere with punctures at the four vertices, then there are three types of simple loops (no matter how much the curve "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.

To determine the loop type, we simply determine the parity of each edge (similar to standard marching tets). *E.g.*, starting at vertex 0, we can count the number of intersections along the edges connecting it to vertices 1, 2, and 3. If this count is odd, then the two vertices are separated by the curve; otherwise, they belong to the same connected component of the tet boundary. More explicitly, let $p := \mathrm{mod}\ (e_{01}, 2) + \mathrm{mod}\ (e_{02}, 2) + \mathrm{mod}\ (e_{03}, 2)$. Then γ is contractible if $p = 0$, is of diagonal type if $p = 2$, and is of corner type if $p = 1$ or $p = 3$.

Vertex label. A simple loop partitions the tet boundary into two pieces; to construct some of our spanning disks, we will need to know which piece is "inside" versus "outside" the loop. 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 need not make an inside/outside distinction.

Spanning disks. Finally, we construct a triangulated disk bounded by γ. The strategy depends on the loop type:

- **Diagonal.** If γ is of diagonal type, we insert a Steiner point a at the center of mass of γ, and construct a triangle from a and each edge of γ.
- **Corner and contractible.** Otherwise, we build a piecewise linear disk contained mostly in the boundary of the tetrahedron itself. In particular, we split each bounding triangle f along the segments of γ, yielding a collection of planar polygons $f \setminus \gamma$. Along each edge of f, we determine the parity of the resulting segments by starting at a vertex, and alternating between inside and outside each time we encounter an intersection point. 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, and instead emit a triangle at the corner.

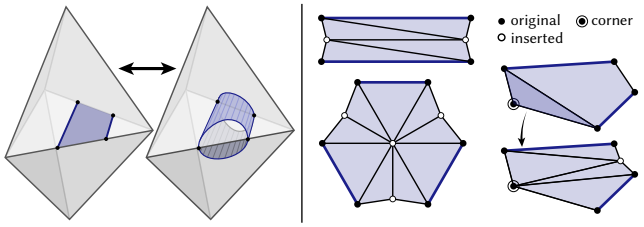


Fig. 8. *Left:* Pairs of polygons generated on tetrahedron boundaries form topologically manifold components, but the piecewise linear geometry is not yet intersection free. *Right:* we obtain a piecewise linear embedding by inserting a few Steiner points on each polygon, and pushing them slightly inside each tet.

3.2.3 Simplicial Embedding. At this point, we have formally constructed a mesh with manifold connectivity, but which may be an immersed Δ-complex (in the sense of Hatcher [2002, Section 2.1]) rather than an embedded simplicial complex. In particular, whenever a non-normal loop γ passes through a face f of the tetrahedral grid, we will get two oppositely-oriented copies of the polygons bound by γ on the two tetrahedra sharing f. Topologically, these two copies make a perfectly valid manifold tube or punctured sphere (Figure 8, *left*), but for downstream applications we typically want to convert such regions to a standard, intersection-free triangle mesh.

Since a single loop γ can have at most three segments running along the edges of a face f, there are three basic polygon types to consider: a quad, a hexagon, and a pentagon made at a corner. We first insert the midpoint of any segment contained in an edge of f, pushing these midpoints slightly into the tetrahedron. For hexagons, we also insert the center of mass, pushing it in the same distance. From here, we can easily triangulate the region so that it stays within the tetrahedron (Figure 8, *right*).

Note that this procedure is needed only when taking the union of the two oppositely-oriented polygons would yield a nonmanifold edge (i.e., an edge contained in three or more triangles of the output mesh). Likewise, if manifold output is not required, one can simply keep one of the two polygons.

3.3 Dual Surface Reconstruction

Restricting vertices of the reconstructed mesh to grid edges makes it difficult to capture sharp features, which rarely line up with the grid. Inspired by *dual contouring* [Ju et al. 2002] and *dual marching* algorithms [Schaefer and Warren 2004], we develop a dual version of our subgrid method that yields more accurate reconstruction at the same grid resolution. This version of the algorithm is also conceptually simpler, and simpler to implement, than the primal version, because we no longer need to carefully attend to the intersection-free property.

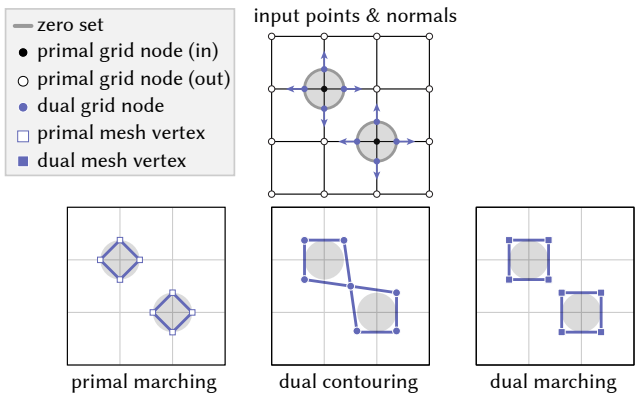


Fig. 9. Dual contouring produces a single vertex per grid cell, whereas dual marching yields a vertex for each component of the primal mesh—here shown in the context of 2D *marching squares*. Since our subgrid meshes may have many components per cell, dual marching is a more natural fit.

The basic idea, as in other dualized contouring algorithms, is to use *Hermite data*, *i.e.*, both positions and normals, to more accurately predict vertex coordinates. Dual contouring associates these coordinates with dual vertices of the (volumetric) background grid, whereas dual marching considers the usual connectivity of the primal marching algorithm, then computes vertex coordinates for each connected component of the primal (surface) mesh (Figure 9). For classic marching tets (rather than marching cubes) these two strategies coincide, since there can be at most one polygon per tet. In the context of our subgrid method, where we can have many polygons per tet, the right choice is to apply dual *marching*, rather than dual contouring, in order to preserve topologically distinct thin features.

The main trade off is that we can no longer guarantee that the output mesh is intersection free: unlike our primal algorithm, where we carefully place Steiner points to avoid-self intersection, the vertex coordinates predicted by dual marching can be fairly arbitrary. However, in cases where an intersection-free guarantee is needed, one could in principle simply revert intersecting cells to the primal configuration (or interpolate between the two configurations); we do not pursue such a scheme here.

Our dual reconstruction algorithm proceeds in two steps: first construct the dual surface connectivity (Section 3.4), then compute vertex coordinates for this dual mesh (Section 3.5).

3.4 Dual Connectivity

To obtain connectivity for dual reconstruction, we compute the closed boundary loops γ for each tet, exactly as described in Section 3.1 (and skipping the subsequent surface reconstruction steps in Section 3.2 and Section 3.3). These loops, together with the intersection points, already define a primal mesh M with (possibly nonplanar) polygonal faces. To get the dual connectivity M', we build the usual *Poincaré dual*, replacing each primal polygon with a dual vertex, each primal edge with a dual edge connecting the dual vertices from adjacent polygons, and each primal vertex with a dual polygon. We then triangulate the polygons of M' arbitrarily and without inserting any additional vertices (though there may be more intelligent splitting strategies here—see Section 5). Note that in order to form the dual, we cannot simply emit a polygon soup (i.e., independent polygons per tet) but must instead form a globally connected mesh. One possibility is to merge polygons based on vertex coordinates, but we found it more robust in practice to store a global reference to the triangle containing each polygon edge, and build the primal connectivity based on these grid combinatorics. The only remaining calculation, then, is where to place each vertex of the dual.

3.5 Dual Geometry

The basic geometric observation behind dual reconstruction algorithms is that points on sharp features will (nearly) sit on the tangent planes of multiple nearby points. For instance, the vertex of a cube sits at the intersection of the three adjacent faces; even a sharp crease along a smooth curve looks more and more like the intersection of two planes as we "zoom in."

Quadratic Error Function. Hence, given a collection of intersection points $x_1, \ldots, x_k \in \mathbb{R}^3$ along a loop γ and associated unit normals $n_1, \ldots, n_k \in S^2 \subset \mathbb{R}^3$, we can look for a point $p \in \mathbb{R}^3$ at the intersection of the planes P_i passing through x_i with normal n_i. In general, of course, these planes may have an empty intersection. Hence, the standard approach is to minimize the *quadratic error function (QEF)* (equivalent to the well-known *quadric error metric* of Garland and Heckbert [1997]):

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

where a term 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. A variety of corrective devices have been explored in the dual isosurfacing literature; in practice we simply minimize a regularized energy

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

i.e., we try to pull the minimizer closer to the loop centroid $\bar{x}$, using a term exhibiting the same quadratic growth as the QEF. We use the parameter $\lambda = 0.1$ for all examples in this paper; in general one could put more care into the design of this regularization procedure.

Note that the QEF is invariant with respect to sign flips on the normals n_i—hence, the input normals do not need to be consistently oriented in order to make good predictions. Also note that in our setting a primal loop will always have at least three vertices—hence, we do not need to handle the degenerate cases $k = 1$ or $k = 2$.

4 EVALUATION AND COMPARISONS

In order to investigate the benefits of our approach, we used both classical marching tetrahedra and our new subgrid marching tetrahedra, along with their dual versions, on a large set of input surfaces including both implicit SDFs and explicit polygon soup meshes. In particular, we use the SDF collection of 1-Lipschitz implicit functions assembled by Takikawa et al. [2022], and for meshes we use the

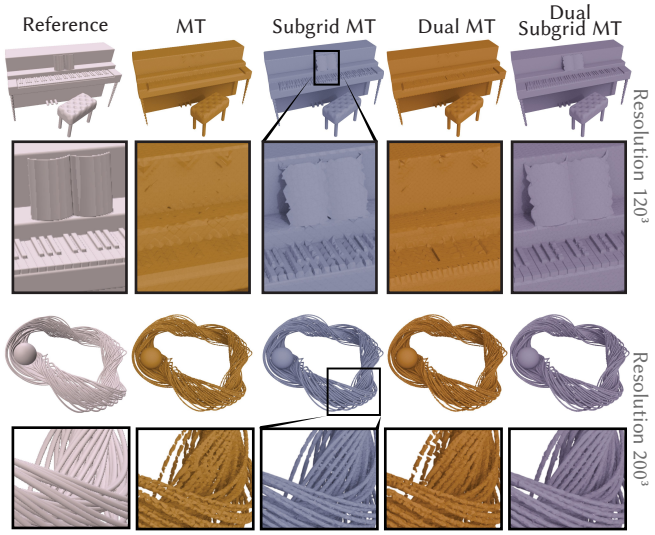


Fig. 10. *Left:* Iso-surfaces of two analytical SDFs, rendered directly via sphere marching. *Right:* Iso-surfaces extracted by classical and subgrid marching tetrahedra at grid resolutions of 120^3 and 200^3 respectively. Subgrid marching tetrahedra is able to preserve thin features like the sheet music and piano keys on the top row, or the loose cables on the bottom row, which are lost or broken by classical marching tets.

3200 models in the DORA dataset [Chen et al. 2025], which contains models taken from Objaverse [Deitke et al. 2023], ABO [Collins et al. 2022], GSO [Downs et al. 2022], and Meta [Dong et al. 2025]. We report timing and runtime of our primal and dual algorithm in Figure 11 and compare them against marching tetrahedra.

Sample results reconstructed from SDFs can be seen in Figure 10, and results reconstructed from meshes can be seen in Figures 12, 14, and 15. We find that, as expected, subgrid marching tetrahedra often captures thin features in the models which are missed by classical marching tets.

In the case of explicit triangle mesh inputs, where the ground truth surface is available, we plot the reconstruction errors and runtimes in Figure 11. In particular, we plot how the average *mean Hausdorff distance* (Section 4.1.3) and runtime change as the grid resolution increases from 16^3 to 256^3. We find that at lower resolutions subgrid marching tetrahedra achieves significantly lower error than classical marching tetrahedra and converges to very low errors in much lower resolution. As expected, both methods converge to the zero error in higher resolutions, but even at resolutions like 300^3 or 500^3 our method resolves features missed by classical marching tets (see Figures 14 and 15). The timing information in Figure 11 shows that our method runs at a speed comparable to classical marching tets.

In Appendix E, we also provide an ablation where we compare subgrid marching tetrahedra and classical marching tetrahedra to an intermediate algorithm which places vertices exactly at zeros of the implicit function located along grid edges (like our method), but is limited to emitting one face per tetrahedron (like classical marching tets). This exact zero placement improves the results slightly compared to classical marching tets, but the lack of sub-grid resolution means that it still suffers from similar artifacts—emphasizing how important it is that subgrid marching tetrahedra is able to use *all* of the intersections along an edge.

4.1 Implementation

In order to run these experiments, we implemented our method and classical marching tets in single-threaded C++ code. As with classical marching tets, subgrid marching tets can in principle run independently on each tetrahedron in the background grid, offering plenty of room for parallel speedups, but we leave such accelerations 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 also in supplemental material), we have also provided a standalone HTML/JavaScript implementation and visualizer of our reconstruction algorithm for a single tetrahedron. This routine is the critical component of our method, analogous to the table of stencils in *marching cubes*; all other steps are standard in geometry processing.

4.1.1 Tetrahedral Grid. An attractive feature of simplicial marching algorithms, including our subgrid marching scheme, is that they can be applied to any simplicial mesh—whether regular or unstructured. We choose the simplest option, to subdivide cubes of a regular voxel grid to obtain our tetrahedral grid; each cube is divided into exactly 5 tetrahedra. Whenever we refer to an N^3 resolution, N is the resolution of the voxel grid.

4.1.2 Finding Intersections. Since many common implicit surface types were developed for the purpose of ray tracing, there are already algorithms that reliably find all intersections along a given segment (Section 4.1.2). Moreover, recent techniques from interval analysis make the method applicable to neural implicit surfaces [**?**]. In general, many implicit surfaces already come with corresponding root finding algorithms:

- polynomial surfaces → Sturm sequences
- Lipschitz functions (including SDFs) → sphere tracing
- harmonic functions → Harnack tracing
- neural implicits → interval analysis

For normal information, required for the dual algorithms, we store the normals at every intersection point along an edge. For mesh inputs, this information is already given at ray-tracing time. For SDF examples, we use finite differences to obtain the normal at every intersection point.

4.1.3 Reconstruction Error. We compute the reconstruction error by measuring the mean Hausdorff distance:

$$d_{\mathrm{mH}}(A,B) = \frac{1}{2}\left(\frac{1}{|A|}\sum_{a\in A}\min_{b\in B}\|a-b\|_2 \;+\; \frac{1}{|B|}\sum_{b\in B}\min_{a\in A}\|b-a\|_2\right), \tag{5}$$

In practice, we approximate these minima numerically by sampling 10000 random points on each mesh. Figure 11 shows the mean reconstruction error, and the 10th and 90th percentile envelopes, for classical and subgrid marching tetrahedra across all 3200 meshes of the DORA dataset.

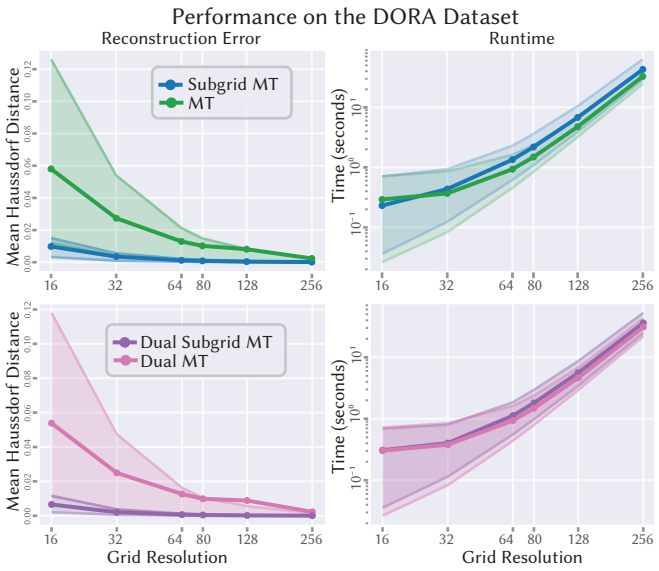


Fig. 11. Here we plot the reconstruction error and runtime cost for subgrid marching tetrahedra and classical marching tetrahedra, evaluated at different resolutions on the DORA dataset. The solid line indicates the average performance, with the 10th-90th percentile shown in the shaded envelope. Subgrid marching tetrahedra produces reconstructions with dramatically lower error at low resolutions—for almost the same computational cost.

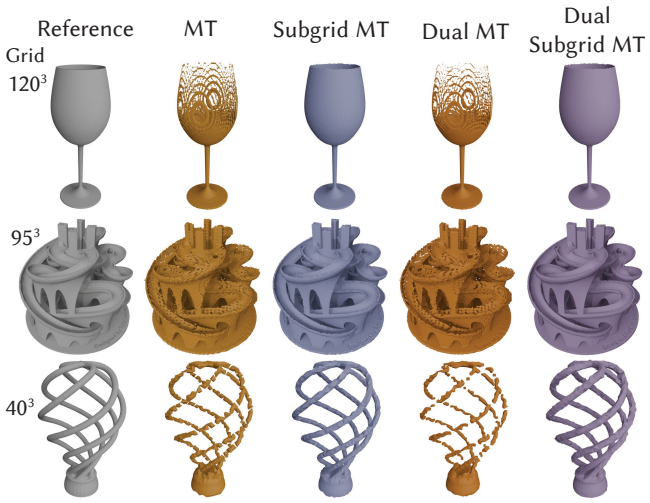


Fig. 12. Meshes obtained via our subgrid approach and marching tets, along with their dual versions. At these low resolutions, our subgrid algorithm preserves the mesh topology, while marching tets struggles to resolve topological and geometric features.

4.2 Subgrid Marching Triangles

Our procedure for reconstructing a curve within a single triangle also effectively 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

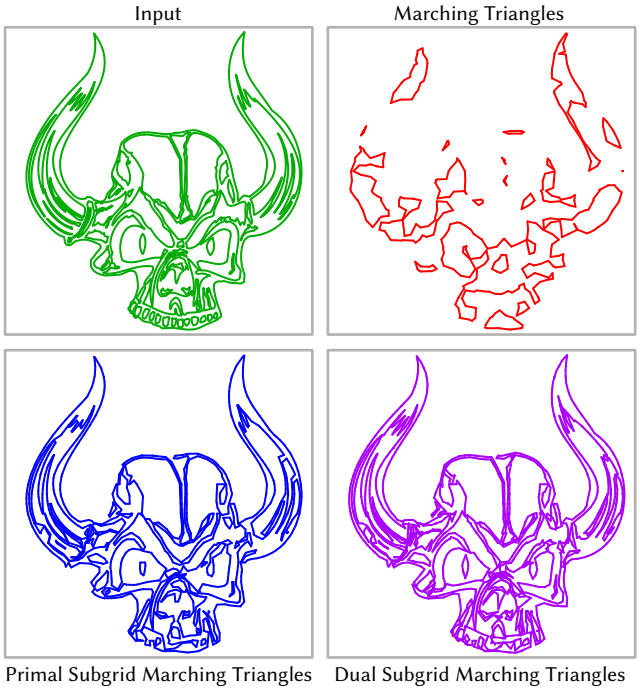


Fig. 13. Our procedure for reconstructing curve segments in individual triangles provides a 2D analogue of our algorithm to extract isolines of a 2D scalar function via subgrid marching triangles. Here we show reconstructions computed on a 32^2 grid. As in the volumetric case, we see that the subgrid accuracy allows us to resolve lots of complex geometry missed by ordinary marching triangles.

intersection counts e_{ij} on the grid edges. Here, 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 can instead subtract 1 from the largest edge coordinate, breaking ties arbitrarily. Figure 13 shows a comparison between this subgrid method and classical marching triangles. Note that, like our 3D procedure, this 2D procedure does not put any conditions on the regularity of the triangle grid, and can hence be used on unstructured triangulations—even surface triangulations.

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, if we measure the amortized cost *per output triangle*, the speed of the two algorithms is quite comparable: the main overhead is the additional logic of our local reconstruction procedure. Moreover, in cases where speed is critical—or where 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), and simply emit these precomputed stencils, rather than computing them on the fly. We did not attempt these kinds of optimizations in our implementation.

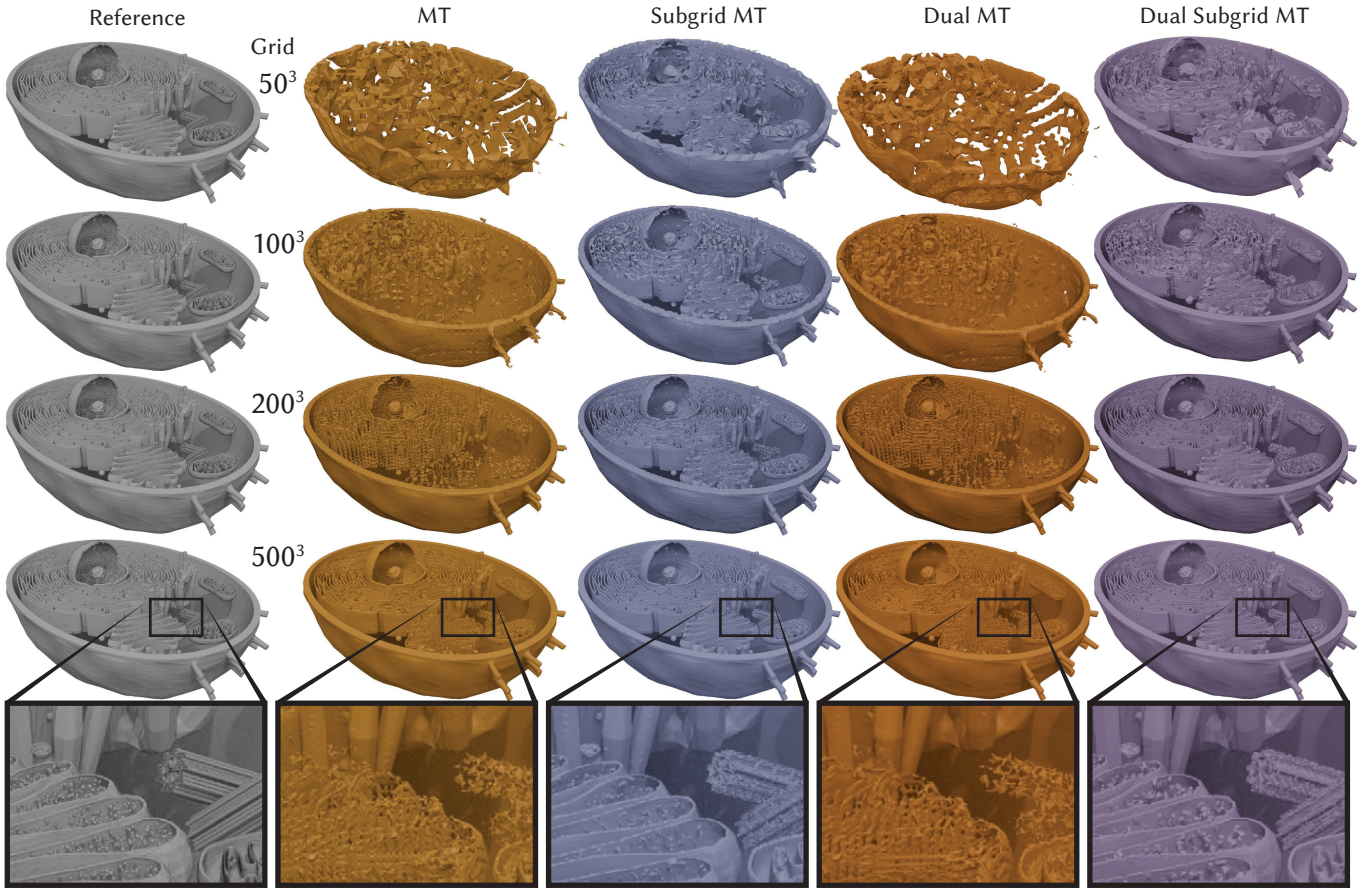


Fig. 14. In this complex cell anatomy model, even a resolution of 500^3 is not sufficient for classical marching tetrahedra to resolve all of the complex geometry.

Just as traditional marching can easily miss 2-dimensional thin sheets not sampled by any node, our subgrid approach can easily miss 1-dimensional curve-like features not sampled along any edge.

In our current implementation, we do not carefully treat the question of how to best split quads into triangles—though as noted by Shen et al. [2023] in the context of the *FlexiCubes* algorithm, additional splitting weights can be used to better capture sharp features. Such weights could in principle be incorporated into our reconstruction algorithm as well.

Although the primal mesh we construct in the primal case is formally manifold, we must introduce additional topology in order to support a simplicial embedding (Section 3.2.3). A good question for future work is whether we can instead transform the edge coordinates themselves into ones describe simpler manifold regions (e.g., those that satisfy the curve normality conditions).

REFERENCES

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.

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.

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 Computer Vision and Pattern Recognition Conference (CVPR)*. 16251–16261.

Laurent D Cohen and Isaac Cohen. 2002. 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 (2002), 1131–1147.

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. *CVPR* (2022).

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.

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*. 13142–13153.

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)*.

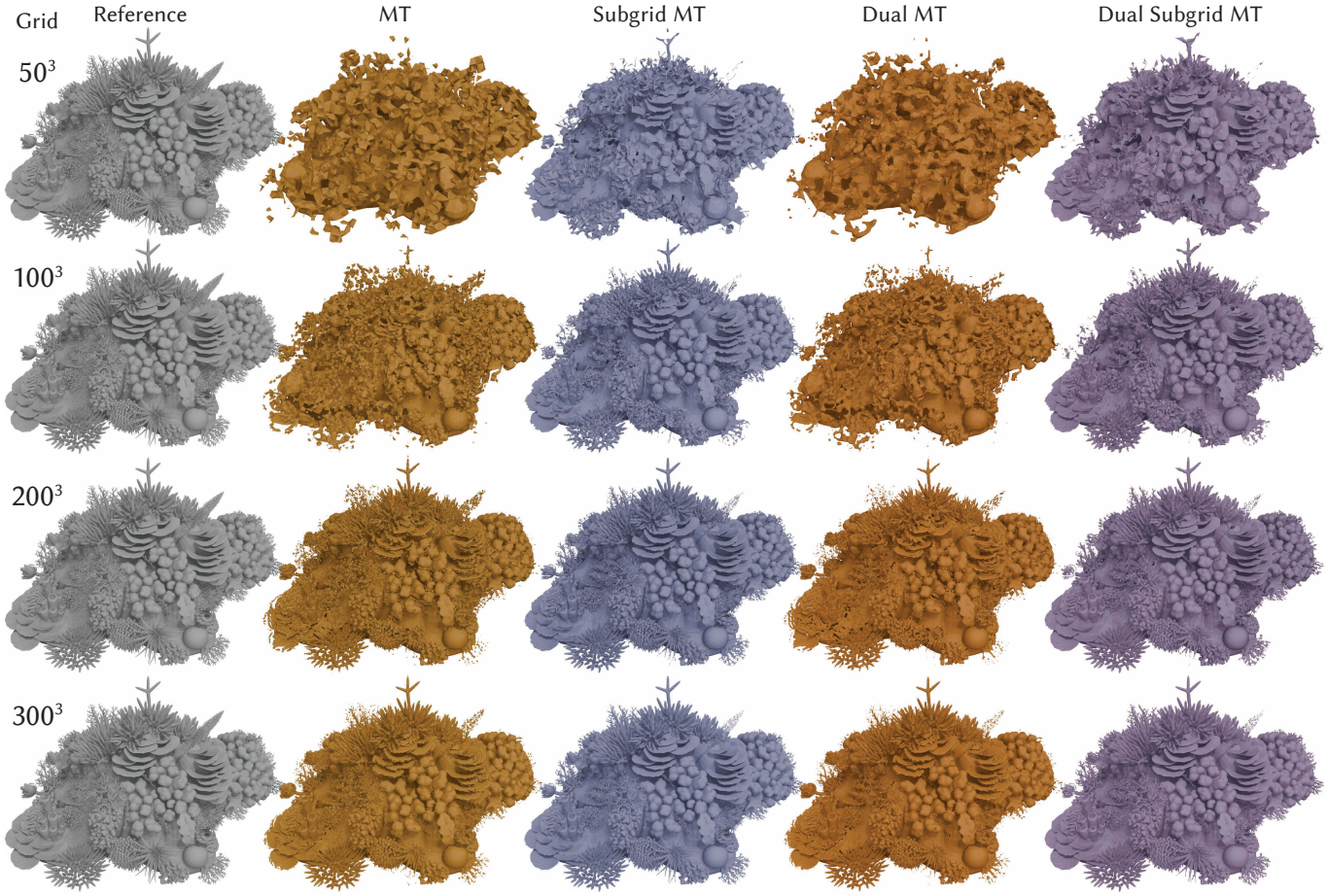


Fig. 15. When reconstructing this coral reef model, subgrid marching tetrahedra starts to resolve the detailed geometry at much coarser resolutions than classic marching tets.

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.

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.

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.

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

Mark Gillespie, Nicholas Sharp, and Keenan Crane. 2021a. Integer Coordinates for Intrinsic Geometry Processing. *ACM Trans. Graph.* 40, 6, Article 252 (2021). https://doi.org/10.1145/3478513.3480522

Mark Gillespie, Boris Springborn, and Keenan Crane. 2021b. Discrete Conformal Equivalence of Polyhedral Surfaces. *ACM Trans. Graph.* 40, 4, Article 103 (2021). https://doi.org/10.1145/3450626.3459763

Wolfgang Haken. 1961. Theorie der Normalflächen: ein Isotopiekriterium für den Kreisknoten. (1961).

Joel Hass. 2012. What is an almost normal surface. *arXiv preprint arXiv:1208.0568* (2012).

Joel Hass and Maria Trnkova. 2020. Approximating isosurfaces by guaranteed-quality triangular meshes. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 29–40.

Allen Hatcher. 2002. *Algebraic topology*. Cambridge University Press.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 339–346.

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

Marc Lackenby. 2024. Some fast algorithms for curves in surfaces. *arXiv preprint arXiv:2401.16056* (2024).

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)*. Association for Computing Machinery, New York, NY, USA, 163–169. https://doi.org/10.1145/37401.37422

Sergei Matveev. 2007. *Algorithmic topology and classification of 3-manifolds*. Springer.

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*. 18665–18675.

Ulrich Pinkall and Konrad Polthier. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics* 2, 1 (1993), 15–36.

Robert J Renka and JW Neuberger. 1995. Minimal surfaces and Sobolev gradients. *Siam journal on scientific computing* 16, 6 (1995), 1412–1427.

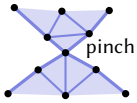
Joachim Hyam Rubinstein. 1994. *Polyhedral minimal surfaces, Heegaard splittings and decision problems for 3-dimensional manifolds*. Department of Mathematics, University Melbourne.

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.
Scott Schaefer, Tao Ju, and Joe Warren. 2007. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics* 13, 3 (2007), 610–619.
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.
Nicholas Sharp, Mark Gillespie, and Keenan Crane. 2021. Geometry Processing with Intrinsic Triangulations. (2021).
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.
Renben Shu, Chen Zhou, and Mohan S Kankanhalli. 1995. Adaptive marching cubes. *The Visual Computer* 11, 4 (1995), 202–217.
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.
Stephanie Wang and Albert Chern. 2021. Computing minimal surfaces with differential forms. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–14.
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.
Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Yikang Yang, Yajie Bao, Jiachen Qian, Siyu Zhu, Xun Cao, Philip Torr, et al. 2025. Direct3d-s2: Gigascale 3d generation made easy with spatial sparse attention. *arXiv preprint arXiv:2505.17412* (2025).
Chris Yu, Caleb Brakensiek, Henrik Schumacher, and Keenan Crane. 2021. Repulsive surfaces. *arXiv preprint arXiv:2107.01664* (2021).

A MANIFOLD PROPERTY

In the even-sum case, *i.e.* when the edge coordinates around each triangle in the background grid obey the even sum condition of Section 2.3, 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 guaranteed to be edge-manifold everywhere, and vertex-manifold at interior vertices, but it may contain nonmanifold "pinch" vertices at the boundary. Note, though, that such pinches are easily split apart into several manifold vertices, and after performing such a split our primal and dual algorithms will produce manifold outputs even when the even-sum condition does not hold.



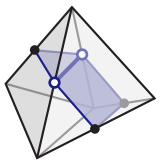
Theorem A.1. *In the even sum case, both the primal and dual algorithms yield manifold connectivity,* i.e.*, an abstract simplicial 2-complex homeomorphic to a surface without boundary.*

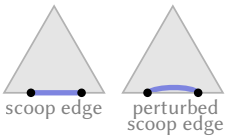
In the odd sum case, the primal algorithm yields an edge-manifold simplicial 2-complex, which may have non-manifold vertices on the boundary.

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 output simplicial complex is manifold or edge-manifold as appropriate. Finally, we note that taking the dual of a manifold simplicial complex (possibly with boundary) and triangulating yields another manifold simplicial complex, which has boundary if and only if the primal complex had boundary. □

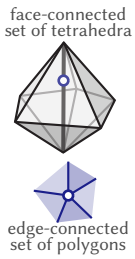
Lemma A.2. *The collection of primal spanning polygons constructed before Section 3.2.3 form an edge-manifold cell complex. In the even-sum case, the cell complex is also guaranteed to be vertex manifold, and in either case interior vertices are always manifold.*

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 2.3) which geometrically run along the edges of the tetrahedral grid. These scoop edges are still constructed in Section 2.3 by reconstructing segments from their edge coordinates in some face, and even an infinitesimal perturbation is enough to push them into their associated face. Thus, when considering the mesh connectivity even these scoop edges are contained in at most two polygons, so they are also manifold edges in the cell complex. Note, though, that as depicted in Figure 8 (*left*) we may have multiple scoop edges connecting the same pair of vertices.





To conclude, we show that in the even-sum case the cell complex is vertex manifold. Since we just saw that every edge is manifold, it suffices to check that the faces neighboring each vertex 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 exactly one polygon containing the vertex, and two such polygons share an edge if and only if the corresponding tetrahedra share a face, so we conclude that the neighboring polygons do indeed form a single edge-connected component.



Note that if the even-sum condition does not hold, some of the neighboring polygons may be missing, so the vertex may fail to be manifold. But if the vertex is an interior vertex of the final mesh, then none of its neighboring polygons are missing, and so it must be manifold. □

B TETRAHEDRAL NORMAL CURVES

In this appendix, we establish some basic properties of normal curves on the boundary of a tetrahedron. Note that we assume only that the *curves* obey the normality conditions from Section 2.3 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 1 has an integer solution if and only if the normality conditions for edge coordinates are satisfied (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 1 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. If we simply write down the RREF format of the augmented linear system in 1, we get:

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

So the solutions are non-negative integers if and only if even-sum condition and triangle inequality is met on all faces ijk.

Furthermore, we 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 will make the smallest q_{ij} equal 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 1, then for every edge ij we can write:

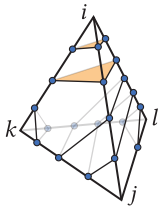
$$e_{ij} = t_i + t_j + q_{il} + q_{ik} \tag{7}$$

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 will be 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:

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

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

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

Note that 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 16 *(top left)* for an example.

B.2 Connected Components of Normal Curves

Here we analyze the number of connected components a triangle-free set of normal curves on a tetrahedron, *i.e.* a set of normal curves with no curve of length 3. Building on B.3, we can assume that the edges coordinates have the (d_1, d_2) pattern.

Theorem B.4. *Let Γ be a triangle-free collection of normal curves on the boundary of a tetrahedron. If we let d_1 and d_2 denote the non-zero diagonal cuts, the number of connected components of Γ is exactly* $\gcd(d_1, d_2)$.

Proof. We show this by introducing 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 16. 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 also connected similarly along the top of the segment.
- Finally, 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 from top of the segment.
- The rightmost $2d_2$ points are connected from bottom of the segment.
- The leftmost $2(d_1 - d_2)$ points are connected also from the bottom.

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).

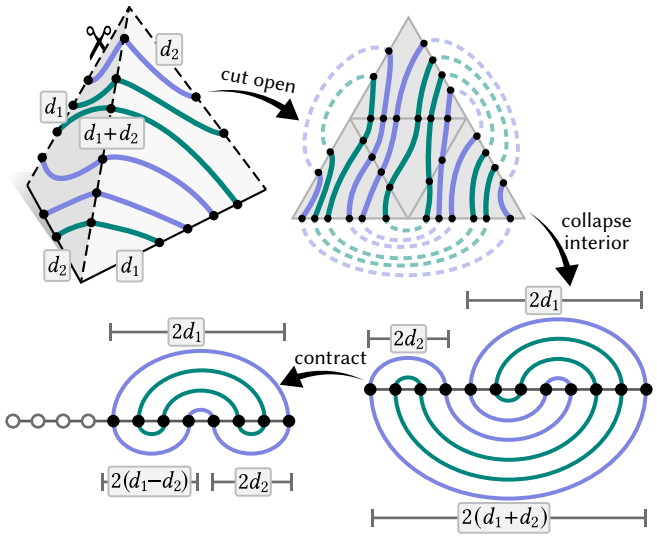


Fig. 16. Here we illustrate our proof of Theorem B.4, that the number of curves on the boundary of a tetrahedron with two non-zero diagonal cuts d_1, d_2 is exactly $\gcd(d_1, d_2)$. We begin by cutting open the tetrahedron and collapsing the resulting triangle down to a line segment, creating a topologically-equivalent 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.

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. *Let Γ be a triangle-free collection of normal curves on the boundary of a tetrahedron. Then all loops in Γ consist of the same number of segments, which we call the length ℓ.*

Proof. At the terminal state of the process outlined in the proof of Theorem B.4, color the points with $d = \gcd(d_1, d_2)$ colors. The color of every point determines its connected component. By reversing the operation, the number of colored points of the same type remains fixed. So there is an equal number of points of each color in the beginning and so each connected component has an equal contribution on the flattened edge. The choice of which edge to flatten 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. *When we have a (d_1, d_2) configuration, with $d = \gcd(d_1, d_2)$, the length of every component is exactly $4\frac{d_1+d_2}{d}$. Consequently, when $d_1, d_2 \geq 1$, the number of segments is at least 8.*

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

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. Before splitting the tetrahedron, the edge coordinates emanating from every vertex are exactly d_1, d_2, and $d_1 + d_2$. The splitting defined in Equation 4 introduces 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 edge coordinates at the interior vertex are all strictly less than $d_1 + d_2$, and so our splitting procedure results in maximum edge coordinate of at least one of the vertices to decrease.

The case where $d_1 = d_2$ is the octagon case which is handled explicitly without requiring further subdivision. So 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. Also note that due to our choice for the interior edge coordinates, normality conditions are satisfied for all the new tetrahedra, allowing the splitting procedure to continue recursively until termination. □

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.*

Note that it is enough to show that these surfaces do not intersect themselves *locally* inside their local tetrahedron and since they are always inside the tetrahedron 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, is contained in the convex hull of γ.*

Proof. The discs we construct for triangles and quads are a simple triangulation of them, hence contained in their 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. For surfaces that have portions built directly on the boundary faces of the tetrahedron (smeared against the tet faces); these regions are enclosed between two segments of γ and so are contained within its convex hull. □

Next we go over different types of surfaces that we construct and prove that they do not intersect. We start by the simplest type of curves and remove the constructed surfaces for each case until non is left.

Theorem D.3. *Triangles do not intersect themselves or any other 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 vertex i so they cannot intersect any other surfaces either. □

So we can remove the corner triangles and assume that no corner triangles exist in our next intersection-free arguments.

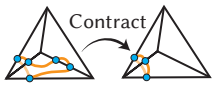
Theorem D.4. *Surfaces constructed for contractible type non-normal curves do not intersect themselves and other surfaces.*

Proof. A non-normal curve γ of contractible type is not contained in another curve of the same kind (Lemma D.6). 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. □

So we can remove the smeared polygons generated by contractible curves and make our next intersection-free arguments assuming that these types of curves don't exist.

Theorem D.5. *Surfaces constructed for corner-type non-normal curves do not intersect themselves and other surfaces.*

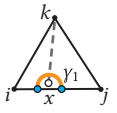
Proof. A corner-type curve γ also contains some smeared polygons similar to those in contractible type curves (Theorem D.4). With the same argument as before, we see that these polygons do not intersect themselves or others. Now consider the vertex i of the tetrahedron that γ loops around. The final 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 imagine contracting the non-normal curve until it becomes a normal corner curve at vertex i (inset for contraction). The contraction process pulls γ tight at non-normal segments until it becomes normal. More formally, we consider the following: at every 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. Once the contraction is done, though, we just 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. □



Now we prove the key lemma on smeared polygons which we used above in Theorem D.4.

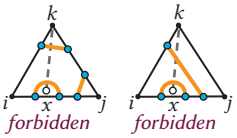
Lemma D.6. *A curve γ of contractible or corner-type can not be contained in another contractible or corner-type curve.*

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.7, providing our desired contradiction. □



Lemma D.7. *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. □



These lemmas and theorems 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.9 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 themselves an other surfaces.

Theorem D.8. *If γ is a diagonal-type non-normal curve, or if it is 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.9 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. □

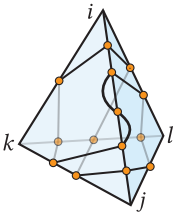
Next we show why diagonal-type non-normal curves do not coexist with the other diagonal-type curves. In general, this type of curve can have an arbitrary number of segments, and an arbitrary number of scoops.

Theorem D.9. *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' on our tetrahedron. First we show that we can make several simplifying assumptions on γ' with loss of generality.

First, note that γ' must correspond to the same diagonal as γ—e.g. if γ separates vertices i, j from k, l, then γ' separates the same pair of vertices. This follows from the fact that γ and γ' do not intersect.

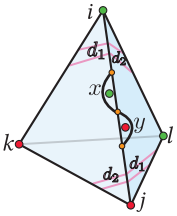
We can also assume that γ' is normal. If it is non-normal, we can contract it and remove its scoops, until it becomes normal.



We can also contract γ and stop the contraction right before it becomes normal; we can take γ to be one contraction away from being normal. At this minimally non-normal state, γ has certain properties that makes the proof simpler. In particular, such a curve γ must have exactly two scoops, which scoop from same edge ij into two different faces ijk and ijl and share one vertex (see inset for an example). These properties are shown in Lemma D.10. We show that this *double-scoop* structure cannot exist in presence of another normal diagonal-type curve γ.

First we consider the regions created by γ; i.e. assume that the only curve on the tetrahedron's boundary is γ. Take a point x inside the scoop on face ijk, as illustrated in the inset below. We know that no corner segment can exist at corner k on face ijk, so x is not in the same region as k (a path from x to k intersects γ only once at the scoop). Also consider a point y inside the scoop on face ijl. With same argument, y can not be in the same region as l, and since we know that x and y are in different regions, then k and l are in different regions as well. For convenience we label k and y's region with A, and l and x's region with B. Vertices i, j could be in either region, and we assume without loss of generality that i is in region A and j is in B.

Now consider the curve γ' and its edge coordinates. Since it is a normal and diagonal-type curve, its edge coordinates must have the (d_1, d_2) pattern mentioned earlier; i.e. each edge coordinate is either d_1, d_2, or $d_1 + d_2$. Also since we are considering a single curve γ' (with one component), we must have $\gcd(d_1, d_2) = 1$ (see B.4). So d_1 and d_2 cannot both be even. Consider the corner segments on γ' on faces ijk and ijl. Since both faces have a scoop on edge ij, these corner segments must have the following structure:



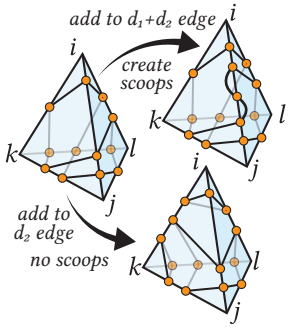
- No corner segments can exist at either corners k or l.
- Corner segments on face ijk, at corner i, are all on one side of the scoop on ijk; and they are on the closer side.
- Corner segments on face ijl, at corner i, are all on one side of the scoop on ijk; and they are on the closer side.

So the edge coordinate ij – for curve γ' – must be the $d_1 + d_2$ edge, and edges ik, kj, li, lj are the d_1 and d_2 (see inset). Also since the corner segments on either face are on different sides the scoops, we must have $d_1 = d_2$, and since γ' is a single component, we must have $d_1 = d_2 = 1$.

But we know that γ' has to be separating i, l from k, j so γ''s edge coordinates on edges jk and il must be even numbers. But they are odd (d_2), providing our desired contradiction. Thus a non-normal diagonal type curve γ cannot co-exist with another diagonal-type curve. □

Lemma D.10. *We call a curve γ minimally non-normal if a single contraction step will make it normal. A minimally non-normal γ has exactly two scoops on a single edge ij. One on face ijk and one on face ijl, and the two scoops share a single vertex.*

Proof. Consider the edge coordinates corresponding to γ. We know that if we do one more contraction step, it will become a normal curve with the (d_1, d_2) pattern. Consider the contracted version of γ and name it γ'. We show that to turn γ' into a non-normalcurve with a single step, we have a limited number of options. Note that turning γ' to a non-normal curve, i.e. reverse of a contraction step, can be done via adding 2 to one of the edges. We expect this addition to generate a non-normal condition on at least a triangle. If we add a 2 to a d_1 edge, then on both adjacent triangles the normality conditions hold, and so no scoop is generated. Same goes for a d_2 edge. So to generate a scoop, we can only add 2 to a $d_1 + d_2$ edge. If $d_1 \neq d_2$ This generates exactly two scoops on the adjacent faces and they have the properties mentioned in the statement.

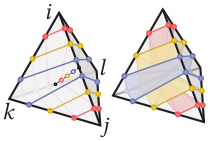


Note that if $d_1 = d_2$, this procedure generates two scoops on either face that exactly align with each other, adding a size 2 curve and so γ' could not have been obtained from γ. □

The only remaining case is the octagon case ($k > 1$, $\ell = 8$, see Section 3.2 and Figure 7). 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 7. 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.8.

Theorem D.11. *Consider k octagonal curves γ_i on the boundary of the tetrahedron. Their constructed surfaces from 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. This segment connects a point between the points d and $d+1$ 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.



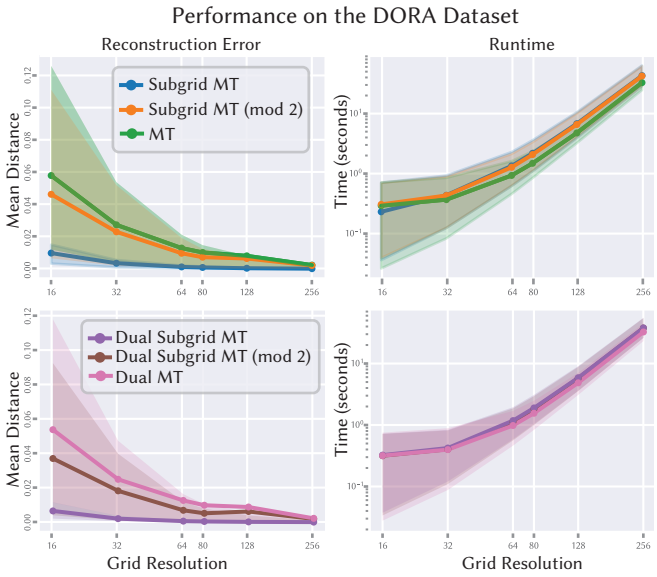


Fig. 17. Here we show the accuracy and performance of the subgrid mod 2 ablation. Its error profile is much closer to classical marching tetrahedra than to subgrid marching tetrahedra, emphasizing the importance of the fact that subgrid marching tetrahedra is able to make use of *all* intersections along each edge.

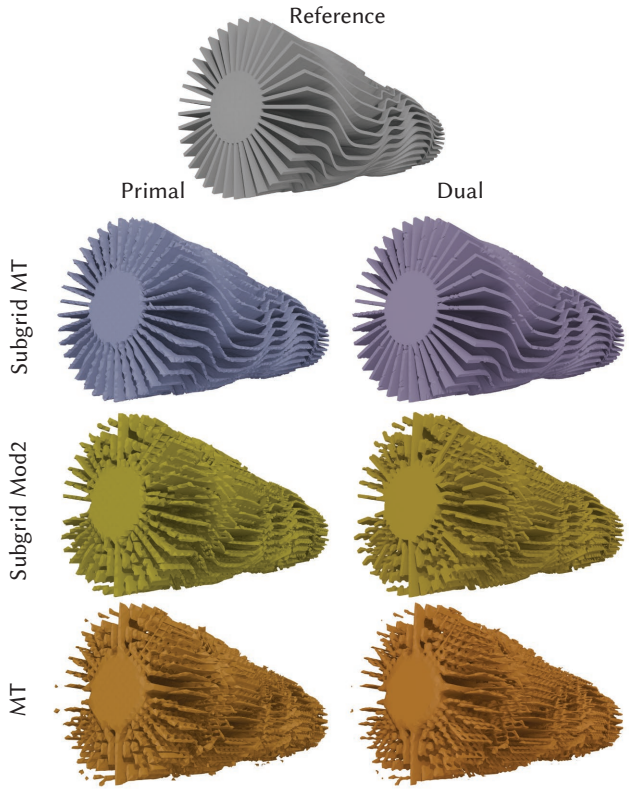
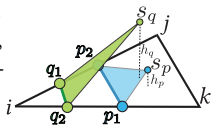


Fig. 18. Comparison of our subgrid approach against its mod 2 version and marching tetrahedra, in a 100^3 grid. Subgrid mod 2 has slightly better geometric quality than marching tetrahedra but still cannot produce more than one disc per tetrahedron.



Consider 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). 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 that s_p. Our placement of Steiner points with respect to every face exactly satisfies this condition; i.e. 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 makes use of two kinds of information that are unavailable to classical marching tetrahedra—the *number* of intersections e_{ij} along each edge, along with the *locations* of the intersections. In this appendix we show an ablation where we provide the intersection locations to marching tetrahedra, while still limiting it to at most one intersection per edge. In particular, the parity of the intersection count e_ij determines whether the two end points of edge ij are in different regions of $\mathbb{R}^3$ (i.e. inside vs outside the iso-surface). This is enough information for detecting *sign changes* across an edge, which is required by marching tetrahedra. Then we can extract an aggregated intersection location along the edge by averaging the positions of all of the true intersections. Note that if there is only one intersection, then this gives us an exact level-set location. In short, this modification improves the accuracy of the primal vertex positions, but produces the same mesh connectivity as classical marching tetrahedra.

We show a performance plot including this modified marching tetrahedra, which we call "subgrid mod 2," in Figure 17 for ablation study and also show some examples in Figure 18. In both cases we see that subgrid mod 2 improves slightly over classical marching tetrahedra, but features much more distortion than the full subgrid marching tetrahedra algorithm.

Received January 20XX