Surface Simplification using Intrinsic Error Metrics

HSUEH-TI DEREK LIU, Roblox & University of Toronto, Canada
MARK GILLESPIE, Carnegie Mellon University, USA
BENJAMIN CHISLETT, University of Toronto, Canada
NICHOLAS SHARP, University of Toronto & NVIDIA, USA
ALEC JACOBSON, University of Toronto & Adobe Research, Canada
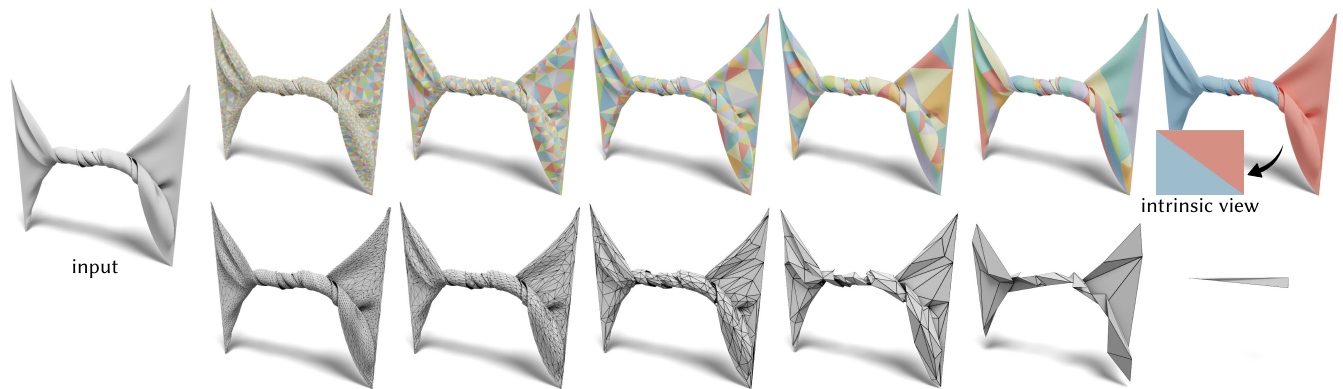KEENAN CRANE, Carnegie Mellon University, USA

Fig. 1. Whereas traditional extrinsic simplification *(bottom row)* must simultaneously juggle element quality and approximation error, triangles produced by our intrinsic scheme *(top row)* can wrap around the original surface—nicely approximating the underlying function space without changing the geometry. Coarse meshes or hierarchies produced by this scheme can be used in "black box" fashion to accelerate solvers without changing user inputs/outputs.

This paper describes a method for fast simplification of surface meshes. Whereas past methods focus on visual appearance, our goal is to solve equations on the surface. Hence, rather than approximate the extrinsic geometry, we construct a coarse *intrinsic triangulation* of the input domain. In the spirit of the *quadric error metric (QEM)*, we perform greedy decimation while agglomerating global information about approximation error. In lieu of extrinsic quadrics, however, we store intrinsic tangent vectors that track how far curvature "drifts" during simplification. This process also yields a bijective map between the fine and coarse mesh, and prolongation operators for both scalar- and vector-valued data. Moreover, we obtain hard guarantees on element quality via intrinsic retriangulation—a feature unique to the intrinsic setting. The overall payoff is a "black box" approach to geometry processing, which decouples mesh resolution from the size of matrices used to solve equations. We show how our method benefits several fundamental tasks, including geometric multigrid, all-pairs geodesic distance, mean curvature flow, geodesic Voronoi diagrams, and the discrete exponential map.

Authors' addresses: Hsueh-Ti Derek Liu, Roblox & University of Toronto, Canada, hsuehtil@gmail.com; Mark Gillespie, Carnegie Mellon University, USA, mgillesp@cs.cmu.edu; Benjamin Chislett, University of Toronto, Canada, chislett.ben@gmail.com; Nicholas Sharp, University of Toronto & NVIDIA, USA, nmwsharp@gmail.com; Alec Jacobson, University of Toronto & Adobe Research, Canada, jacobson@cs.toronto.edu; Keenan Crane, Carnegie Mellon University, USA, kmcrane@cs.cmu.edu.

CCS Concepts: • **Computing methodologies → Mesh geometry models**.

Additional Key Words and Phrases: geometry processing, mesh simplification

1 INTRODUCTION

Algorithms for mesh simplification were originally motivated by the need to maintain visual fidelity for rendering and display [Hoppe 1996]. Since image generation fundamentally depends on *extrinsic* quantities like vertex positions and surface normals, extrinsic metrics (like the distance to the original surface) were a natural choice [Garland and Heckbert 1997]. However, in a wide variety of problems throughout computer graphics, geometry processing, and scientific computing, the goal is to solve equations on surface meshes, rather than display them on screen. Since functions on surfaces have derivatives only in tangential directions, differential operators appearing in these equations (such as the Laplacian) are almost always *intrinsic*—even in cases where one solves for extrinsic quantities [Finnendahl et al. 2023]. Hence, to develop fast, accurate solvers for equations on surfaces, it is natural to seek error metrics that focus on intrinsic geometry, *i.e.*, quantities that depend only on distances along the surface, rather than coordinates in space.
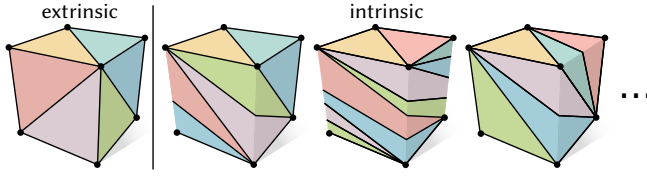
Fig. 2. In contrast to standard extrinsic meshes *(left)*, where edges are straight line segments in $\mathbb{R}^n$, intrinsic triangulations connect vertices along arbitrary geodesic paths between vertices *(right)*.

A principled approach to this problem is enabled by recent work on flexible data structures for *intrinsic triangulations* [Fisher et al. 2006; Gillespie et al. 2021a; Sharp et al. 2019a]. Roughly speaking, an intrinsic triangulation is one where edges need not be straight line segments in space, but can instead be any *geodesic arc* along the surface (Fig. 2). More generally, an intrinsic triangulation is an assignment of positive lengths to edges—with no requirement that there be vertex positions that realize these lengths. Not surprisingly, this construction provides a vastly larger space of possibilities for mesh processing, by de-coupling the elements used to approximate the geometry from those used to approximate functions on the surface. Moreover, standard objects like the Laplacian can still be built directly from edge lengths, allowing many existing algorithms to be run as-is. However, past work has not provided any mechanism for *coarsening* intrinsic triangulations—as proposed in this paper.

1.1 Method Overview

In the extrinsic case, greedy iterative decimation has proven remarkably effective, with a notable example being the *quadric error metric (QEM)* of Garland and Heckbert [1997]. Despite being over a quarter-century old, QEM remains the method of choice in many modern systems [Karis et al. 2021]. The key insight of QEM is that one obtains useful information about global approximation error by aggregating the distortion induced by each local operation into a constant-size record at each vertex. Our method, which we dub the *intrinsic curvature error (ICE) metric*, adopts the same basic strategy, but tracks intrinsic rather than extrinsic data. There are two geometric perspectives on this metric, developed further in Sec. 5:

- **Local Picture.** Whereas extrinsic methods penalize deviation of positions in space, we penalize changes in the intrinsic metric. The local cost of removing a vertex is determined by the *optimal transport cost* of redistributing its curvature to neighboring vertices. More precisely, we compute an approximate *1-Wasserstein distance* between Gaussian curvature distributions before and after flattening the removed vertex (Fig. 3, *top left*).
- **Global Picture.** One can also aggregate information about where error comes from in order to make better greedy decisions. In QEM, aggregation is achieved by summing *quadrics*, which approximate the squared distance to the ancestors of each vertex. In contrast, the ICE metric tracks an approximate curvature-weighted center of mass. More precisely, at each vertex i we track (i) the total accumulated curvature and (ii) a tangent vector $\mathbf{t}_i$ such that the exponential map $\exp_i(\mathbf{t}_i)$ approximates the curvature-weighted *Karcher mean* of all ancestors (Fig. 3, *bottom*).
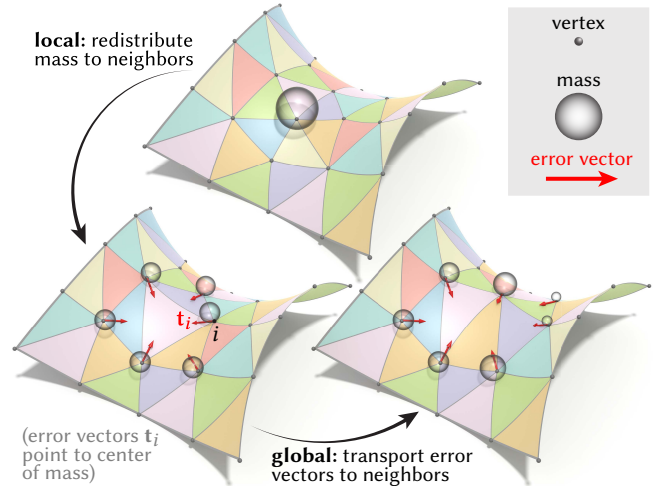

Fig. 3. Our method constructs a coarse triangulation over a fixed geometric domain. In each **local** step we redistribute curvature or other quantities from a removed vertex to its neighbors. From step to step we also agglomerate **global** information about error via tangent vectors that point toward the approximate center of mass of the decimated vertices.

By greedily applying decimation operations that keep these tangent vectors small, we maintain a good approximation of the original intrinsic geometry—for instance, curvature does not "drift" from one place to another. Moreover, unlike the extrinsic setting, we can freely change mesh connectivity without incurring any additional geometric error—making it trivial to improve element quality via tools like *intrinsic Delaunay triangulation* (Sec. 3.3.2).

2 RELATED WORK

Numerous tasks in geometry processing and simulation use coarsened or hierarchical representations of geometry [Garland 1999; Guskov et al. 1999], including compression (*e.g.* using wavelets [Peyré and Mallat 2005; Schröder 1996] or Laplacian bases [Karni and Gotsman 2000]), surface modeling [Botsch and Kobbelt 2004; Kobbelt et al. 1998; Zorin et al. 1997], physical simulation [Grinspun et al. 2002; Zhang et al. 2022], parameterization [Ray and Lévy 2003], and eigendecomposition [Nasikun et al. 2018; Nasikun and Hildebrandt 2022]. Though the core computation in many of these tasks is inherently intrinsic, the coarsening process itself has so far been performed in the extrinsic domain. We briefly review work on surface simplification and remeshing relevant to our approach; see Botsch et al. [2010, Chapter 7] for a more detailed discussion.

2.1 Intrinsic Triangulations

As noted in Sec. 1, the machinery of intrinsic triangulations is central to our approach Sharp et al. [2021]. Early work on intrinsic triangulations explored the basic formulation [Regge 1961] and its deep connection to *Delaunay triangulations* [Bobenko and Springborn 2007; Indermitte et al. 2001; Rivin 1994]. Subsequent work on *discrete uniformization* [Gu et al. 2018; Luo 2004a; Springborn et al. 2008] and an intrinsic *Laplace operator* [Bobenko and Springborn 2007] has recently stimulated broader applications in geometry

processing [Finnendahl et al. 2023; Gillespie et al. 2021a,b; Sharp and Crane 2020a,b; Sharp et al. 2019a; Takayama 2022]. Several general-purpose data structures have been developed for intrinsic triangulations [Fisher et al. 2006], including those that support refinement operations [Gillespie et al. 2021a; Sharp et al. 2019a]. However none support operations needed for intrinsic coarsening, as proposed here.

2.2 Surface Simplification

2.2.1 Local Decimation. A large class of coarsening methods apply incremental local decimation via vertex removal [Schroeder et al. 1992], vertex redistribution [Turk 1992], vertex clustering [Alexa and Kyprianidis 2015; Low and Tan 1997; Rossignac and Borrel 1993], face collapse [Gieng et al. 1997], and edge collapse strategies [Hoppe 1996], including QEM [Garland and Heckbert 1997]. Although methods based on global energy minimization can produce impressive results [Cohen-Steiner et al. 2004; Hoppe et al. 1993], local decimation schemes remain popular since they are easy to implement, typically exhibit near-linear scaling (each decimation operation is essentially $O(1)$), and can easily meet an exact target vertex budget (by stopping when the budget is reached). QEM-based simplification in particular has endured because its cheap *local* heuristic yields excellent *global* approximation when aggregated over many decimation operations. Moreover, QEM is easily adapted to other attributes such as color or texture [Garland and Heckbert 1998; Hoppe 1999].

In the intrinsic setting, we likewise favor a local decimation approach because it yields fast execution times and near-linear scaling (Sec. 8.2), and easily incorporates rich geometric criteria (Sec. 8.6); as in QEM, aggregating information across local operations leads to high-quality global approximation (Sec. 5). Unlike extrinsic, visualization-focused methods, the intrinsic setting also furnishes *quality guarantees* valuable for simulation (Sec. 6.3).

2.2.2 Global Remeshing. In contrast to local decimation, which incrementally mutates the given triangulation, global remeshing methods seek only to approximate the given geometry, using an entirely new triangulation (possibly a coarser one). Global remeshing can be performed via, *e.g.*, streamline tracing [Alliez et al. 2003] or global parameterization [Alliez et al. 2005, 2002], with significant emphasis in recent years on field-aligned methods [Bommes et al. 2013b]. However a high-quality global parameterization is notoriously difficult to compute—especially if a seamless grid is needed for mesh generation [Bommes et al. 2013a]. Unlike our ICE metric (which is based on transport cost), error metrics based on local parametric distortion [Schmidt et al. 2019] or pointwise changes in curvature [Ebke et al. 2016] can fail to account for global tangential "drift" across the surface. Moreover, parameterization-based methods do not take advantage of the flexibility of intrinsic triangulations, requiring at all times an explicit embedding into the 2D plane. A key exception are methods based on modification of edge lengths [Capouellez and Zorin 2022], most notably the *conformal equivalence of triangle meshes (CETM)* algorithm of Springborn et al. [2008], which we apply locally to define our vertex removal operation (Sec. 4.1). A more global intrinsic coarsening strategy might be to conformally map the whole surface to a high-quality *cone metric* [Fang et al. 2021; Li et al. 2022; Soliman et al. 2018] then
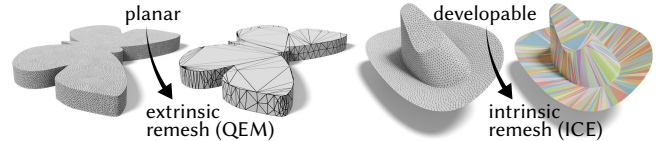

Fig. 4. Just as extrinsic simplification pays no cost for removing vertices from an initially flat region *(left)*, our intrinsic method pays no cost for removing vertices from an initially developable region *(right)*.

remove all flat vertices; however, even just computing a good cone configuration is already more expensive than our entire algorithm.

2.2.3 Tracking Correspondence. Many applications require not only a coarse mesh, but also a way of mapping various attributes between coarse and fine meshes [Kharevych et al. 2009; Li et al. 2015; Liu et al. 2019]. Early methods compose local 2D mappings (inset) to obtain a so-called *successive mapping* [Cohen et al. 1997; Khodakovsky et al. 2003; Lee et al. 1998], as more recently discussed by Liu et al. [2020, 2021]. Alternatively, correspondence can be defined via normal offsets [Guskov et al. 2000], texture domain chart boundaries [Sander et al. 2001], or bijective projection [Jiang et al. 2020, 2021], though methods based on extrinsic correspondence can struggle in the presence of, *e.g.*, mesh self-intersections (Fig. 20). Since our method performs both simplification *and* mapping via intrinsic triangulations, we can achieve lower-distortion mappings than methods restricted to the smaller space of extrinsic mesh sequences (Figs. 18 and 19), in turn improving the numerical behavior of many algorithms (Sec. 8).
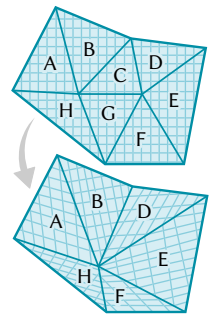
2.3 Mesh Hierarchies

Coarsening and correspondence tracking are also key components of multi-resolution methods. Whereas extrinsic coarsening is essential for, *e.g.*, adaptive visualization [Hoppe 1996], intrinsic coarsening is well-suited to multiresolution solvers such as geometric multigrid [Liu et al. 2021]. Here, *coarse-to-fine* schemes, *e.g.* based on subdivision surfaces [Zorin et al. 2000], yield regular connectivity and principled prolongation operators based on subdivision basis functions [de Goes et al. 2016; Shoham et al. 2019]. However, without careful preprocessing [Eck et al. 1995; Hu et al. 2022] subdivision behaves poorly on coarse, low-quality meshes encountered in the wild [Zhou and Jacobson 2016]. We instead adopt a robust *fine-to-coarse* strategy, via repeated decimation (Sec. 8.7). Though extensively studied for extrinsic meshes, both for adaptive rendering [Hoppe 1996, 1997; Popovic and Hoppe 1997] and modeling/simulation [Aksoylu et al. 2005; Liu et al. 2021; Manson and Schaefer 2011], a fine-to-coarse hierarchy based on intrinsic triangulations enables geometric multigrid to succeed on extremely low-quality meshes where past methods fail (Sec. 8.7).
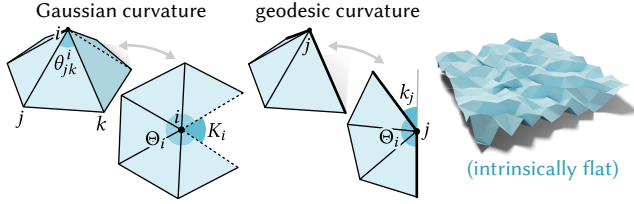
Fig. 5. Gaussian curvature K and geodesic curvature κ depend only on corner angles θ, which in turn depend only on edge lengths ℓ. Hence, surfaces that appear bent in $\mathbb{R}^3$ may in fact have zero intrinsic curvature *(right)*.

3 BACKGROUND

An intrinsic mesh encodes geometry via edge lengths rather than vertex positions; this data is in turn sufficient to support a wide variety of surface processing and simulation tasks. Here we give a brief review—see Sharp et al. [2021] for an in-depth introduction.

3.1 Connectivity

We consider a manifold, orientable triangle mesh $M = (V, E, F)$ with vertices $i \in V$, edges $ij \in E$, and faces $ijk \in F$. Likewise, we write $\widetilde{M} = (\widetilde{V}, \widetilde{E}, \widetilde{F})$ for any modification of M. We use degree(i) to denote the *(face) degree*, i.e., the number of faces containing i. A degree-1 boundary vertex is called an *ear*; otherwise it is a *regular* boundary vertex. We use u_i, u_{ij}, u_{ijk} to denote a value at a vertex, edge, and face, respectively, and u^i_{jk} for a quantity at corner i of triangle ijk. Sometimes we also consider values u_{ij} on *oriented* edges, where $u_{ij} \neq u_{ji}$.
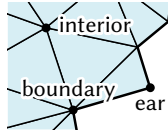
To represent the full space of intrinsic triangulations (which furnishes important algorithmic guarantees [Bobenko and Springborn 2007]), we assume that M is a Δ-*complex* [Hatcher 2002, §2.1]. Unlike a simplicial complex, elements in a Δ-complex are not uniquely determined by their vertices. For instance, two edges of the same face may be glued together to form a cone (see inset). Hence, we write $ijk \in F$ we refer to only one of possibly several faces with vertices i, j, and k—which themselves need not be distinct. Likewise, an edge ij may connect a vertex to itself ($i = j$); we refer to such edges as *self-edges*. Throughout we let $\mathcal{N}_i := \{ j \in V | ij \in E \}$ be the neighbors of vertex i, excluding i itself in the case of a self-edge ii. The connectivity of a Δ-complex can be encoded via an *edge gluing matrix* [Sharp and Crane 2020a, §4.1] or a *halfedge mesh* [Botsch et al. 2010, Chapter 2].

3.2 Geometry

Any set of positive edge lengths $\ell : E \to \mathbb{R}_{>0}$ that satisfy the triangle inequality $\ell_{ij} + \ell_{jk} > \ell_{ki}$ at each triangle corner determines a valid intrinsic metric, i.e., a Euclidean geometry for each triangle. We typically obtain initial edge lengths $\ell_{ij} = \|p_i - p_j\|$ from input vertex positions $p : V \to \mathbb{R}^3$, but in principle could start with any abstract metric (e.g., coming from a *cone flattening* [Soliman et al. 2018]). Interior angles $\theta^i_{jk} \in (0, \pi)$ at corners can be determined
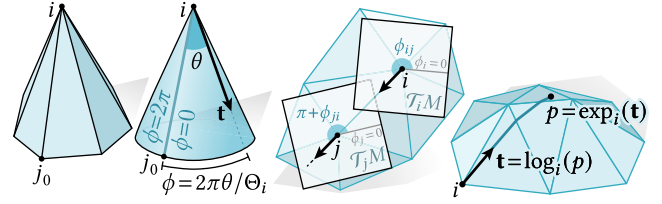

Fig. 6. Notation and conventions for tangent vectors *(left)*, parallel transport *(center)*, and the logarithmic/exponential maps *(right)*.

from the edge lengths, via the law of cosines. We use $\|\mathbf{v}\|$ to denote the Euclidean norm of any vector $\mathbf{v} \in \mathbb{R}^n$.

3.2.1 Curvature. Intrinsically, the curvature of a surface is completely described by its Gaussian and geodesic curvature—though an intrinsically flat surface can still be extrinsically bent like a crumpled sheet of paper (Fig. 5). On a triangle mesh, let $\Theta_i = \sum_{ijk \in \mathcal{N}_i} \theta^i_{jk}$ be the angle sum around vertex i. The *discrete Gaussian curvature* at interior vertex i is then given by the angle defect

$$K_i := 2\pi - \Theta_i, \tag{1}$$

measuring deviation from the angle sum 2π of the flat plane. Likewise, the *discrete geodesic curvature* at boundary vertex i is

$$\kappa_j := \pi - \Theta_j, \tag{2}$$

measuring deviation from a straight line.

3.2.2 Tangent Vectors. In a small neighborhood of any vertex i, the intrinsic metric looks like a cone of total angle Θ_i; we use $\mathcal{T}_i M$ to denote the set of tangent vectors at i (Fig. 6, *left*). Following Knöppel et al. [2013, §6], we express the direction of any tangent vector $\mathbf{t} \in \mathcal{T}_i M$ as a normalized angle $\phi := 2\pi\theta/\Theta \in [0, 2\pi)$, where θ is the angle of $\mathbf{t}$ relative to an arbitrary but fixed reference edge ij_0. The vector itself is then encoded as a complex number $\|\mathbf{t}\| e^{\iota\phi} \in \mathbb{C}$, where ι is the imaginary unit. In particular, the *angular coordinate* $\phi_{ij} \in [0, 2\pi)$ encodes the outgoing direction of an oriented edge ij; we use $\mathbf{e}_{ij} \in \mathcal{T}_i M$ to denote the vector with direction ϕ_{ij} and magnitude ℓ_{ij}. The corresponding direction at vertex j is given by $\phi_{ji} + \pi$. Hence, we can *parallel transport* vectors from $\mathcal{T}_i M$ to $\mathcal{T}_j M$ (Fig. 6, *center*) via a rotation by $\mathbf{R}_{ij} := e^{\iota((\phi_{ji}+\pi)-\phi_{ij})}$ (encoded by a unit complex number). See Sharp et al. [2019b, §3.3 and §5.2] for further discussion.

3.2.3 Exponential and Logarithmic Map. The *exponential map* $\exp_i(\mathbf{t})$ of a tangent vector $\mathbf{t}$ at vertex i computes the point p reached by starting at vertex i and walking straight (i.e., along a geodesic) with initial direction $\mathbf{t}$ for a distance $\|\mathbf{t}\|$ (Fig. 6, *right*). In particular, for any oriented edge ij we have $\exp_i(\mathbf{e}_{ij}) = j$. For a given point $p \in M$, the *logarithmic map* $\log_i(p)$ gives the smallest tangent vector $\mathbf{t}$ at i such that $\exp_i(\mathbf{t}) = p$. Note that, in general, $\log_i(j)$ may not yield the edge vector $\mathbf{e}_{ij}$, since there may be a shorter path from i to j.

3.3 Retriangulation

3.3.1 Intrinsic Edge Flip. Consider an edge ij contained in triangles ijk, jil. An *intrinsic edge flip* replaces ij with a geodesic arc along the opposite diagonal kl, where the length of the new edge kl is
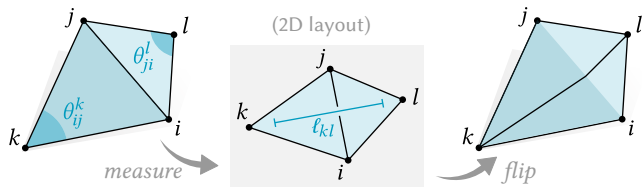
Fig. 7. An intrinsic flip of edge ij measures the length ℓ_{kl} of the opposite diagonal in a 2D layout, then updates connectivity. Greedily flipping until $\theta_{ij}^k + \theta_{ji}^l \leq \pi$ at all edges ij yields an *intrinsic Delaunay triangulation*.
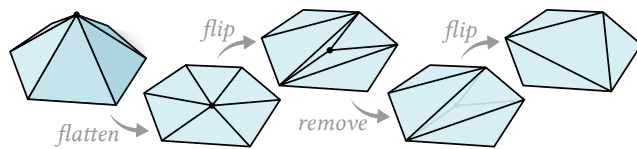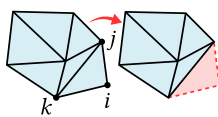

Fig. 8. We decimate an interior vertex by intrinsically flattening it, flipping to degree 3, removing it from the mesh, then flipping back to an intrinsic Delaunay triangulation. (For boundary vertices, we instead flip to degree 2.)

determined via a planar unfolding (Fig. 7). Hence, an intrinsic flip exactly preserves the original geometry—in particular, the discrete curvature is unchanged. An edge ij is *flippable* if and only if (i) degree(i), degree(j) ≥ 2 and (ii) triangles ijk, jil form a convex quadrilateral, *i.e.*, if both $\theta_{jk}^i + \theta_{lj}^i$ and $\theta_{ki}^j + \theta_{il}^j$ are less than π [Sharp and Crane 2020b, §3.1.3]. Note that these conditions are considerably easier to check than in the extrinsic case [Liu et al. 2020, App. C].

3.3.2 Intrinsic Delaunay Triangulation. A triangulation is *intrinsic Delaunay* if it satisfies the angle sum condition $\theta_{ij}^k + \theta_{ji}^l < \pi$ at all interior edges $ij \in E$ (Fig. 7). Such triangulations extend many useful properties of 2D Delaunay triangulations to surface meshes—[Sharp et al. 2021, §4.1.1] gives a detailed list. A triangulation can be made intrinsic Delaunay via a simple greedy algorithm: flip non-Delaunay edges until none remain [Bobenko and Springborn 2007].

4 VERTEX REMOVAL

Extrinsic simplification methods reduce vertex count by making local changes to connectivity [Garland and Heckbert 1997; Hoppe 1996; Schroeder et al. 1992]. We extend local simplification to the setting of intrinsic triangulations, using *vertex removal* as our atomic operation. Intrinsic simplification provides strictly more possibilities than its extrinsic counterpart (see inset), since any extrinsic operation can be represented intrinsically.

Our method removes a vertex i in three steps, illustrated in Fig. 8:

(1) Intrinsically flatten i (Sec. 4.1).
(2) Remove i from the triangulation (Sec. 4.2).
(3) Flip to an intrinsic Delaunay triangulation (Sec. 3.3.2).

The vertex removal step extends the scheme from [Gillespie et al. 2021a, §3.5] to boundary vertices. Note that all changes to the geometry occur in the first step, redistributing the curvature at i to neighboring vertices $j \in \mathcal{N}_i$. The second step merely retriangulates a flat region, and the third step performs only intrinsic edge flips. Hence, our error metric in Sec. 5 will need only consider the first (flattening) step to prioritize vertex removals. Maintaining a Delaunay triangulation at each step helps ensure numerical robustness throughout simplification.

Special cases. To remove an ear vertex i, it is tempting to simply remove the triangle ijk containing i. However, doing so leaves points on the fine mesh that do not map to

any point on the coarse mesh. Instead, we transform any ear into a regular boundary vertex by first flipping the opposite edge jk.

We cannot remove vertices i incident on a boundary self-edge, since every boundary loop must contain at least one vertex. Likewise, vertices i of self-faces (*i.e.*, triangles with only a single distinct vertex) can cause trouble for flipping, and are skipped.
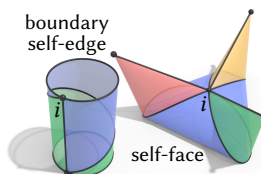
4.1 Vertex Flattening

We first eliminate all curvature at vertex i. For this operation to remain local and valid we must bijectively flatten the neighborhood $\mathcal{N}_i$, while keeping edge lengths along the boundary of this region fixed. Extrinsic flattening schemes can fix boundary vertices [Floater 1997; Weber and Zorin 2014], but it is unclear how to construct the least-distorting boundary polygon with prescribed lengths. In contrast, the CETM algorithm of Springborn et al. [2008] supports edge length constraints, and operates directly on an intrinsic triangulation. Moreover, using a conformal map will simplify the tangent vector prolongation scheme in Sec. 7.3.

Following Luo [2004b], two sets of edge lengths ℓ and $\tilde{\ell}$ are *conformally equivalent* if there exist values $u : V \to \mathbb{R}$ such that

$$\tilde{\ell}_{ij} = e^{(u_i + u_j)/2} \ell_{ij}, \quad \forall ij \in E. \tag{3}$$

Given initial lengths ℓ, CETM finds conformally equivalent edge lengths $\tilde{\ell}$ with prescribed angle sums $\widehat{\Theta}_i$ by minimizing a convex energy $\mathcal{E}(u)$.

In our case, we need only determine a single scale factor u_i at the removed vertex i. We let $\widehat{\Theta}_i = 2\pi$ for interior vertices (zero Gaussian curvature), and $\widehat{\Theta}_i = \pi$ at regular boundary vertices (zero geodesic curvature). Setting $u_j = 0$ for all other vertices $j \in \mathcal{N}_i$ ensures that the boundary lengths are unchanged—in fact, Springborn et al. [2008, Appendix E] show that these boundary conditions also induce minimal area distortion. Using the expressions for the gradient and Hessian of $\mathcal{E}$ from [Springborn et al. 2008, Equations 9 and 10], we solve the 1D root finding problem $\nabla\mathcal{E}(u_i) = 0$ via Newton's method:

$$u_i \leftarrow u_i - \frac{\widehat{\Theta}_i - \sum_{ijk \in \mathcal{N}_i} \theta_{jk}^i}{\frac{1}{2} \sum_{ijk \in \mathcal{N}_i} \cot\theta_{ij}^k + \cot\theta_{ki}^j}. \tag{4}$$

Notice that we express this formula as a sum over faces, so that it applies to both interior and boundary vertices. In practice, this scheme converges in about five iterations. Occasionally, the new edge lengths $\tilde{\ell}$ (computed via Eq. (3)) fail to satisfy the triangle inequality. In this case we try performing edge flips, *à la* Springborn

et al. [2008, §3.2]. If these flips still fail to resolve the issue, we skip this vertex and revisit it in future coarsening iterations.

4.2 Flat Vertex Removal

To remove a flattened vertex i, we flip it to a degree-3 vertex and replace the three triangles iab, ibc, ica incident on i with the single triangle abc (inset, left). Since the vertex neighborhood is already flat, these operations preserve the geometry. Gillespie et al. [2021a, Appendix D.1] show that iteratively flipping any remaining flippable edge ij incident on i will yield a degree-3 vertex, so long as the neighborhood $\mathcal{N}_i$ remains simplicial. Hence, at each step we first flip any self-edges ($i = j$); if there are none, we flip the edge ij with largest angle sum $\theta_{ij}^k + \theta_{ji}^l$ (since only convex triangle pairs can be flipped). In the rare case where degree(i) > 3 and no flippable edges remain, we skip this vertex removal and revert the mesh to its previous state. If i is a boundary vertex, we perform edge flips until degree(i) = 2 and replace the two resulting triangles iab, ica with the single triangle abc (inset, right). Here again the geometry is unchanged, since i has no geodesic curvature. When i is an ear vertex we need only flip the opposite edge to give i degree-2, while for regular boundary vertices we use the same procedure as for interior vertices.

After removal, we must also update the angles ϕ_{jk} and corresponding edge vectors $\mathbf{e}_{jk}$ for each edge jk with endpoints in $\mathcal{N}_i$ (Sec. 3.2.2). We then flip the mesh to an intrinsic Delaunay triangulation, *à la* Sec. 3.3.2.

5 ERROR METRIC

To prioritize vertex removals, we must define a notion of cost. Standard extrinsic metrics, such as QEM, are not appropriate: even if they could somehow be evaluated using intrinsic data, they would attempt to preserve aspects of the geometry that are not relevant for intrinsic problems (as discussed in Sec. 1). Our method is however inspired by the remarkable effectiveness of greedy local error accumulation in QEM. Likewise, metrics that focus on finite element equality (*à la* [Shewchuk 2002]) are not appropriate, since at intermediate steps of coarsening the triangulation used to encode the intrinsic geometry is transient and subject to change. Standard considerations from finite element theory do however provide good justification for flipping the final triangulation to Delaunay.

Our ICE metric is instead based on two intrinsic and triangulation-independent concepts: *optimal transport* [Peyré et al. 2019], and the *Karcher mean* [Karcher 2014]. Optimal transport helps quantify the effort of redistributing mass, providing the local cost for our "memoryless" metric (Sec. 5.1). Karcher means encode the center of mass of all fine vertices contributing to a coarse vertex i, providing the basis for our "memory-based" metric (Sec. 5.2). These two pieces fit together in a natural way: after a single vertex removal, the mass-weighted norm of all error vectors $\mathbf{t}_i$ encoding Karcher means

is exactly equal to the optimal transport cost. Hence, after many vertex removals this norm approximates the cost of transporting the initial fine mass distribution to the coarsened vertices. Vertex removals that keep cost small should hence be prioritized, since they better preserve the initial mass distribution. Just as in QEM, this information is captured by a fixed-size representation (masses and tangent vectors at each vertex) that is easily agglomerated during coarsening. To get a good *geometric* approximation, we use curvature as our basic notion of "mass", but can also use other attributes such as area (Sec. 5.4).

For clarity of exposition we first define error metrics in 2D, before generalizing to surfaces (Sec. 5.3), and incorporating data like curvature or other attributes (Sec. 5.4). Note that we view the memoryless error metric only as an intermediate step to explain the memory-based version, and use the memory-based metric for all results and experiments. As suggested by the inset example, the memoryless metric tends to keep vertices at highly-curved points, whereas the memory-based version better distributes vertices proportional to nearby curvature. However, there may be application contexts where the memoryless version is preferable (*e.g.*, [Hoppe 1999]).

5.1 2D Error Metric (Memoryless)

Consider a mass distribution $m : V \rightarrow \mathbb{R}_{\geq 0}$ at mesh vertices, representing any nonnegative user-defined quantity (signed quantities will be addressed in Sec. 5.4). Suppose we remove vertex i, redistributing its mass m_i to its immediate neighbors $j \in \mathcal{N}_i$. In particular, let $\alpha_{ij} \in [0, 1]$ be the fraction of m_i sent to vertex j (hence $\sum_{j \in \mathcal{N}_i} \alpha_{ij} = 1$), so that the new mass at j is

$$\widetilde{m}_j = m_j + \alpha_{ij} m_i. \tag{5}$$

5.1.1 Error Vectors. Suppose we want to track not only the mass distribution, but also where mass came from. Then at each vertex i we can store an *error vector* $\mathbf{t}_i$ (initially set to zero) pointing to the center of mass $\mathbf{c}_i$ of all vertices that contributed to the current value of m_i. Explicitly, after removing i, the center of mass at vertex j is

$$\widetilde{\mathbf{c}}_j = \frac{\alpha_{ij} m_i \mathbf{x}_i + m_j \mathbf{x}_j}{\alpha_{ij} m_i + m_j},$$
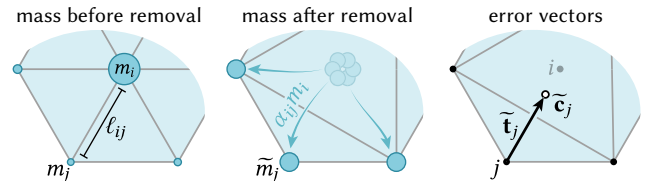

Fig. 9. The local cost of removing any vertex i is the *optimal transport cost* of transporting its mass m_i to its neighbors $j \in \mathcal{N}_i$. We can also calculate this cost as the sum of new masses $\widetilde{m}_j$ times the length of *error vectors* $\widetilde{t}_j$, which point to the new centers of mass $\widetilde{\mathbf{c}}_j$.

where $\mathbf{x}_i \in \mathbb{R}^2$ denotes the location of vertex i. Hence, the vector pointing from $\mathbf{x}_j$ to $\widetilde{\mathbf{c}}_j$ is

$$\widetilde{\mathbf{t}}_j = \widetilde{\mathbf{c}}_j - \mathbf{x}_j = \frac{\alpha_{ij} m_i \mathbf{e}_{ji}}{\alpha_{ij} m_i + m_j},$$

where $\mathbf{e}_{ji} = \mathbf{x}_i - \mathbf{x}_j$ is the vector along edge ji. The total cost of removing i can then be measured by summing up the mass-weighted norms of these vectors. Noting that $\|\mathbf{e}_{ji}\| = \ell_{ij}$, we get a cost

$$C_i = \sum_{j \in \mathcal{N}_i} \widetilde{m}_j \|\widetilde{\mathbf{t}}_j\| = \sum_{j \in \mathcal{N}_i} \alpha_{ij} m_i \ell_{ij}. \quad (6)$$

This cost also coincides with the so-called *1-Wasserstein distance* between the old and new mass distribution [Peyré et al. 2019, Chapter 2]. Intuitively, this distance measures the total "effort" of moving mass from i to neighbors j, penalizing not only the amount of mass moved, but also the distance traveled.

5.2 2D Error Metric (Memory-Based)

Rather than assign a cost to each vertex removal in isolation, we can accumulate information about how mass has been redistributed across all prior removals. At each step, we still update the mass distribution via Eq. (5), but now update vectors encoding the centers of mass via

$$\widetilde{\mathbf{t}}_j = \frac{\alpha_{ij} m_i (\mathbf{t}_i + \mathbf{e}_{ji}) + m_j \mathbf{t}_j}{\alpha_{ij} m_i + m_j}. \quad (7)$$

In other words, we re-express $\mathbf{t}_i$ relative to $\mathbf{x}_j$ by adding the edge vector $\mathbf{e}_{ji}$, then take the mass-weighted average of the old error vector $\mathbf{t}_j$ with this new vector. The overall cost is still evaluated via Eq. (6), but now approximates the effort of moving the initial mass distribution to the current one—rather than just penalizing the most recent change. This cost is only approximate since the 1-Wasserstein distance to the center of mass is not in general equal to the distance to the original fine distribution—but it is usually quite close. Thus, our error metric favors decimation sequences which keep each coarse vertex close to the center of all fine vertices that contribute to its mass.
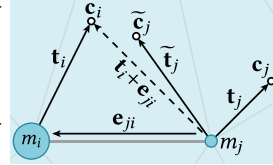
5.3 Surface Error Metric (Memory-Based)

To extend this scheme to surfaces, we must address the fact that tangent vectors from different tangent spaces cannot be added directly. In particular, we cannot re-express the error vector $\mathbf{t}_i$ at a neighboring vertex j by simply adding the edge vector $\mathbf{e}_{ji}$. If $\mathbf{c} := \exp_i(\mathbf{t}_i)$ is the center of mass encoded by $\mathbf{t}_i$, then ideally we would just compute the vector $\log_j(\mathbf{c})$ pointing from j to $\mathbf{c}$ (see inset). Although there are algorithms for computing the log map (*e.g.*, [Sharp et al. 2019b, §8.2] and [Sharp and Crane 2020b, §6.5]), they are far too expensive to apply for each vertex removal. Instead, we approximate this vector by parallel transporting $\mathbf{t}_i$ from i to j (*à la* Sec. 3.2.2) and offsetting by $\mathbf{e}_{ji}$ as in 2D, yielding a new vector
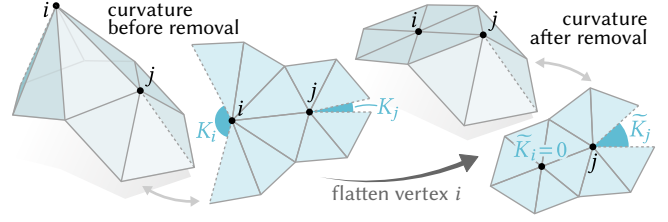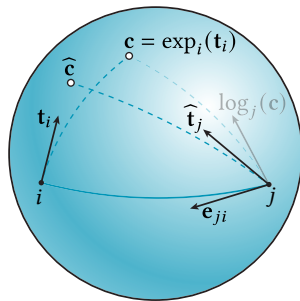

Fig. 10. Flattening a vertex i changes the angle sums Θ at neighboring vertices j, effectively redistributing the discrete curvature $K = 2\pi - \Theta$. We use the change in curvature from K to $\widetilde{K}$ to guide simplification.

$\widehat{\mathbf{t}}_j := \mathbf{R}_{ij} \mathbf{t}_i + \mathbf{e}_{ji}$. The final error vector $\widetilde{\mathbf{t}}_j$ stored at j is then given by a weighted average, just as in Sec. 5.2:

$$\widetilde{\mathbf{t}}_j = \frac{\alpha_{ij} m_i (\mathbf{R}_{ij} \mathbf{t}_i + \mathbf{e}_{ji}) + m_j \mathbf{t}_j}{\alpha_{ij} m_i + m_j}. \quad (8)$$

The mass is updated as in Eq. (5), and the overall cost is again given by Eq. (6). Note, then, that on curved surfaces the vector $\widetilde{\mathbf{t}}_i$ merely approximates the center of mass of the fine mass distribution corresponding to coarse vertex i. Yet since this approximation is reasonably accurate and cheap to compute, it provides an efficient error metric akin to QEM.

5.4 Intrinsic Curvature Error Metric

Due to the Gauss-Bonnet theorem, flattening a vertex i (*à la* Sec. 4.1) conservatively redistributes curvature to neighboring vertices j, making curvature a natural "mass" distribution to guide simplification. A challenge here is that the old and new curvatures K and $\widetilde{K}$ are not in general positive quantities. One possibility might be to use a transport cost for signed measures such as [Mainini 2012], but doing so would require us to solve a small optimal transport problem for each vertex removal. We instead adopt a cheap alternative. In particular, we define convex weights

$$\alpha_{ij} := \frac{|\widetilde{K}_j - K_j|}{\sum_{l \in \mathcal{N}_i} |\widetilde{K}_l - K_l|}. \quad (9)$$

For boundary vertices we use the same formula, but replace Gaussian curvature K with geodesic curvature κ. If vertex i is already flat prior to removal, then there is no change in curvature and we simply distribute mass equally to all neighbors. We then split the initial fine curvature function K (or κ) into two positive mass functions $K_i^+ := \max(K_i, 0)$ and $K_i^- := -\min(K_i, 0)$. Each of these quantities is tracked throughout simplification exactly like m_i in Sec. 5.3, using two separate vectors $\mathbf{t}_i^+$ and $\mathbf{t}_i^-$ (*resp.*), and weights α from Eq. (9). The overall error, which defines the ICE metric, is then the sum of the errors in the two curvature functions (*à la* Eq. (6)). Note that if a vertex i cannot be flattened or removed, we assign it an infinite cost (which may later get updated to a finite value when its neighbors are removed—see Alg. 1).

5.4.1 Auxiliary Data. Similar to [Garland and Heckbert 1998], other quantities at vertices (areas, colors, *etc.*) can be used to drive simplification in an analogous fashion: each signed quantity is split into two positive mass functions, and a list of all "channels" $m^0, \ldots, m^k$

is tracked along with associated tangent vectors $\mathbf{t}^0, \ldots, \mathbf{t}^k$. The cost is then given by

$$C_i = \sum_{j \in \mathcal{N}_i} \sum_{p=0}^{k} w^p m_i^p \|\mathbf{t}_j^p\|,$$

where a choice of weights $w^1, \ldots, w^k \in \mathbb{R}_{\geq 0}$ puts an emphasis on different features. For instance, Fig. 11 shows the impact of different weightings on curvature versus area.

6 SIMPLIFICATION

We now have all the ingredients to perform intrinsic simplification. Just as in QEM, we first initialize a priority queue by evaluating the ICE metric at all vertices (Sec. 6.1), then greedily remove the lowest-cost vertex from this queue until we reach a target vertex count n (Sec. 6.2), or until no more vertices can be removed. Alg. 1 provides pseudocode; a full reference implementation can be found at https://github.com/HTDerekLiu/intrinsic-simplification.

6.1 Initialization

For each vertex $i \in V$ we compute the initial masses m_i (e.g., the curvature functions K^+ and K^- from Sec. 5.4) and an initial error vector $\mathbf{t}_i = 0$. To compute the cost of removing i we perform a tentative flattening à la Sec. 4.1 and use the resulting weights α_{ij} from Eq. (9) to evaluate the cost C_i via the second sum in Eq. (6). If flattening yields invalid edge lengths, or vertex i cannot be flipped to degree 3 (or 2 on the boundary), we let $C_i = \infty$. After evaluating the cost function, we "undo" the tentative removal, i.e., we restore the previous connectivity and revert any changes to edge lengths.

6.2 Coarsening

At each iteration, we pick the vertex i with the minimum cost C_i from our priority queue. If $C_i = \infty$, then no more vertices can be removed and we terminate. Otherwise, we apply the removal procedure from Sec. 4. The resulting weights α_{ij} (Eq. (9)) are used to compute new masses $\widetilde{m}_j$ (Eq. (5)) and updated transport vectors $\widetilde{\mathbf{t}}_j$ (Eq. (8)) for each neighbor $j \in \mathcal{N}$. We then flip the mesh back to intrinsic Delaunay à la Sec. 3.3.2—note that to initialize the greedy flipping algorithm we need only enqueue edges in $\mathcal{N}_i$, since the mesh was already Delaunay prior to removing vertex i. Finally, we must also update the priority queue with new costs C_j by tentatively flattening each neighbor j, and evaluating the first sum in Eq. (6) (this time over neighbors $k \in \mathcal{N}_j$). Here, finite costs may become infinite (or vice versa), since vertices that were previously removable may no longer be removable.

6.3 Intrinsic Retriangulation

Though our overall goal is to coarsen the input, carefully *adding* vertices to the mesh via *intrinsic Delaunay refinement* [Sharp et al. 2019a, Section 4.2] can be quite valuable in two distinct ways. On typical models, this pre- or post-processing adds only a fraction of a second to overall execution time [Sharp et al. 2019a, Section 6].

Pre-refinement. As stated, our coarsening algorithm (Alg. 1) can produce only meshes whose vertices V are a subset of the input vertices $\widetilde{V}$. However, we can often obtain a smaller or more accurate

---

Algorithm 1: Intrinsic Coarsening

Input : M, ℓ, n ▷ mesh, edge lengths, target vertex count
Output: $\widetilde{M}, \widetilde{\ell}$ ▷ coarsened mesh & edge lengths

1. ▷ initialization
2. $Q \leftarrow \text{EMPTYPRIORITYQUEUE}()$
3. $M, \ell \leftarrow \text{FLIPTODELAUNAY}(M, \ell)$ ▷ Sec. 3.3
4. **foreach** *vertex* $i \in \widetilde{V}$ **do**
5. $(m_i, \mathbf{t}_i) \leftarrow ((K_i^-, K_i^+), 0)$ ▷ initial mass & error
      $c \leftarrow \text{INTRINSICCURVATUREERROR}(M, \ell, i)$ ▷ Sec. 5.4
6. $\text{ENQUEUE}(Q, i, c)$
7. ▷ coarsening
8. **while** $\text{VERTEXCOUNT}(M) > n$ **and** $!\text{EMPTY}(Q)$ **do**
9. $i \leftarrow \text{POP}(Q)$ ▷ extract minimum-cost vertex
10. $\ell \leftarrow \text{FLATTEN}(M, \ell, i)$ ▷ Sec. 4.1
11. $M, \ell, m, \mathbf{t} \leftarrow \text{REMOVEVERTEX}(M, \ell, i)$ ▷ Sec. 4.2, 5.3
12. $M, \ell \leftarrow \text{FLIPTODELAUNAY}(M, \ell, i)$
13.   **foreach** *vertex* $j \in \mathcal{N}_i$ **do**
14. $c \leftarrow \text{INTRINSICCURVATUREERROR}(M, \ell, j)$
15. $Q \leftarrow \text{UPDATEPRIORITY}(Q, j, c)$
16. **return** (M, ℓ)

---

representation of the original metric by first inserting a larger set of candidate vertices, prior to simplification. For instance, in Fig. 12 we run Delaunay refinement on the input until all corner angles θ_{jk}^i are no smaller than 25°. As a result, it becomes much easier to construct high-quality coarse triangles in regions with few input vertices (e.g., on the rocket model). Fig. 13 shows how this strategy can even improve quality on coarse input models, where we do not seek to reduce the vertex count.

Post-refinement. Since coarsening maintains an intrinsic Delaunay triangulation, the final mesh is already guaranteed to exhibit properties valuable for applications—such as positive edge weights for the discrete Laplacian [Bobenko and Springborn 2007]. As an optional post-process, we can also provide hard guarantees on element *quality*: as recently proven by Gillespie et al. [2021a], intrinsic Delaunay refinement yields a triangulation where the smallest corner angle θ_i^{jk} is no smaller than 30° (hence no greater than 120°), so long as (i) M is closed and (ii) $\Theta_i \geq 60°$ for all input vertices $i \in V$. In practical terms, this guarantee ensures that even very low-quality input meshes can be used successfully in numerical algorithms such as those explored in Sec. 8. This feature is unique to the intrinsic setting: extrinsic surface meshing algorithms like *restricted Delaunay refinement* provide guarantees on global geometry and topology, but not on element quality [Cheng et al. 2012, Chapter 13], nor even positive weights for the Laplacian. The only danger is that the final triangulation is no longer guaranteed to be coarser than the input mesh, though in practice this situation is unlikely to occur unless the input is already quite coarse.

6.4 Numerics

As in any mesh processing algorithm, near-degenerate triangles (whether in the input or constructed during simplification) can

% curvature/% area

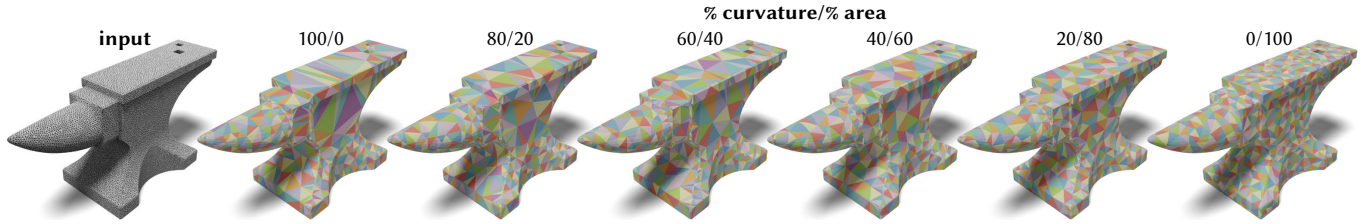input   100/0   80/20   60/40   40/60   20/80   0/100

Fig. 11. We can mix and match different quantities to guide coarsening. Here, for instance, strongly weighting Gaussian curvature emphasizes preservation of intrinsic geometry, whereas strongly weighting area prioritizes uniform triangle size.
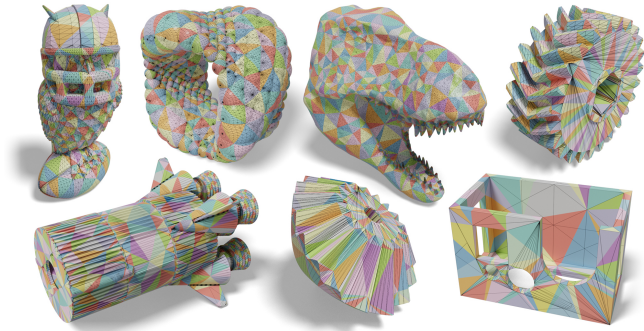

Fig. 12. Adding vertices to the mesh prior to coarsening provides more options for simplification and remeshing, since coarse vertices $\widetilde{V}$ no longer need be a subset of fine vertices V.
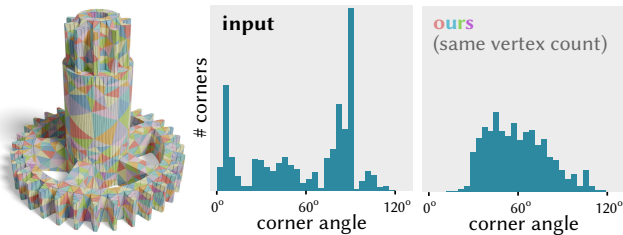

Fig. 13. Even for an identical vertex count, refining then coarsening yields higher quality elements, as quantified by histograms of corner angles θ^i_{jk}.

cause numerical issues due to floating point error. We hence follow best practices wherever possible [Shewchuk 1999], *e.g.*, to determine if a point p is inside a triangle during pointwise mapping (Sec. 7.1) we compute the sign of $\det(p - x_i, p - x_j)$ for each triangle edge ij, rather than directly computing barycentric coordinates (thereby avoiding division by area).

7 MAPPING AND PROLONGATION

For some tasks (say, computing Laplacian eigenvalues) the coarsened triangulation can be used directly; more broadly we need some way of evaluating correspondence between the coarse and fine mesh. Here we consider two basic viewpoints: correspondence of *points* (Sec. 7.1) and correspondence of *functions* (Sec. 7.2).

7.1 Pointwise Mapping

To map any point p on the fine mesh to a point $\widetilde{p}$ on the coarse mesh, we track its barycentric coordinates through local coarsening operations (namely: edge flips, vertex flattenings, and vertex removals). This map is trivially bijective, since at each step we simply re-write the given barycentric coordinates with respect to a different triangulation of the same planar region. The only way to violate bijectivity would be to perform a non-bijective vertex flattening—which we explicitly forbid (see Sec. 4.1). In the applications we consider (Sec. 8), all points p that must be tracked are known ahead of time, and can be tracked during simplification. To evaluate this map on demand, one could record the list of local operations, and "re-play" these operations for each new query point, as in [Liu et al. 2020].

Edge flips. To track a point p through an intrinsic flip of edge ij, we unfold the two triangles ijk, jil into the plane (*e.g.*, using formulas from Sharp et al. [2021, Section 2.3.7]), and compute the barycentric coordinates of p in the new triangle (see inset).
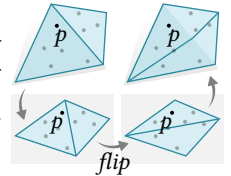

Vertex flattening. We must also compute new barycentric coordinates $\widetilde{b}$ after each vertex flattening (Sec. 4.1). Here we use the projective interpolation scheme of Springborn et al. [2008, §3.4]. Since edge lengths satisfy Eq. (3), this scheme defines a continuous (C^0) bijective map. Let b_i, b_j, b_k be barycentric coordinates for a point in face ijk, and let u_i be the scale factor at i. Then

$$(\widetilde{b}_i, \widetilde{b}_j, \widetilde{b}_k) = \frac{(e^{u_i} b_i, b_j, b_k)}{e^{u_i} b_i + b_j + b_k}, \tag{10}$$

where the denominator ensures our updated values still sum to 1.

Vertex removal. Once vertex i is flattened and flipped to degree three, its neighborhood can be laid out in the plane without distortion (see inset). Here we apply standard formulas to compute barycentric coordinates for vertex i in the new triangle, along with coordinates for any points located in the three removed triangles.
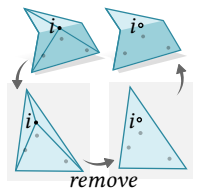
7.2 Prolongation

Algorithms such as multigrid (Sec. 8.7) often require not only pointwise correspondences, but also *prolongation operators* which transfer functions from a coarse mesh to a finer one. We define a prolongation operator via an approach similar to Lee et al. [1998] and
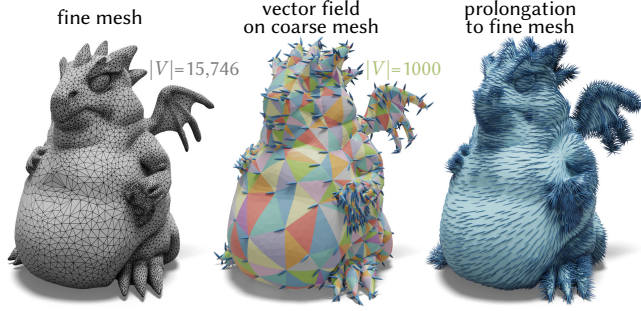
Fig. 14. We significantly reduce the cost of computing smooth vector fields on a fine mesh *(left)* by solving on a coarse mesh *(center)* and applying vector prolongation *(right)*.

Liu et al. [2021]. In particular, we track the barycentric coordinates of all fine vertices *à la* Sec. 7.1. A function on the coarse mesh is then mapped to the fine mesh via barycentric linear interpolation. Explicitly, suppose each fine vertex $i \in V$ has barycentric coordinates b_1, b_2, b_3 in coarse triangle $n_1 n_2 n_3 \in \widetilde{F}$. We then build a sparse matrix $\mathbf{P} \in \mathbb{R}^{|V| \times |\widetilde{V}|}$ where row i has three nonzeros $\mathbf{P}_{i,n_j} = b_{n_j}$, for $j = 1, 2, 3$. Prolongation then amounts to a matrix-vector product $f = \mathbf{P}\widetilde{f}$, where $\widetilde{f} \in \mathbb{R}^{|\widetilde{V}|}$ is a function on the coarse mesh.

7.3 Vector Field Prolongation

We can also transfer vector fields from coarse to fine (Fig. 14). This process has two steps (detailed below): first interpolate vectors over the coarse mesh, then sample the interpolated vector field onto the vertices of the fine mesh. Both steps are again represented by a single prolongation matrix $\mathbf{P}^{\text{vec}} \in \mathbb{C}^{|V| \times |\widetilde{V}|}$, this time with complex entries. However, since we store tangent vectors in a different normalized coordinate system at each vertex (Sec. 3.2.2), we must compute unnormalized vectors before performing interpolation. In particular, let $\mathbf{u}_i^0 \in \mathbb{C}$ be the angle-normalized vectors stored at vertices, and let $\phi_i := \arg(\mathbf{u}_i^0)$ be the corresponding angles. Then $\mathbf{u}_i := \|\mathbf{u}_i^0\| e^{\iota \Theta_i \phi_i / 2\pi}$ are the corresponding unnormalized vectors, and prolongation amounts to a matrix-vector multiply $\mathbf{P}^{\text{vec}}\mathbf{u}$. Note that this same prolongation scheme can be applied as-is to symmetric direction fields (line fields, cross fields, *etc.*), via the complex encoding introduced by Knöppel et al. [2013].

7.3.1 Coarse Interpolation.
In each triangle ijk, we adopt a coordinate system where oriented edge ij points in the $\phi = 0$ direction. Let $\mathbf{w}_{ij} := (\mathbf{e}_{ij} / \ell_{ij})^{\Theta_i / 2\pi}$ be the unit vectors along each oriented edge ij, in unnormalized coordinates at i, and let

$$\beta_{ij} := 0, \quad \beta_{jk} := (\pi - \theta_{ki}^j), \quad \text{and} \quad \beta_{ki} := (\pi - \theta_{ki}^j) + (\pi - \theta_{ij}^k)$$

be the angles of these unit vectors with respect to the coordinate system of the triangle. Then $\mathbf{z}_i := e^{\iota \beta_{ij}} / \mathbf{w}_{ij}$ is a rotation taking $\mathbf{u}_i$ to the corresponding vector in the triangle's coordinate system, and we can express the interpolated vector at any point $\mathbf{p}$ with barycentric
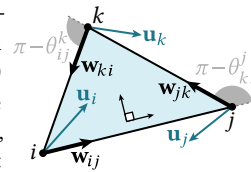
coordinates $b(\mathbf{p})$ as

$$\mathbf{u}(\mathbf{p}) := b_i(\mathbf{p})\mathbf{z}_i\mathbf{u}_i + b_j(\mathbf{p})\mathbf{z}_j\mathbf{u}_j + b_k(\mathbf{p})\mathbf{z}_k\mathbf{u}_k.$$

Note that the resulting field is not continuous across edges, but is sufficiently regular for prolongation; if desired, better continuity can be achieved via the scheme of Liu et al. [2016, Section 4.3].

7.3.2 Fine Sampling.
Consider a fine vertex $i \in V$ mapped to a point $\mathbf{x}_i$ in coarse triangle $n_1 n_2 n_3 \in \widetilde{F}$ (expressed in a 2D layout, via the tracked barycentric coordinates). To map the interpolated vector $\mathbf{u}(\mathbf{x}_i)$ back to the vertex coordinate system, we must then compute a change of coordinates from triangle $n_1 n_2 n_3$ to the tangent space $\mathcal{T}_i M$. Since we change the geometry via conformal flattening (Sec. 4.1), this change of coordinates is well-described by measuring the rotation and scaling of a single tangent vector. In particular, for any $j \in \mathcal{N}_i$ such that $\mathbf{x}_j$ is contained in the same triangle, we let $\mathbf{w}_{ij} := \mathbf{e}_{ij} / \|\mathbf{e}_{ij}\|$ be the unit vector along the fine edge, and compute the corresponding unit vector $\widetilde{\mathbf{w}}_{ij} := (\mathbf{p}_j - \mathbf{p}_i) / \|\mathbf{p}_j - \mathbf{p}_i\|$ on the coarse triangle (approximating the tangent map). The rotation and scaling between coordinate systems is then captured by the complex number $\psi_i := \mathbf{w}_{ij} / \widetilde{\mathbf{w}}_{ij}$, and the final interpolated value in the normalized coordinate system at i is given by $\psi_i \mathbf{u}_i(\mathbf{x}_i)$. In the rare case where no neighboring $\mathbf{x}_j$ sits in the same triangle, we simply take the average of known interpolated values. Overall, then, row i of the vector prolongation matrix $\mathbf{P}^{\text{vec}}$ has three nonzero entries $b_j \psi_i \mathbf{z}_j$, corresponding to the three columns n_j, $j = 1, 2, 3$.
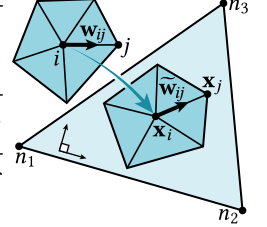
8 EVALUATION & RESULTS

Here we evaluate the performance, robustness, and quality of our method (Sec. 8.2 and 8.3), compare it to extrinsic alternatives (Sec. 8.4), and explore its effectiveness in the context of several fundamental algorithms (Sec. 8.5, 8.6, and 8.7). We also describe the strategy used to visualize results throughout the paper (Sec. 8.1). Note that all experiments were run on a 4.1GHz Intel i7-8750H with 16GB RAM.

8.1 Visualization

Traditionally, intrinsic triangulations are visualized via a *common subdivision* [Fisher et al. 2006; Gillespie et al. 2021a; Sharp et al. 2019a], *i.e.*, the input mesh is split along geodesic arcs corresponding to intrinsic edges. However, since coarsening does not exactly preserve intrinsic geometry, edges may no longer be geodesics. We instead use a texture mapping approach (Fig. 15). Given initial texture coordinates (computed via [Sawhney and Crane 2018]), we compute barycentric coordinates for each texel covered by a fine triangle. These coordinates are then tracked through the coarsening process, *à la* Sec. 7.1. Following Sharp et al. [2019a] we assign a greedy coloring to the coarse triangles; texels adopt the color of their associated triangle. This visualization may not exactly depict coarse lengths or areas, but faithfully represents the bijective map. A possible alternative is to construct an explicit *topological* subdivision, where edges need not be geodesics [Schmidt et al. 2020; Takayama 2022], but which may enable more sophisticated attribute transfer [Gillespie et al. 2021a, §4.3].
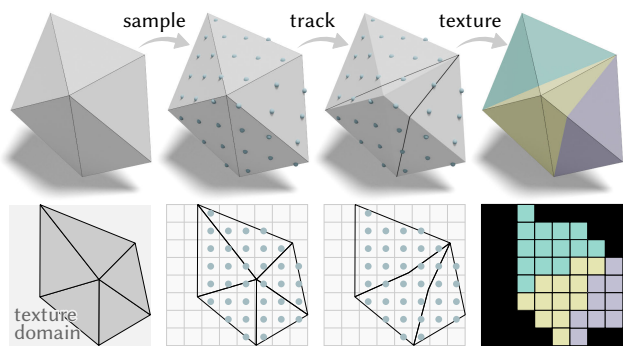
Fig. 15. To visualize a coarse intrinsic triangulation on a mesh with texture coordinates *(left)*, we track the barycentric coordinates of each texel *(center left)* through the simplification process *(center right)*, and use coarse triangle IDs to color the final texels *(right)*.

8.2 Benchmark

We evaluated our method on about 6k manifold meshes from the *Thingi10k* dataset [Zhou and Jacobson 2016]. For robustness experiments we preprocess meshes via intrinsic Delaunay refinement [Sharp et al. 2019a, §4.2] with a lower angle bound of $25°$, providing more candidate locations for coarse vertices. Since some meshes contain thousands of connected components, we normalize input/output vertex counts by the number of components.

Performance. Fig. 16 plots the total cost of our method, including decimation, maintaining a bijective map, and constructing the prolongation operator $\mathbf{P}$. Since each vertex removal is an $O(1)$ operation, we achieve near-linear scaling with respect to input size, though for very large meshes the $O(n \log n)$ cost of maintaining a priority queue will ultimately dominate. In absolute terms, our method decimates about 10,000 vertices per second.

Robustness. Our method fails only if no remaining vertex can be removed, *i.e.*, if (i) flattening would violate the triangle inequality or (ii) flipping to degree-3 is not possible (Sec. 4). We hence quantify robustness by coarsening as much as possible, then measuring the ratio $|\widetilde{V}|/|V|$. In Fig. 17 we successfully reduce 98% and 84% of models down to 10% and 1% (*resp.*) of the input resolution, prior to Delaunay refinement. Large ratios occur only for high-genus models that cannot be significantly coarsened without modifying global topology. On about 0.1% of meshes vertex removal failed due to floating-point error; *integer coordinates* [Gillespie et al. 2021a] may help to further improve robustness.

8.3 Measuring Distortion

To evaluate our method, we measure the parametric distortion of the bijective map $\varphi : M \to \widetilde{M}$ between the fine and coarse mesh—which is distinct from the ICE metric used for coarsening. For each fine edge ij we first find an approximation $\hat{\ell}_{ij}$ of its length under φ. Since all fine edges are minimal geodesics in the extrinsic mesh, we assume this property is preserved
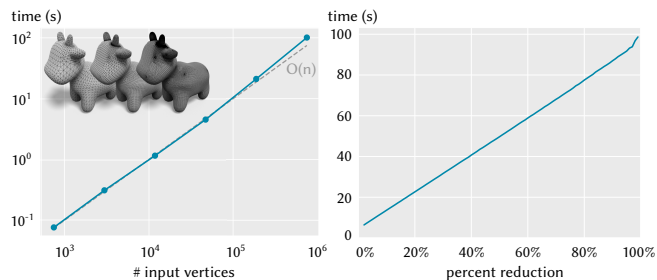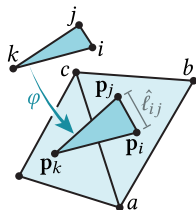

Fig. 16. Since we use only greedy local operations, total cost is roughly linear in both mesh size and percent reduction (including the cost to build $\mathbf{P}$). *Left:* increasingly fine subdivisions are coarsened to 1% of their initial size. *Right:* a subdivision with 750k vertices is coarsened to various resolutions.
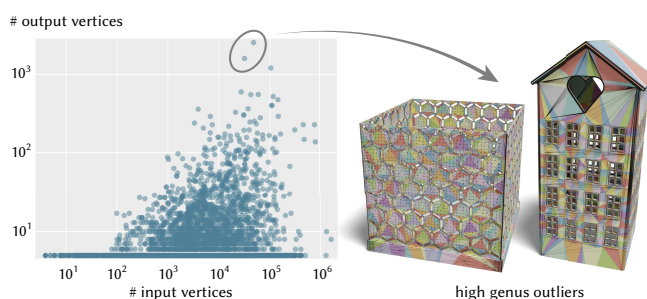

Fig. 17. Our method can coarsen even difficult meshes down to a small number of vertices. Here, 98% of *Thingi10k* meshes are coarsened to less than 10% of their input size *(left)*. The exception are very high-genus meshes, which cannot be coarsened without changing the global topology *(right)*.

under φ and compute the minimal geodesic distance between $\mathbf{p}_i$ and $\mathbf{p}_j$ If both endpoints sit in the same coarse triangle *abc*, we can simply measure the distance between image points $\mathbf{p}_i := \varphi(i)$ and $\mathbf{p}_j := \varphi(j)$ in a local layout of *abc*, computed via barycentric coordinates. Otherwise, we compute geodesic distance using the method of Mitchell et al. [1987]. Finally, for each fine triangle ijk we use lengths $\hat{\ell}_{ij}, \hat{\ell}_{jk}, \hat{\ell}_{ki}$ to construct representative vertex positions $\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k \in \mathbb{R}^2$, via Sharp et al. [2021, §2.3.7]. Distortion relative to input vertices $\mathbf{p}_i^0, \mathbf{p}_j^0, \mathbf{p}_k^0$ can then be quantified using any standard per-triangle measure—we use the *anisotropic distortion* and *area distortion* as defined by Khodakovsky et al. [2003, §2].

8.4 Comparison with Extrinsic Methods

Relative to past methods, the flexibility gained by working in the larger space of intrinsic triangulations leads to smaller geometric distortion on meshes of equivalent size. For instance, in Fig. 18 we coarsen a 28k bunny mesh down to 200 vertices with both the method of Liu et al. [2021] and our method. Even on this highly regular geometry we observe a modest reduction of both area distortion and anisotropic distortion. For more difficult triangulations, or surfaces with lower intrinsic curvature (*e.g.*, Fig. 1), we observe more significant gains. As an extreme case, Fig. 19 coarsens a developable surface from [Verhoeven et al. 2022] via both QEM and ICE.
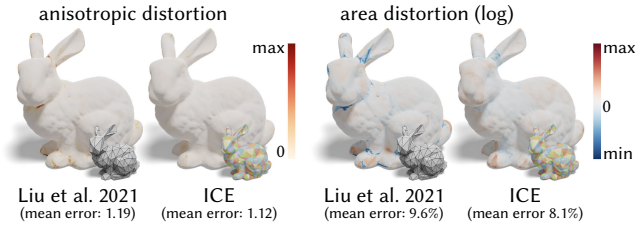
Fig. 18. Even on an extremely nice triangulation of a highly regular surface we see a reduction in distortion relative to past methods—owing to the much larger space of intrinsic triangulations.
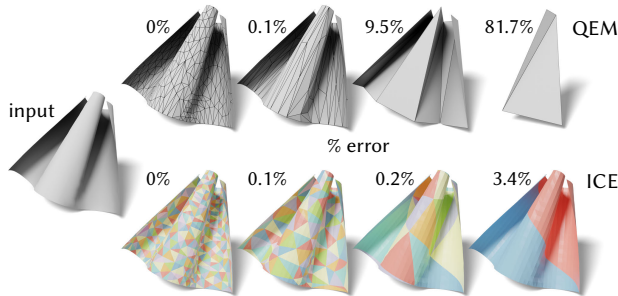

Fig. 19. On surfaces with small extrinsic curvature, such as this mesh from cloth simulation, we achieve dramatically lower error in surface area compared to extrinsic methods like QEM.
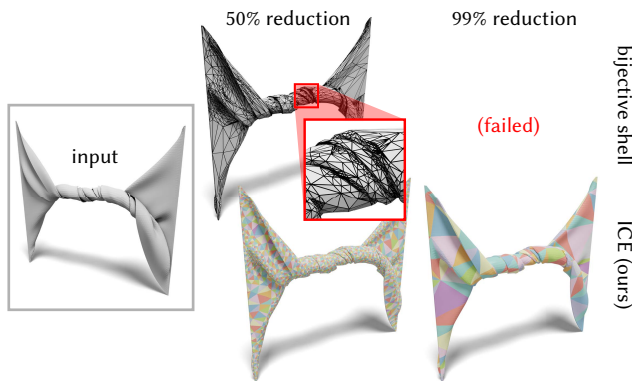

Fig. 20. Methods that rely on extrinsic information to construct a mapping during coarsening can fail in the presence of self-intersections. Here, the *bijective shell* method, which relies on extrinsic ray casting [Jiang et al. 2020, §3.2], fails to coarsen below 50% of the input size. In contrast, our intrinsic approach easily obtains an extremely coarse decimation.

Since coarse extrinsic edges are shortest paths in $\mathbb{R}^n$, they underestimate intrinsic distances (hence areas); in contrast, intrinsic edges are essentially embedded in the original surface, providing better approximation of the original geometry.
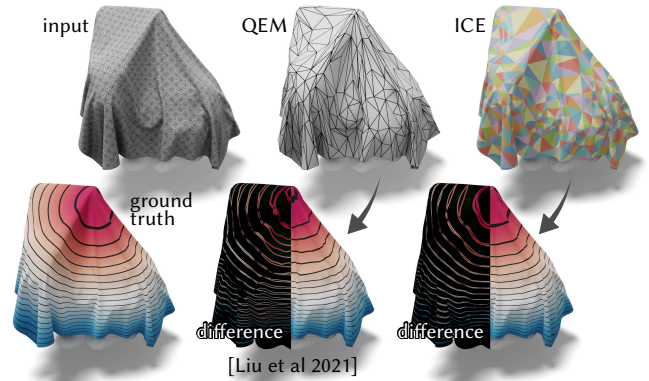

Fig. 21. For the same vertex budget as extrinsic methods like QEM, ICE provides more accurate solutions for basic problems like solving a Poisson equation—seen here via smoother isolines that better approximate the ground truth.
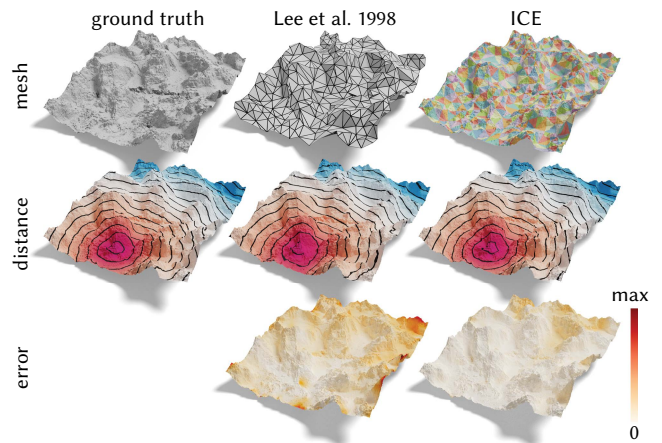

Fig. 22. Since geodesic distance is an intrinsic quantity, it is more accurately approximated via intrinsic coarsening—here providing a 4x reduction in relative error.

8.5 Geometric Algorithms

8.5.1 Partial Differential Equations. Better domain approximation in turn improves the quality of solutions computed on coarse meshes. For example, in Fig. 21 we coarsen a cloth simulation mesh down to 500 vertices with an extrinsic method ([Liu et al. 2021] using QEM simplification) and our intrinsic method. We then solve a Poisson problem on the coarse meshes and apply prolongation, yielding more accurate results in the intrinsic case.

8.5.2 Single-Source Geodesic Distance. Geodesic distance is an intrinsic quantity, making it a natural fit for intrinsic coarsening. In Fig. 22 we compare ICE to the extrinsic method of Lee et al. [1998] by measuring the difference between the exact distance on the fine input, and prolongated distances from the coarse meshes (both computed via [Mitchell et al. 1987]); here ICE achieves a roughly 4x
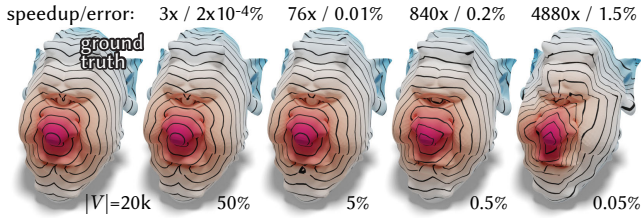
Fig. 23. Intrinsic coarsening offers an attractive approach to approximating single-source geodesic distance, here providing a three orders of magnitude speedup for a fraction of a percent relative error.
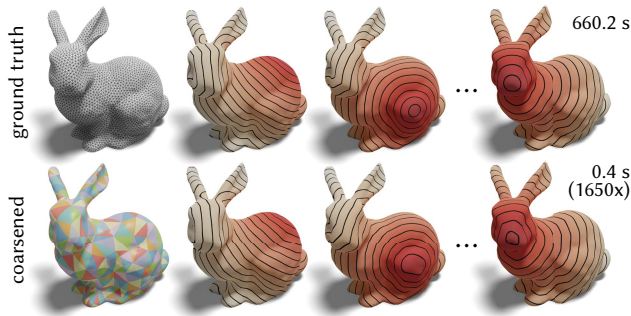

Fig. 24. For a mesh with 6k vertices we obtain an all-pairs geodesic distance matrix 1650x faster, while incurring only 1.4% relative error.
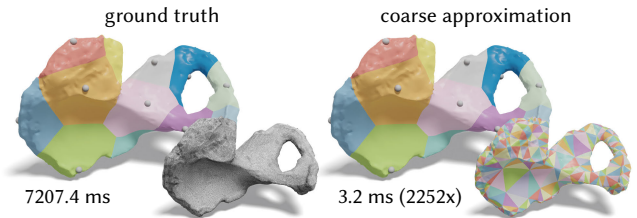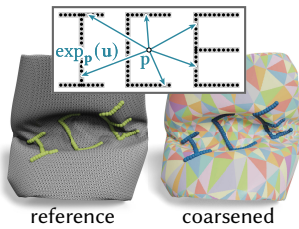

Fig. 25. Fast computation of geodesic distance in turn yields fast computation of other quantities—here we compute geodesic Voronoi diagrams three orders of magnitude faster (2252x) than on the original mesh, while misclassifying only 1% of fine vertices.
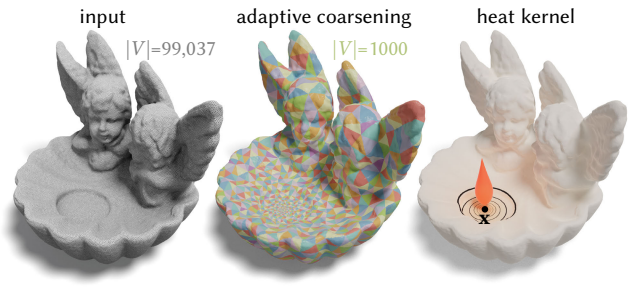

Fig. 26. By coarsening more aggressively away from the point **x**, we better resolve the exponential falloff in a heat kernel centered at **x**.

reduction in relative error. Fig. 23 illustrates the speed-accuracy trade off of using ICE, here reducing cost by three orders of magnitude while introducing only ∼1% relative approximation error.

8.5.3 All-Pairs Geodesic Distance. The benefits of an accurate intrinsic approximation become even more pronounced when approximating the dense matrix $\mathbf{D} \in \mathbb{R}^{|V| \times |V|}$ of all pairs of geodesic distances—a shape descriptor often used in correspondence and learning methods [Shamai and Kimmel 2017]. We can compute a low-rank approximation of $\mathbb{D}$ via

$$\widehat{\mathbf{D}} := \mathbf{P}\widetilde{\mathbf{D}}\mathbf{P}^\top,$$

where $\widetilde{\mathbf{D}}$ is the coarse all-pairs matrix (computed again via [Mitchell et al. 1987]). See for instance Fig. 24—here again we achieve several orders of magnitude speedup, with only 1.4% relative error.

8.5.4 Riemannian Computational Geometry. More broadly, standard geometric quantities computed on the coarse mesh provide excellent approximations of the fine solution. For instance, in Fig. 25 we use Mitchell et al. [1987] to compute a geodesic Voronoi diagram on the

reference   coarsened

coarse mesh, yielding a near-perfect approximation at a tiny fraction of the cost—such diagrams in turn provide the starting point for remeshing and other applications [Ye et al. 2019]. Likewise, we can dramatically reduce the cost of evaluating the discrete exponential

map over long distances (*à la* [Sharp et al. 2021, §2.4.2]), replacing many small steps through fine triangles with a small handful of ray-edge intersections, while arriving at nearly identical points (see inset). Other classic algorithms, such as *Steiner tree approximation* [Sharp et al. 2019a, §5.1] could likewise be accelerated by simply swapping out our coarse intrinsic mesh for the fine one.

8.6 Adaptive Coarsening

The local nature of our scheme makes it easy to adaptively coarsen (or preserve) the mesh according to various geometric criteria. Here we expand on the basic weighting strategy from Sec. 5.4.1.

8.6.1 Spatial Adaptivity. We can emphasize a region of interest by prescribing spatially-varying masses m_i on the fine vertices $i \in V$; regions where m_i is small are then coarsened less aggressively. For instance, Fig. 26 uses masses $m_i := 1/d_{\mathbf{x}}^2(i)$ (where $d_{\mathbf{x}}$ is the distance to x) to adapt the coarse mesh to the *heat kernel* centered at **x**.

8.6.2 Anisotropic Coarsening. Alternatively, we can emphasize important directions by non-uniformly scaling the input edge lengths along directions of interest—such meshes are better suited to, *e.g.*, solving PDEs with anisotropic coefficients. More explicitly, given vectors $\mathbf{u}_i$ at vertices, we scale each length ℓ_{ij} by a factor $(1 - \tau) + \frac{\tau}{2}((\mathbf{u}_i \cdot \widehat{\mathbf{e}}_{ij})^2 + (\mathbf{u}_j \cdot \widehat{\mathbf{e}}_{ji})^2)$, where $\widehat{\mathbf{e}}_{ij} := \mathbf{e}_{ij}/\|\mathbf{e}_{ij}\| \in \mathcal{T}_i M$ is the unit tangent vector along edge ij, and the parameter $\tau \in [0, 1]$ controls the strength of anisotropy. For instance, in Fig. 27 the vectors $\mathbf{u}_i$ are the (max or min) principal curvature directions, computed via the method of Panozzo et al. [2010]. A challenge here, noted by Campen
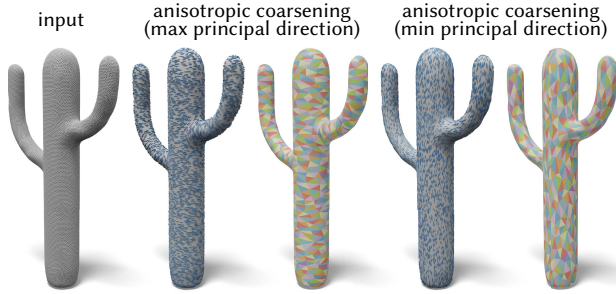
Fig. 27. Anisotropic coarsening can be achieved by applying nonuniform scaling to input edge lengths, here using principal curvature directions.

Fig. 28. Fixing vertices during coarsening enables us to exactly preserve boundary conditions and constraint curves when solving PDEs on surfaces.

et al. [2013, Section 4.1], is that a simple rescaling can violate the triangle inequality, which in practice limits the strength of anisotropy. How to robustly express anisotropic changes to the discrete metric is an interesting question for future work.

8.6.3 Boundary Preservation. Finally, we can fix a user-specified set of vertices in order to, *e.g.*, exactly preserve boundary conditions for a PDE. For instance, in Fig. 28 we fix vertices that encode Dirichlet boundary conditions for a Poisson problem, accurately preserving both the boundary data and the constraint curve.

8.7 Intrinsic Mesh Hierarchies

Mesh hierarchies are used throughout visual, geometric, and scientific computing to accelerate solvers via, *e.g.*, Cholesky preconditioners [Chen et al. 2021], multigrid methods [Aksoylu et al. 2005; Liu et al. 2021], or GPU acceleration Mahmoud et al. [2021] further demonstrate the possibility of parallelizing geometry processing with mesh hierarchies. Despite the fact that many of these applications need only intrinsic operators, these methods do not take full advantage of the intrinsic setting. Our method can be used to construct an *intrinsic* mesh hierarchy by simplifying the input to a sequence of progressively coarser meshes—Fig. 29 shows one such example. Using the prolongation matrices **P** between consecutive levels, we can then build a surface multigrid method *à la* Liu et al. [2021]. This intrinsic multigrid method is well-suited to algorithms expressed in terms of discrete differential operators—even if they involve extrinsic data—such as the modified mean curvature flow of Kazhdan et al. [2012] (Fig. 30). In general, the additional flexibility of working with intrinsic triangulations yields greater robustness than previous, extrinsic methods, especially on low-quality input. For instance, in Fig. 31 the method of Liu et al. [2021] fails to build a valid mesh hierarchy; extrinsically refining the input via [Cignoni

Fig. 29. We can build an intrinsic multigrid mesh hierarchy via repeated coarsening, here reducing the vertex count by a factor 1/4 at each level.
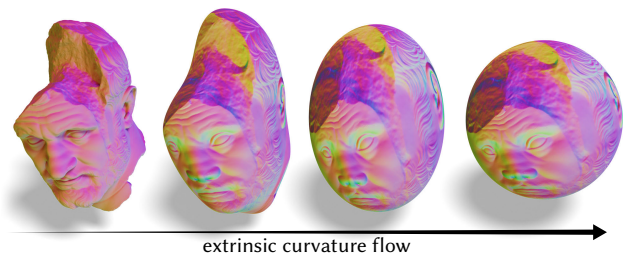

Fig. 30. Here we use our intrinsic multigrid scheme to accelerate the extrinsic curvature flow of Kazhdan et al. [2012], achieving a 20x speedup.

et al. 2008] preserves the geometry, but the solver now fails due to low-quality triangles; global extrinsic remeshing enables the solver to succeed, but yields a solution on a different domain than the input. Our intrinsic approach easily succeeds on this example, since (as noted in Fig. 1) it does not have to simultaneously juggle mesh quality and element quality, and can rely on hard guarantees about triangulation quality, as discussed in Sec. 6.3.

9 LIMITATIONS & FUTURE WORK

In order to keep computation cheap and local, the ICE metric makes three basic approximations. First, we approximate the mass distribution of all "ancestor" vertices by concentrating their sum at their center of mass (Sec. 5.1). This approximation is very much in the spirit of QEM, which approximates the extrinsic distribution of all ancestors via a single quadratic function. Second, when re-assigning mass to a neighboring vertex, we approximate computation of the logarithmic map via parallel transport along an edge (Sec. 5.3). This approximation has no analogue in QEM, and in the future it may be worth considering other approximations—or at least performing an ablation (relative to the exact log map) to better understand the impact of this approximation on overall performance. Third, we approximate the cost of redistributing curvature from a removed vertex i to its neighbors j via the change in curvature at each vertex j (Eq. (9)). As
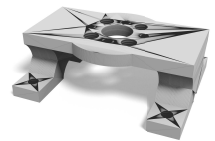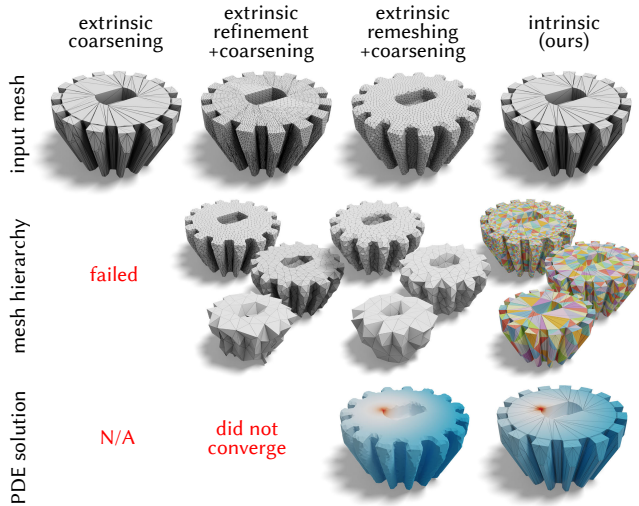
Fig. 31. Obtaining a reliable surface mesh hierarchy for geometric multigrid can be surprisingly challenging—often requiring global remeshing of the geometry to obtain acceptable results. Our intrinsic scheme builds on established guarantees, ensuring success even on extremely low-quality inputs.

noted in Sec. 5.4 a more principled (but also more expensive) alternative might be to directly compute the transport cost for a signed measure, which still involves only the local vertex neighborhood.

Other aspects of the method could also be improved or generalized. One significant question, noted in Sec. 7.1, is how to more easily compute point correspondences without replaying (or reversing) coarsening operations. Techniques such as integer coordinates [Gillespie et al. 2021a] might improve floating point robustness on extremely poor-quality inputs (inset). Likewise, *Ptolemy edge flips* [Gillespie et al. 2021b] might further improve robustness to rare violation of the triangle inequality during flattening. Extending prolongation to *discrete differential forms* [Desbrun et al. 2006] could help accelerate a wider variety of geometry processing tasks [Crane et al. 2013]. Similarly, it may prove valuable to consider how to best perform simplification in a dynamic context—especially since many natural deformations are near-isometric. Finally, a more careful treatment of memory management and parallel data structures might help bring performance closer to highly-optimized libraries for extrinsic simplification [Kapoulkine 2019].

ACKNOWLEDGMENTS

REFERENCES

Burak Aksoylu, Andrei Khodakovsky, and Peter Schröder. 2005. Multilevel Solvers for Unstructured Surface Meshes. *SIAM J. Sci. Comput.* 26, 4 (2005).

Marc Alexa and Jan Eric Kyprianidis. 2015. Error diffusion on meshes. *Computers & Graphics* 46 (2015).

Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. 2003. Anisotropic polygonal remeshing. In *ACM SIGGRAPH 2003 Papers*.

Pierre Alliez, Éric Colin de Verdière, Olivier Devillers, and Martin Isenburg. 2005. Centroidal Voronoi diagrams for isotropic surface remeshing. *Graph. Mod.* (2005).

Pierre Alliez, Mark Meyer, and Mathieu Desbrun. 2002. Interactive geometry remeshing. *ACM Trans. Graph.* 21, 3 (2002).

Alexander I Bobenko and Boris A Springborn. 2007. A discrete Laplace–Beltrami operator for simplicial surfaces. *Discrete & Computational Geometry* 38, 4 (2007).

David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. 2013a. Integer-grid maps for reliable quad meshing. *ACM Trans. Graph.* 32, 4 (2013).

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2013b. Quad-mesh generation and processing: A survey. In *Computer graphics forum*, Vol. 32. 51–76.

Mario Botsch and Leif Kobbelt. 2004. A Remeshing Approach to Multiresolution Modeling. In *Eurographics Symp. Geom. Proc.*, Vol. 71.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon Mesh Processing*. CRC press.

Marcel Campen, Martin Heistermann, and Leif Kobbelt. 2013. Practical anisotropic geodesy. In *Computer Graphics Forum*, Vol. 32.

Ryan Capouellez and Denis Zorin. 2022. Metric Optimization in Penner Coordinates. *arXiv preprint arXiv:2206.11456* (2022).

Jiong Chen, Florian Schäfer, Jin Huang, and Mathieu Desbrun. 2021. Multiscale cholesky preconditioning for ill-conditioned problems. *ACM Trans. Graph.* 40, 4 (2021).

Siu-Wing Cheng, Tamal K Dey, and Jonathan Shewchuk. 2012. *Delaunay mesh generation*. CRC Press.

Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. 2008. MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference 2008, Salerno, Italy, 2008*. Eurographics.

Jonathan D. Cohen, Dinesh Manocha, and Marc Olano. 1997. Simplifying polygonal models using successive mappings. In *Proc. IEEE Vis. Conf.*

David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2004. Variational shape approximation. *ACM Trans. Graph.* 23, 3 (2004).

Keenan Crane, Fernando De Goes, Mathieu Desbrun, and Peter Schröder. 2013. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH Courses*.

Fernando de Goes, Mathieu Desbrun, Mark Meyer, and Tony DeRose. 2016. Subdivision exterior calculus for geometry processing. *ACM Trans. Graph.* 35, 4 (2016).

Mathieu Desbrun, Eva Kanso, and Yiying Tong. 2006. Discrete differential forms for computational modeling. In *ACM SIGGRAPH 2006 Courses*.

Hans-Christian Ebke, Patrick Schmidt, Marcel Campen, and Leif Kobbelt. 2016. Interactively controlled quad remeshing of high resolution 3D models. *ACM Trans. Graph.* 35, 6 (2016).

Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. 1995. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH*.

Qing Fang, Wenqing Ouyang, Mo Li, Ligang Liu, and Xiao-Ming Fu. 2021. Computing sparse cones with bounded distortion for conformal parameterizations. *ACM Trans. Graph.* 40, 6 (2021), 1–9.

Ugo Finnendahl, Matthias Schwartz, and Marc Alexa. 2023. ARAP Revisited Discretizing the Elastic Energy using Intrinsic Voronoi Cells. In *Computer Graphics Forum*.

Matthew Fisher, Boris Springborn, Alexander I Bobenko, and Peter Schröder. 2006. An algorithm for the construction of intrinsic Delaunay triangulations with applications to digital geometry processing. In *ACM SIGGRAPH 2006*.

Michael S Floater. 1997. Parametrization and smooth approximation of surface triangulations. *Computer aided geometric design* 14, 3 (1997).

Michael Garland. 1999. Multiresolution Modeling: Survey and Future Opportunities. In *Eurographics 1999 - STARs*. Eurographics Association.

Michael Garland and Paul S. Heckbert. 1997. Surface simplification using quadric error metrics. In *SIGGRAPH 1997*. ACM.

Michael Garland and Paul S. Heckbert. 1998. Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings Visualization '98*. IEEE and ACM.

Tran S. Gieng, Bernd Hamann, Kenneth I. Joy, Gregory L. Schussman, and Issac J. Trotts. 1997. Smooth hierarchical surface triangulations. In *Proc. IEEE Vis.*

Mark Gillespie, Nicholas Sharp, and Keenan Crane. 2021a. Integer coordinates for intrinsic geometry processing. *ACM Trans. Graph.* 40, 6 (2021).

Mark Gillespie, Boris Springborn, and Keenan Crane. 2021b. Discrete conformal equivalence of polyhedral surfaces. *ACM Trans. Graph.* 40, 4 (2021).

Eitan Grinspun, Petr Krysl, and Peter Schröder. 2002. CHARMS: a simple framework for adaptive simulation. *ACM Trans. Graph.* 21, 3 (2002).

Xianfeng David Gu, Feng Luo, Jian Sun, and Tianqi Wu. 2018. A Discrete Uniformization Theorem for Polyhedral Surfaces. *Journal of Differential Geometry* 109, 2 (2018).

Igor Guskov, Wim Sweldens, and Peter Schröder. 1999. Multiresolution Signal Processing for Meshes. In *SIGGRAPH 1999*. ACM.

Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. 2000. Normal meshes. In *SIGGRAPH 2000*. ACM.

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

Hugues Hoppe. 1996. Progressive Meshes. In *SIGGRAPH 1996*. ACM.

Hugues Hoppe. 1997. View-dependent refinement of progressive meshes. In *SIGGRAPH 1997*. ACM.

Hugues Hoppe. 1999. New Quadric Metric for Simplifying Meshes with Appearance Attributes. In *Proc. IEEE Vis. Conf.*

Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1993. Mesh Optimization. In *Proc. SIGGRAPH*.

Shi-Min Hu, Zheng-Ning Liu, Meng-Hao Guo, Junxiong Cai, Jiahui Huang, Tai-Jiang Mu, and Ralph R. Martin. 2022. Subdivision-based Mesh Convolution Networks. *ACM Trans. Graph.* 41, 3 (2022).

Claude Indermitte, Thomas M. Liebling, Marc Troyanov, and Heinz Clémençon. 2001. Voronoi Diagrams on Piecewise Flat Surfaces and an Application to Biological Growth. *Theoretical Computer Science* 263 (2001).

Zhongshi Jiang, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2020. Bijective projection in a shell. *ACM Trans. Graph.* 39, 6 (2020).

Zhongshi Jiang, Ziyi Zhang, Yixin Hu, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2021. Bijective and coarse high-order tetrahedral meshes. *ACM Trans. Graph.* 40, 4 (2021).

Arseny Kapoulkine. 2019. *meshoptimizer*. https://github.com/zeux/meshoptimizer

Hermann Karcher. 2014. Riemannian center of mass and so called Karcher mean. *arXiv preprint arXiv:1407.2087* (2014).

Brian Karis, Rune Stubbe, and Graham Wihlidal. 2021. A Deep Dive into Nanite Virtualized Geometry. In *ACM SIGGRAPH 2021 Courses*.

Zachi Karni and Craig Gotsman. 2000. Spectral compression of mesh geometry. In *SIGGRAPH 2000*. ACM.

Michael Kazhdan, Jake Solomon, and Mirela Ben-Chen. 2012. Can Mean-Curvature Flow be Modified to be Non-singular? *Comput. Graph. Forum* 31, 5 (2012).

Liliya Kharevych, Patrick Mullen, Houman Owhadi, and Mathieu Desbrun. 2009. Numerical coarsening of inhomogeneous elastic materials. *ACM Trans. Graph.* 28, 3 (2009).

Andrei Khodakovsky, Nathan Litke, and Peter Schröder. 2003. Globally smooth parameterizations with low distortion. *ACM Trans. Graph.* 22, 3 (2003).

Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2013. Globally optimal direction fields. *ACM Trans. Graph.* 32, 4 (2013).

Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. 1998. Interactive Multi-Resolution Modeling on Arbitrary Meshes. In *SIGGRAPH 1998*. ACM.

Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence C. Cowsar, and David P. Dobkin. 1998. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *SIGGRAPH 1998*. ACM.

Dingzeyu Li, Yun (Raymond) Fei, and Changxi Zheng. 2015. Interactive Acoustic Transfer Approximation for Modal Sound. *ACM Trans. Graph.* 35, 1 (2015).

Mo Li, Qing Fang, Wenqing Ouyang, Ligang Liu, and Xiao-Ming Fu. 2022. Computing sparse integer-constrained cones for conformal parameterizations. *ACM Trans. Graph.* 41, 4 (2022), 1–13.

Beibei Liu, Yiying Tong, Fernando De Goes, and Mathieu Desbrun. 2016. Discrete connection and covariant derivative for vector field analysis and design. *ACM Trans. Graph.* 35, 3 (2016).

Hsueh-Ti Derek Liu, Alec Jacobson, and Maks Ovsjanikov. 2019. Spectral coarsening of geometric operators. *ACM Trans. Graph.* 38, 4 (2019).

Hsueh-Ti Derek Liu, Vladimir G. Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. 2020. Neural subdivision. *ACM Trans. Graph.* 39, 4 (2020).

Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. 2021. Surface multigrid via intrinsic prolongation. *ACM Trans. Graph.* 40, 4 (2021).

Kok-Lim Low and Tiow Seng Tan. 1997. Model Simplification Using Vertex-Clustering. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*. ACM.

Feng Luo. 2004a. Combinatorial Yamabe Flow on Surfaces. *Communications in Contemporary Mathematics* 6, 05 (2004).

Feng Luo. 2004b. Combinatorial Yamabe Flow on Surfaces. *Communications in Contemporary Mathematics* 6, 5 (2004).

Ahmed H. Mahmoud, Serban D. Porumbescu, and John D. Owens. 2021. RXMesh: a GPU mesh data structure. *ACM Trans. Graph.* 40, 4 (2021).

Edoardo Mainini. 2012. A description of transport cost for signed measures. *Journal of Mathematical Sciences* 181 (2012).

Josiah Manson and Scott Schaefer. 2011. Hierarchical Deformation of Locally Rigid Meshes. *Comput. Graph. Forum* 30, 8 (2011).

Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. *SIAM J. Comput.* 16, 4 (1987).

Ahmad Nasikun, Christopher Brandt, and Klaus Hildebrandt. 2018. Fast Approximation of Laplace-Beltrami Eigenproblems. *Comput. Graph. Forum* 37, 5 (2018).

Ahmad Nasikun and Klaus Hildebrandt. 2022. The Hierarchical Subspace Iteration Method for Laplace-Beltrami Eigenproblems. *ACM Trans. Graph.* 41, 2 (2022).

Daniele Panozzo, Enrico Puppo, and Luigi Rocca. 2010. Efficient multi-scale curvature and crease estimation. *Proc. of Comp. Graph., Comp. Vis. and Math.* 1, 6 (2010).

Gabriel Peyré, Marco Cuturi, et al. 2019. Computational optimal transport: With applications to data science. *Found. Trend. Mach. Learn.* 11, 5-6 (2019).

Gabriel Peyré and Stéphane Mallat. 2005. Surface compression with geometric bandelets. In *ACM Trans. Graph.*, Vol. 24. ACM.

Jovan Popovic and Hugues Hoppe. 1997. Progressive simplicial complexes. In *SIGGRAPH 1997*. ACM.

Nicolas Ray and Bruno Lévy. 2003. Hierarchical Least Squares Conformal Map. In *Pac. Conf. Comp. Graph. and App.*

Tullio Regge. 1961. General Relativity without Coordinates. *Il Nuovo Cimento (1955-1965)* 19, 3 (1961).

Igor Rivin. 1994. Euclidean Structures on Simplicial Surfaces and Hyperbolic Volume. *Annals of mathematics* 139, 3 (1994).

Jarek Rossignac and Paul Borrel. 1993. Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics*. Springer.

Pedro V. Sander, John M. Snyder, Steven J. Gortler, and Hugues Hoppe. 2001. Texture mapping progressive meshes. In *SIGGRAPH 2001*. ACM.

Rohan Sawhney and Keenan Crane. 2018. Boundary First Flattening. *ACM Trans. Graph.* 37, 1 (2018).

Patrick Schmidt, Janis Born, Marcel Campen, and Leif Kobbelt. 2019. Distortion-minimizing injective maps between surfaces. *ACM Trans. Graph.* 38, 6 (2019).

Patrick Schmidt, Marcel Campen, Janis Born, and Leif Kobbelt. 2020. Inter-surface maps via constant-curvature metrics. *ACM Trans. Graph.* 39, 4 (2020).

Peter Schröder. 1996. Wavelets in computer graphics. *Proc. IEEE* 84, 4 (1996).

William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. 1992. Decimation of triangle meshes. In *SIGGRAPH 1992*. ACM.

Gil Shamai and Ron Kimmel. 2017. Geodesic distance descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Nicholas Sharp and Keenan Crane. 2020a. A laplacian for nonmanifold triangle meshes. In *Computer Graphics Forum*, Vol. 39.

Nicholas Sharp and Keenan Crane. 2020b. You Can Find Geodesic Paths in Triangle Meshes by Just Flipping Edges. *ACM Trans. Graph.* 39, 6 (2020).

Nicholas Sharp, Mark Gillespie, and Keenan Crane. 2021. Geometry Processing with Intrinsic Triangulations. In *ACM SIGGRAPH 2021 courses*. ACM.

Nicholas Sharp, Yousuf Soliman, and Keenan Crane. 2019a. Navigating intrinsic triangulations. *ACM Trans. Graph.* 38, 4 (2019).

Nicholas Sharp, Yousuf Soliman, and Keenan Crane. 2019b. The Vector Heat Method. *ACM Trans. Graph.* 38, 3 (2019).

Jonathan Richard Shewchuk. 1999. Lecture notes on geometric robustness. *Eleventh International Meshing Roundtable* (1999).

Jonathan Richard Shewchuk. 2002. What Is a Good Linear Finite Element? Interpolation, Conditioning, Anisotropy, and Quality Measures.

Meged Shoham, Amir Vaxman, and Mirela Ben-Chen. 2019. Hierarchical Functional Maps between Subdivision Surfaces. *Comput. Graph. Forum* 38, 5 (2019).

Yousuf Soliman, Dejan Slepčev, and Keenan Crane. 2018. Optimal Cone Singularities for Conformal Flattening. *ACM Trans. Graph.* 37, 4 (2018).

Boris Springborn, Peter Schröder, and Ulrich Pinkall. 2008. Conformal equivalence of triangle meshes. *ACM Trans. Graph.* 27, 3 (2008).

Kenshi Takayama. 2022. Compatible intrinsic triangulations. *ACM Trans. Graph.* 41, 4 (2022).

Greg Turk. 1992. Re-tiling polygonal surfaces. In *SIGGRAPH 1992*. ACM.

Floor Verhoeven, Amir Vaxman, Tim Hoffmann, and Olga Sorkine-Hornung. 2022. Dev2PQ: Planar Quadrilateral Strip Remeshing of Developable Surfaces. *ACM Trans. Graph.* 41, 3 (2022).

Ofir Weber and Denis Zorin. 2014. Locally Injective Parametrization with Arbitrary Fixed Boundaries. *ACM Trans. Graph.* 33, 4 (2014).

Zipeng Ye, Ran Yi, Minjing Yu, Yong-Jin Liu, and Ying He. 2019. Geodesic centroidal voronoi tessellations: Theories, algorithms and applications. *arXiv preprint arXiv:1907.00523* (2019).

Jiayi Eris Zhang, Jèrèmie Dumas, Yun (Raymond) Fei, Alec Jacobson, Doug L. James, and Danny M. Kaufman. 2022. Progressive Simulation for Cloth Quasistatics. *ACM Trans. Graph.* 41, 6, Article 218 (2022).

Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).

Denis Zorin, Peter Schröder, T De Rose, L Kobbelt, A Levin, and W Sweldens. 2000. Subdivision for modeling and animation. *SIGGRAPH Course Notes* (2000).

Denis Zorin, Peter Schröder, and Wim Sweldens. 1997. Interactive multiresolution mesh editing. In *SIGGRAPH 1997*. ACM.