Keeping the Curves:
Guiding Mesh Simplification with the Gauss–Bonnet Theorem

A Thesis
Presented to
The Division of Mathematical and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Vaughn R. Zaayer

May 2026

Approved for the Division
(Mathematics)

_____

Kyle M. Ormsby

Acknowledgements

There are many, many people that without whom, I would not have been able to complete both this thesis and my journey at Reed. I am incredibly grateful for the students, faculty, and staff here that have contributed to my growth over the past few years.

I would like to thank my academic and thesis advisor Professor Kyle Ormsby for his guidance and patience. Working with him has taught me to push myself, and to embrace curiosity in all my future endeavors.

Of course, none of this would have been possible without my loving and supportive family. I would like extend my endless gratitude to my mom, my step-father Dave, my sister Annika, and my brother Aidan. They all have been by my side throughout the highs and lows of my undergraduate experience, and I could not be more grateful to have them.

The friends I have made here at Reed have helped me keep my chin up through each and every challenge I have faced. To Clara, Elaina, Julia, and Kasey — thank you all for the great memories we have shared, all starting from when we were clueless Freshmen in the halls of Naito. To Catherine and Sam — thank you both for the countless laughs, and for inspiring me to go above and beyond in pursuing my ambitions. To Ashton, Beth, Daniel, and Eliana — thank you for the many wonderful moments from over the years. I am forever grateful to have crossed paths with all of you, as it has taught me just as much as any class I have taken here.

List of Abbreviations

ASC	Abstract Simplicial Complex
GSC	Geometric Simplicial Complex
QEM	Quadric Error Metric
ICE	Intrinsic Curvature Error
iDT	Intrinsic Delaunay Triangulation

Table of Contents

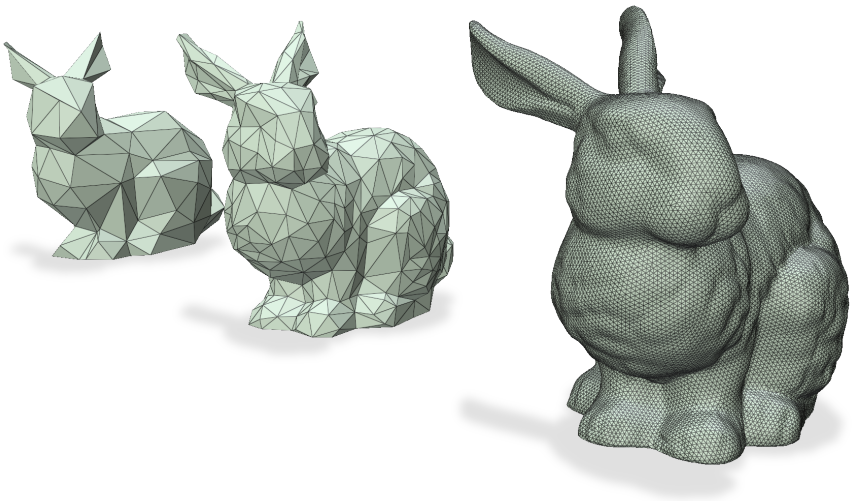
List of Figures

Abstract

Meshes representing 3-dimensional objects are used extensively in numerous areas of industry and study. In particular, a mesh may be used as an approximation for a real-life object, which is then used in practical simulations (for example, a mesh of a car being used for aerodynamic analysis). While higher vertex-count meshes better approximate the true results for a smooth surface, they also incur heavy computational cost. One solution is to produce a "coarsened" or "simplified" mesh that has a lower vertex count while maintaining similarity to its original counterpart. Coarsening schemes like Garland and Heckbert's *Surface Simplification Using Quadric Error Metrics* [GH97] provide decent approximations of an input mesh, but leave room for improvement in preserving key mesh features. Alternatively, intrinsic quantities like discrete Gaussian curvature can be defined and leveraged alongside the discrete Gauss–Bonnet Theorem to better maintain similarity between coarsened iterations of a mesh. In particular, mesh simplification using *Intrinsic Curvature Error* [Liu+23] utilizes these quantities to determine how much the removal of any vertex contributes to the difference between a fine and coarsened mesh. When comparing the two coarsening methods, *Intrinsic Curvature Error* yields a substantial increase in accuracy over *Quadric Error Metrics* while maintaining competitive runtimes.

Chapter 1

Introduction to Meshes



Many academic fields and professions rely on digital representations of 3-dimensional objects. Doctors use 3D scanning technology to diagnose and treat patients, engineers prototype complex designs with CAD, and VFX artists craft giant Kaiju monsters for the latest action movie. In each case, this 3D spatial information is represented as a *mesh*. Major innovations have been made in the world of mesh processing, forming the mathematical backbone for fields like computer vision, computer graphics, and geometry processing. Meshes are also invaluable in simulations such as aerodynamic and structural stress analysis.

1.1 Defining a Mesh

A mesh M contains two components: a *topological complex* and its *geometric data*. The former describes connectivity — which vertices belong to which edges, and which vertices and edges belong to which faces. The latter describes quantities associated with the mesh's shape, such as vertex positions, edge lengths, and face areas.

1.1.1 Topological Complex

Definition 1.1.1. Let T be a set of non-empty, finite subsets of some set S. We say that T is an *abstract simplicial complex* (ASC) if for every set $X \in T$, all non-empty subsets $Y \subseteq X$ also belong to T. Each $X \in T$ is called an *abstract simplex*.

The *vertex set* of an ASC T is defined as $V(T) = \{\{v\} \in T\}$, with the elements of $V(T)$ being *vertices*. For ASCs T_1, T_2, we consider the map $f_0 : V(T_1) \to V(T_2)$ to be a *vertex map* if for every simplex $\{i_0, \ldots, i_n\} \in T_1$, the image set $\{f_0(i_0), \ldots, f_0(i_n)\}$ is a simplex in T_2. The map f_0 extends to an *abstract simplicial map* $f : T_1 \to T_2$ by $f(\{i_0, \ldots, i_n\}) = \{f_0(i_0), \ldots, f_0(i_n)\}$. If both f and f_0 are bijections, then f is an *isomorphism* and T_1 and T_2 are *isomorphic*, denoted $T_1 \cong T_2$.

Definition 1.1.2. Let T be an abstract simplicial complex. We say that $L \subseteq T$ is a *subcomplex* if for every $X \in L$, every non-empty subset $Y \subseteq X$ is also in L. If L is a subcomplex consisting of a single element $X \in T$ and all of its non-empty subsets, then L is the closure of the simplex X. A simplex containing k elements is a $(k-1)$-simplex.

The *dimension* of an abstract simplex X is $\dim(X) = |X| - 1$. A maximal set $X \in T$ is called a *facet*, which is a simplex not contained in any other simplices in T. We say the dimension of T is the same as the dimension of its highest dimensional facets.

In Figure 1.1, we have the vertex set $V = \{i, j, k, l\}$. Each edge is a subset of size 2 of V, with the edge connecting vertices i and j being the 1-simplex $\{i, j\}$. Similarly, the face containing i, j, and k is the 2-simplex $\{i, j, k\}$, whose three 1-simplex components are $\{i, j\}$, $\{j, k\}$, $\{k, i\}$.

V	$\{i, j, k, l\}$
E	$\{ij, jk, ki, jl, lk\}$
F	$\{ijk, jlk\}$

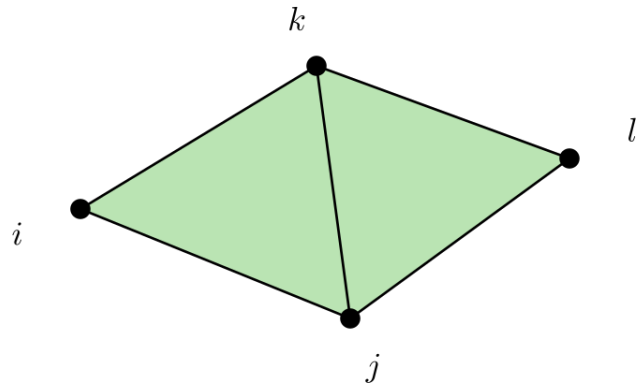
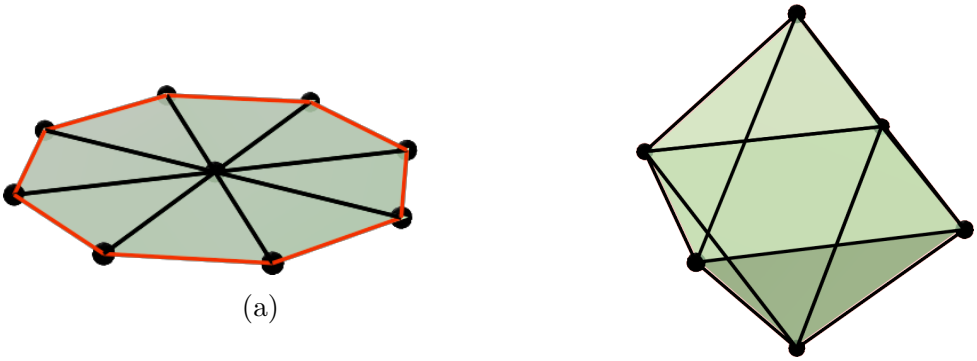


Figure 1.1: A topological complex with the vertex set $V = \{i, j, k, l\}$.

A simplicial complex cannot have repeated elements, so for any simplex containing the vertices i, j, we can assume that $i \neq j$. The number of $(k+1)$-simplices a k-simplex is a part of is the *degree* of that simplex. For example, if a vertex i is contained in ij, ik, and il, then $\deg(i) = 3$.

Definition 1.1.3. A k-dimensional abstract simplicial complex T is *pure* if every simplex of dimension less than k is a face of some simplex $\sigma \in T$ with $\dim(\sigma) = k$.

In our mesh, we require the topological complex to be a pure 2-dimensional simplicial complex, possibly with boundary. Informally, this means that each vertex belongs to at least two edges, and each edge belongs to at least one face. From here, we will define the topological complex of a mesh M as an abstract complex $T = T_0 \sqcup T_1 \sqcup T_2$, where T_0, T_1, and T_2 denote the sets of 0-, 1-, and 2-simplices, respectively. Along with our vertex set $T_0 = V$, we also define the *edge set* $T_1 = E$ and the *face set* $T_2 = F$.



(a) (b)

Figure 1.2: Topological complex (a) has a boundary (boundary edges highlighted), while topological complex (b) does not.

Definition 1.1.4. For a 1-simplex $ij \in T_1$, we say that ij is a *boundary edge* if it is contained in only one 2-simplex. The set of boundary edges is $\partial E \subseteq E$. Each vertex in a boundary edge is considered a *boundary vertex*.

If the topological complex of M has at least one boundary edge, then we say that M has a boundary.

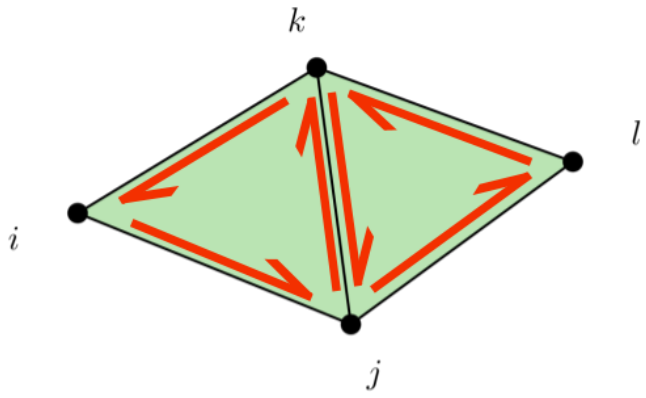


Figure 1.3: A topological complex with oriented 1-simplices. The orientations are defined as $\vec{ijk}$ and $\vec{jlk}$.

An orientation of a simplex is an equivalence class of orderings of its vertices under even permutations. Every abstract simplex admits two orientations. For each edge

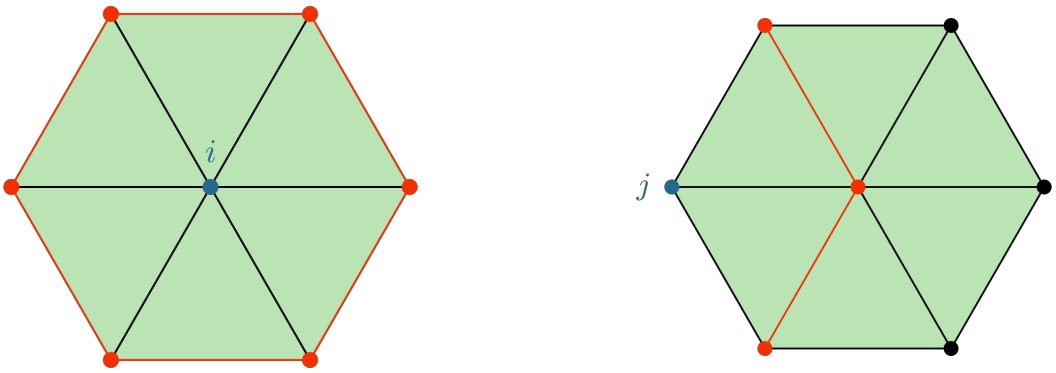
we can assign an ordering of the vertices, so an edge ij can be either $\vec{ij}$ or $\vec{ji}$. A face is oriented by permutations of ijk. In particular, $\vec{ijk}$ corresponds to the oriented edges being $\vec{ij}, \vec{jk}$, and $\vec{ki}$ – this gives us a counter-clockwise (CCW) orientation of ijk. The clockwise (CW) orientation gives us $\vec{ik}, \vec{kj}$, and $\vec{ji}$. In $\mathbb{R}^3$, CCW and CW are ambiguous, as the notation is dependent on the choice of viewing direction.

We say that two faces are *consistently oriented* if they disagree on a shared edge. Using the example from earlier, consider the faces ijk and jlk. If we give both faces a CCW orientation, we can notice that the shared edge jk is oriented as $\vec{jk}$ in the triangle ijk while the triangle jlk gives that same edge the orientation $\vec{kj}$. Alternatively, if both faces define the orientation $\vec{jk}$, that would mean ijk is oriented CCW while jlk is oriented CW.

Now we can say that a mesh as a whole is *orientable* if every facet in its topological complex is consistently oriented with its neighbors.

If an edge $ij \in E$ is contained in exactly 1 face (boundary edge) or exactly 2 faces (interior edge), then ij is a *manifold edge*. Vertices in manifold edges are *manifold vertices*.

Definition 1.1.5. Let T be an ASC and $i \in T$ a vertex of T. We define the *link* of a vertex in an abstract simplicial complex to be the set containing every abstract simplex $X \in T$ such that $i \notin X$ and $X \cup i$ is an abstract simplex in T.



(a) Link of interior vertex i.

(b) Link of boundary vertex j.

Figure 1.4: The links of an interior vertex i and boundary vertex j in a pure 2-dimensional ASC.

Definition 1.1.6. Let M be a mesh with topological complex T. We define M as a *manifold* mesh if and only if

1. the link of every vertex $i \in V$ is a cycle (if i is an interior vertex) or a finite path graph (if i is a boundary vertex) (see Figure 1.4), and

2. every edge $ij \in E$ is a manifold edge.

Otherwise, M is a *non-manifold* mesh.

Remark 1.1.1. Since we are in the 2-dimensional setting, a mesh M with a boundary has separate requirements for boundary vertices and edges from interior vertices and edges. On the boundary of M, each edge ij must be included in exactly one face, and the link of each vertex i must be a finite path graph. For the interior of M, each edge ij must be included in exactly two faces, and the link of each vertex i must be a single cycle. If both sets of these requirements are met, then M is a manifold mesh.

1.1.2 Geometric Data

As mentioned before, the topological complex of a mesh only describes connectivity. How a mesh sits in space is defined by its *geometric data*. In particular, we want to describe the geometric attributes (vertex positions, face areas, etc.) of 2-dimensional pure abstract simplicial complexes realized in $\mathbb{R}^3$.

In $\mathbb{R}^3$, the point i has the coordinates (x_i, y_i, z_i), and the corresponding vector $\vec{v}_i = \langle x_i, y_i, z_i \rangle$. If for a set of points $i_1, i_2, ..., i_l \in \mathbb{R}^3$ the difference vectors $\vec{v}_2 - \vec{v}_1, \vec{v}_3 - \vec{v}_1, ..., \vec{v}_l - \vec{v}_1$ are linearly independent, then we say that the points $i_1, i_2, ..., i_l$ are *affinely independent*. In other words, if i, j, k are affinely independent points in $\mathbb{R}^3$, then they are non-collinear.

Definition 1.1.7. In $\mathbb{R}^3$, there are 0-, 1-, and 2-dimensional geometric simplices:

- A 0-simplex is represented by a point.
- A 1-simplex is represented by a line segment.
- A 2-simplex is represented by a filled-in triangle.

Like with an abstract simplex (1.1.1), a *geometric k-simplex* is constructed of geometric $(k-1)$-simplices. That is, a geometric 1-simplex is made of two geometric 0-simplices (forming the line segment between two points in $\mathbb{R}^3$). All geometric 0-simplices that make up a geometric k-simplex must be affinely independent. We also refer to 0-simplices as vertices.

Definition 1.1.8. A *geometric simplicial complex* (GSC) G is a collection of geometric simplices in $\mathbb{R}^n$ such that:

1. If a geometric k-simplex σ is in G, then all faces of σ are also in G.
2. The intersection of any two simplices in G is either empty or a face of each.
3. G is a locally finite collection.

As with ASCs, a facet of G is the highest-dimension geometric simplex contained within it, and $\dim(G)$ is equal to the dimension of its highest dimensional facet.

Our meshes are defined in $\mathbb{R}^3$, so we will only be using 2-dimensional GSCs to define geometric properties.

We can realize a topological complex T geometrically by specifying a GSC G whose vertex scheme is isomorphic to T. Let the vertex scheme $\mathcal{K}$ of a GSC G be

the ASC whose simplices are the vertex sets of the geometric simplices in G; that is, $\mathcal{K} = \{\{v_0, ..., v_k\} : \text{Conv}(v_0, ..., v_k) \in G\}$. If $\mathcal{K}$ is isomorphic to T as an ASC ($K \cong T$), then we call G a *geometric realization* of T. All geometric information such as vertex positions, edge lengths, face areas, and others can be computed directly from $\mathcal{K}$.

1.2 Algorithms on Mesh Surfaces

Many applications of meshes involve surface algorithms. For example, a mesh representing the frame of a car can be used to simulate how stress permeates throughout the frame at a point of impact.

1.2.1 Partial Differential Equations and the Finite Element Method

Many algorithms that run on the surface of a mesh rely on *partial differential equations* (PDEs). For example, a PDE is used to track the diffusion of heat across a surface over time using the heat equation

$$\frac{\partial u}{\partial t} = \Delta u,$$

where Δ is the *Laplacian*. PDEs of this form are often discretized through a sequence of *Poisson problems*:

$$\Delta u = f,$$

where f represents some known input data or source function. Given f and the surface domain, we solve for the target variable u. Generally, computing the exact solution to a Poisson problem is too impractical, so we estimate u with a linear combination of basis functions ϕ. This approach is called the *finite element method* (FEM). Our goal is to come up with a $\tilde{u}$ such that

$$\tilde{u} = \sum_i x_i \phi_i,$$

with $x_i \in \mathbb{R}$ and $\{\phi_i\}$ being the finite set of basis functions. With a discrete mesh M, we can associate one basis function ϕ_i with each vertex i. A standard choice is the piecewise linear "hat function," which equals 1 at vertex i and 0 at each neighboring vertex. The ideal case would be finding some $\tilde{u}$ such that $|\tilde{u} - u| = 0$ — however, we obviously do not know what u is to begin with. Instead we test the residual $\Delta\tilde{u} - f$ with each basis function ϕ_j with the L^2 inner product:

$$\langle \Delta\tilde{u} - f, \phi_j \rangle = 0.$$

When the residual $\tilde{u} - f$ is orthogonal to ϕ_j, we satisfy the Galerkin condition. That is, we achieve the minimum possible error with the finite element ϕ_j.

Say we want to approximate some function $u : X \to \mathbb{R}$ where X is a smooth, differentiable surface. We create an approximation of X with a mesh M. As M is a piecewise-linear surface, each vertex i is associated with the linear basis function ϕ_i. We can "test" our candidate $\Delta\tilde{u}$ by computing the L^2 norm of $\nabla\tilde{u}$ and $\nabla\phi_j$ — since $\tilde{u}$ is a linear combination of basis functions, we have

$$\langle \nabla\tilde{u}, \nabla\phi_j \rangle = \sum_{i \in V} x_i \langle \nabla\phi_i, \nabla\phi_j \rangle.$$

Each $\langle \nabla\phi_i, \nabla\phi_j \rangle$ is encoded in the *Laplace matrix* as L_{ij}. The entries in L are $w_{ij} = \sum_{ijk} \frac{1}{2} \cot \theta_{ij}^k$, or the *cotangent weight* of the edge ij, where ϕ_{ij}^k is the angle incident the vertex k in the triangle ijk. On high quality triangulations, all cotangent weights will be nonnegative [SGC21]. We can now solve the problem $Lx = b$ where x is the coefficients x_i and $b_i = \langle f, \phi_i \rangle$ [Cra25].

Computing approximations of geodesic distance via the *heat method* is a fairly straightforward implementation of PDEs on a mesh surface. Broadly, this method uses a point as a source of "heat," then measures the dispersion of heat across the surface at some optimal time step t. The distance from the heat source to any point on the mesh is then approximated from that point's heat value [CWW13].

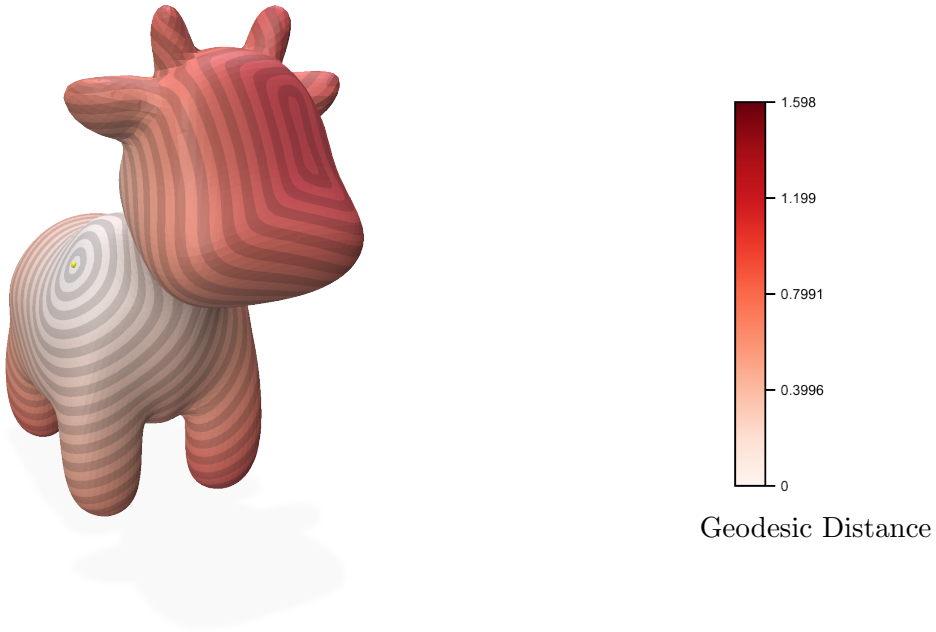


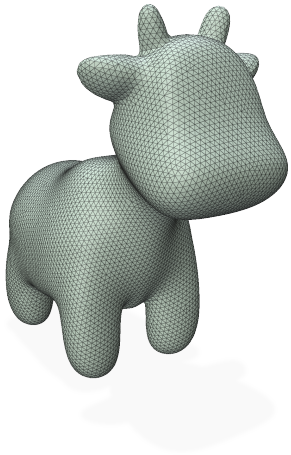
Figure 1.5: "Spot" with a colormap texture representing geodesic distance, computed using the heat method. The geodesic approximation was computed using the Geometry Central C++ library [SC+19] and rendered using Polyscope [Sha+19].

1.2.2 Mesh Coarsening

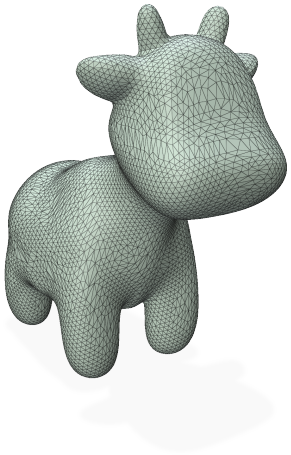
Computing an accurate approximation of some function u on a mesh comes with a tradeoff. As the resolution of M (i.e. $|V|$) increases, we expect the values of $\tilde{u}$ on M to

get closer to the values of u on X. However, the approximation $\tilde{u}$ must be computed for each and every vertex of M, which quickly becomes expensive. In settings where computational efficiency is crucial, we may want to take a high resolution M and create a lower resolution *coarsened* mesh $\tilde{M}$. Our goal is to reduce the number of vertices of an existing mesh (and thus the number of computations) while maintaining relatively high accuracy of the FEM solution on the *fine* M. This then poses the question — how do we decide which vertices to remove?

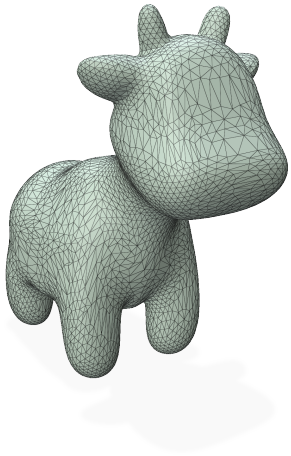
One such method is through Garland and Heckbert's *Quadric Error Metric* (QEM) [GH97]. While we will avoid an in-depth description here, QEM roughly works by selecting two vertices $i, j \in V$ that are either under some distance threshold or connected via an edge, and collapses one into the other. An immediate issue is the collapsing of two vertices that do not share an edge, potentially creating a non-manifold mesh from a manifold one. For our purposes, we will only consider collapses along edges when using QEM decimation.



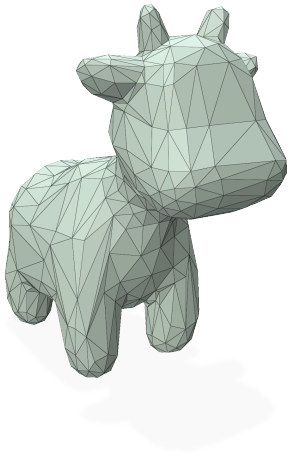
(a) Original resolution ($|V| = 11533$).



(b) $\approx 75\%$ resolution ($|V| = 8660$).



(c) $\approx 50\%$ resolution ($|V| = 5775$).



(d) $\approx 5\%$ resolution ($|V| = 573$).

Figure 1.6: Four resolutions of "Spot," ordered from highest to lowest resolution. Each coarsened mesh was computed via QEM.

As you can see in Figure 1.6, QEM maintains visual consistency between each resolution, prioritizing the removal of vertices on more "flat" areas (such as the front of the head). For computer graphics use cases, this is excellent — objects farther away from the viewport do not need visual detail, so coarsening while maintaining general visual consistency is ideal for reducing rendering costs.

1.2.3 Finite Element Quality

Tangentially related to how the resolution of a mesh affects the accuracy of the FEM, we are also concerned with individual finite element *quality*. In this case, our finite elements are the triangular faces on our mesh. We consider a "high" quality element to be as close to being an equilateral triangle as possible, and a "low" quality element is near-degenerate. Specific details about how the shape of an individual element impacts overall accuracy of the FEM can be found from Schewchuk [She02], but our main takeaway is that evaluating the FEM on near-degenerate triangles in our mesh yields floating point errors that can dramatically impact overall accuracy.

Figure 1.7: A mesh of a cow ($|V| = 400,000$) with the results of the heat-based geodesic distance approximation. This particular triangulation is poor (with a minimum angle of $0.05°$), resulting in many inconsistencies. First of all, the minimal distance given by the color map is -1.273, despite geodesic distance always being positive. Notably this minimum value occurs on the rear right leg, even though the source point is on the left flank. The isolines are also splotchy and inconsistent, reflecting significant inaccuracy. A much more accurate result is given in Figure 2.2.

Further, coarsening a mesh via QEM will not remove these near-degenerate triangles,

so a poor initial triangulation of M will lead to poor triangulations of a coarsened output $\tilde{M}$.

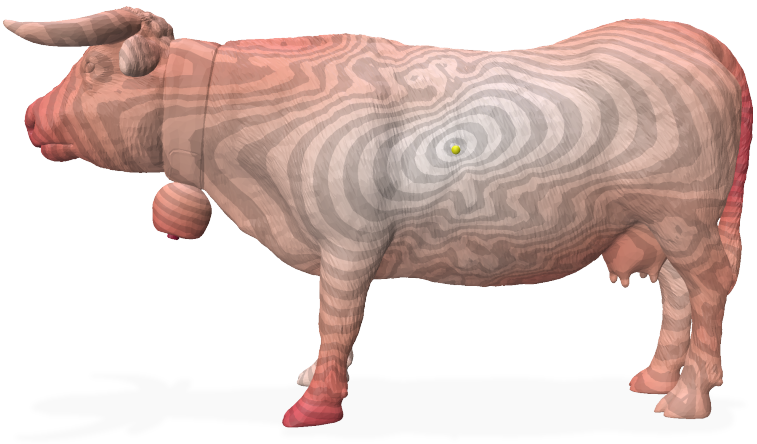


Figure 1.8: The same model as in Figure 1.7 but reduced to $\approx 25\%$ of the original vertex count via QEM. Computing the geodesic distance approximation on the coarsened mesh gives slightly better results around the source point, but still contains the major errors present in the fine mesh.

Fortunately we can mitigate the pitfalls of poor quality triangulations by computing a *Delaunay triangulation*, which will be discussed in Section 2.1.

Chapter 2

Intrinsic Simplification

Coarsening a mesh via quadric error metrics (QEM) uses embedding-specific geometric information through vertex positions. However, we can also create coarsened meshes by leveraging the *intrinsic* quantities of the input. Algorithms running on the intrinsically coarsened mesh benefit from substantial reductions in computational cost compared to the fine mesh while maintaining greater accuracy than when run on extrinsically coarsened meshes. The coarsening method using *Intrinsic Curvature Error* (ICE) from Liu et al. [Liu+23] takes advantage of intrinsic quantities, as we will discuss further in Section 2.3.3.

2.1 Delaunay Triangulations

Consider a GSC G with triangles $\sigma_m \in G$. Let $\Omega_G = \bigcup \sigma_m$ be the *domain* of G — equivalently, the domain of G is the set of all points in $\mathbb{R}^3$ contained in some geometric simplex of G. We say that G is a *triangulation* of the domain Ω_G. A domain may have multiple triangulations, so for GSCs $G \neq G'$ we can still have $\Omega_G = \Omega_{G'}$.

Not all triangulations are of equal quality, however. Generally we want all triangles in a triangulation to be as close to equiangular as possible, avoiding skinny near-degenerate triangles. Luckily, we can leverage the *Delaunay triangulation*, which maximizes the minimum angle over all triangulations of the given point set. In other words, in the set of all possible triangulations, the Delaunay triangulation is guaranteed to have the most "optimal" triangles.

For a planar point set, the Delaunay triangulation can be reached from any initial triangulation by a greedy edge-flipping algorithm. Extrinsic edge flips on a mesh change its geometry, since the new diagonal kl is a straight line in $\mathbb{R}^3$ cutting through the old configuration. To avoid this, we use *intrinsic* edge flips, which operate on edge lengths directly.

2.1.1 Intrinsic Delaunay Triangulations

Recall that a mesh is defined in two parts: the topological complex (how mesh elements connect to one another) and geometric data (how the mesh is shaped). Geometric

data can come in the form of either *intrinsic* or *extrinsic* properties.

Definition 2.1.1. An *intrinsic* quantity is one that can be computed from only edge lengths within the surface, without reference to how the surface is embedded in ambient space.

Definition 2.1.2. An *extrinsic* quantity is one that requires the ambient embedding in order to compute.

For example, face area would be an intrinsic quantity, as it can be computed purely from the edge lengths and angles of a triangle. Vertex positions, on the other hand, are explicitly tied to how the mesh is embedded in $\mathbb{R}^3$. From here we can start constructing an intrinsic triangulation of our mesh surface.

For an intrinsic triangulation, we assign a length $l_{ij} \in \mathbb{R}_{>0}$ for each edge $ij \in E$ to define a mesh's shape. If we're starting with a mesh defined extrinsically with vertex coordinates, we can get the edge length of ij with the standard Euclidean distance function:

$$l_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2},$$

where x_i, y_i, and z_i represent the (x, y, z) coordinates of vertex i. All edge lengths must be non-negative, and any face $ijk \in F$ must also satisfy the triangle inequality:

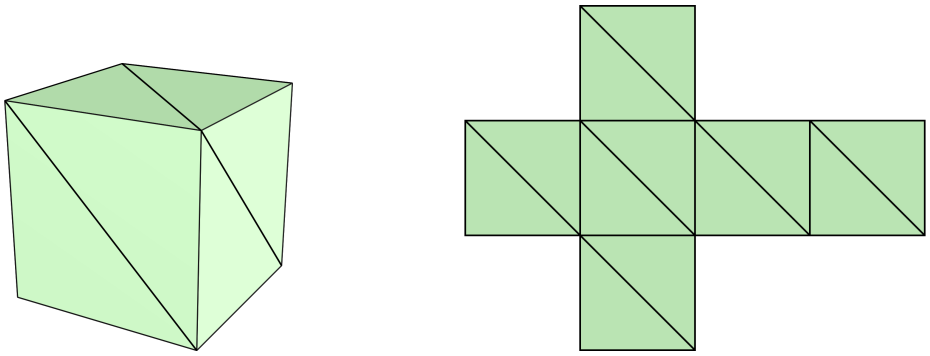
$$\begin{aligned} l_{ij} + l_{jk} &\geq l_{ik}, \\ l_{jk} + l_{ki} &\geq l_{ij}, \\ l_{ki} + l_{ij} &\geq l_{jk}. \end{aligned}$$

By decoupling the triangulation of our mesh from its geometry, we are now free to modify the triangulation to our liking. Our basic operation for doing so is the *edge flip* operation.

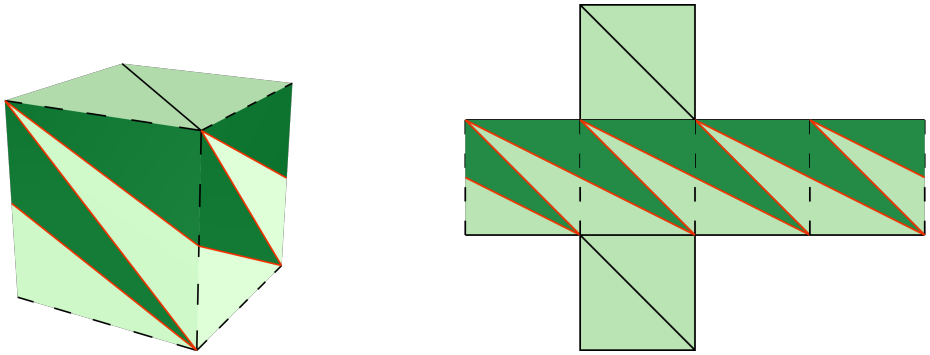
Definition 2.1.3. Consider two triangles, one defined by ijk and the other defined by ilj, joined by the edge ij. We say the edge ij is *flippable* if $ijk \cup ilj$ form a convex quad (specifically, the planar quad from unfolding ijk and ilj across edge ij into the plane), and i, j both have degrees of at least 2 [SGC21, Section 2.3.4].

Select a flippable edge ij from our mesh, and let ij be contained in ijk and ijl. To perform an edge flip, we simply draw a new edge kl and delete ij. Edge flips on an intrinsic triangulation notably *do not* affect the geometry of our mesh. Visually, this appears as a geodesic line that is able to wrap around edges on the mesh's surface when connecting two vertices.

Definition 2.1.4. An edge ij is *intrinsic Delaunay* if the two angles α and β opposite to ij in its incident triangles satisfy $\alpha + \beta \leq \pi$. If every edge in a mesh is intrinsic Delaunay, then the mesh (topological complex together with its intrinsic edge lengths) is an *intrinsic Delaunay triangulation* (iDT).



(a) A geometric simplicial complex with only extrinsicly defined edges.



(b) A geometric simplicial complex with intrinsically defined edges (highlighted). Notably, these edges are still intrinsically straight despite having a bend extrinsically.

Figure 2.1: Extrinsic edges (a) versus intrinsic edges (b).

Within an intrinsic triangulation, we can iterate over each edge ij and determine if it is intrinsic Delaunay — if not, we simply perform an edge flip. After a finite number of flips we create an intrinsic Delaunay triangulation.

An iDT has several nice qualities. For instance, an iDT both maximizes minimum corner angles and ensures that every entry in the Laplace matrix L is nonnegative [SGC21, Section 4.10.1]. As discussed in Section 1.2.3, these are the most ideal conditions for avoiding poor quality finite elements. We instead use the FEM on L from our iDT for significantly more robust results.

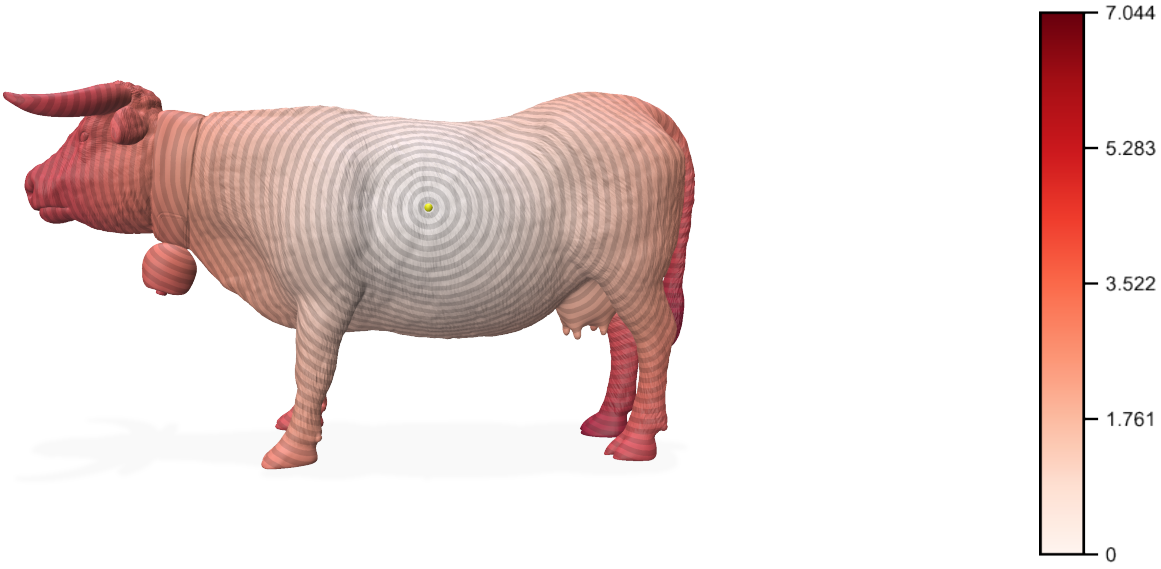


Figure 2.2: The same cow mesh, geodesic approximation algorithm, and source point as Figure 1.7, except with geodesic approximation computed on the higher quality intrinsic Delaunay triangulation (which has a minimum angle of 1.87°).

2.2 Discrete Curvature

A critical part of any mesh coarsening algorithm is the *error metric*, which guides the algorithm in deciding which vertices to remove and in what order. As discussed in section 1.2.2, we want to reduce the mesh's vertex count while maintaining "similarity" — how we determine similarity depends on what we want to do with our coarsened mesh. QEM does well maintaining similarity with the fine mesh, but generally disregards features important for surface algorithms. While QEM is based on vertex positions, we can leverage intrinsic quantities to maintain better approximations of PDEs using FEM. A particular intrinsic geometric quantity, *discrete curvature*, ends up being the perfect tool to track how closely our coarsened mesh approximates the fine mesh. After defining discrete curvature we will also explore the *discrete Gauss–Bonnet Theorem* in Section 2.2.2, which gives us a conservation property of curvature that is used heavily in ICE.

From this point onwards, we will only consider *closed* meshes, or meshes without boundary.

2.2.1 Curvature on Differential Surfaces

We want to define curvature on our mesh, which is a *piecewise-linear surface* — note that we will use the terms "piecewise-linear surface" and "discrete surface" interchangeably. To build intuition on how curvature is defined on a discrete surface, we will start with the much more common notion of curvature on smooth, differential surfaces. To do that, we begin with curvature in $\mathbb{R}^2$.

Consider an oriented smooth curve $r : \mathbb{R} \to \mathbb{R}^2$, with $t \mapsto r(t)$. We define a fixed

point $i = r(t_0)$, as well as two points $j = r(t_0 - \epsilon)$ and $k = r(t_0 + \delta)$, where $\epsilon, \delta > 0$.

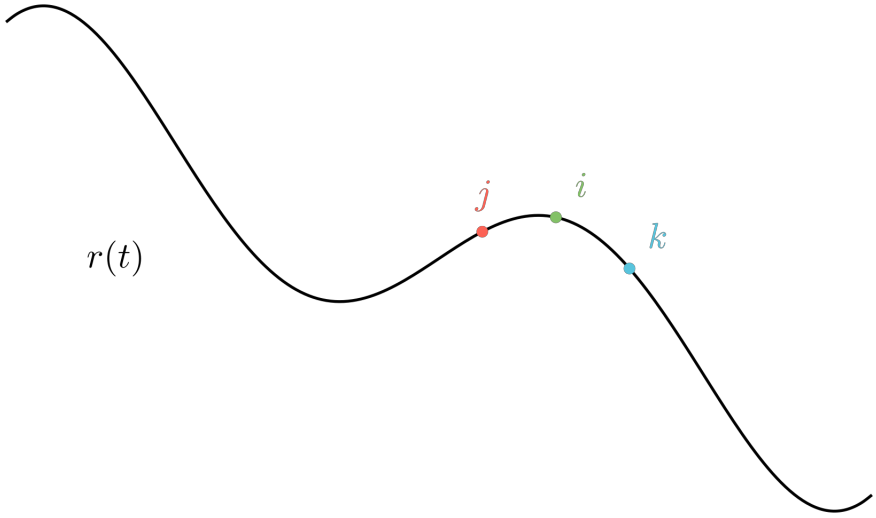


Figure 2.3: The curve $r(t)$ in $\mathbb{R}^2$ with the points i, j, k.

Using these three points, we can define an *osculating circle* with the center point $c \in \mathbb{R}^2$ and radius $\rho \in \mathbb{R}_{>0}$.

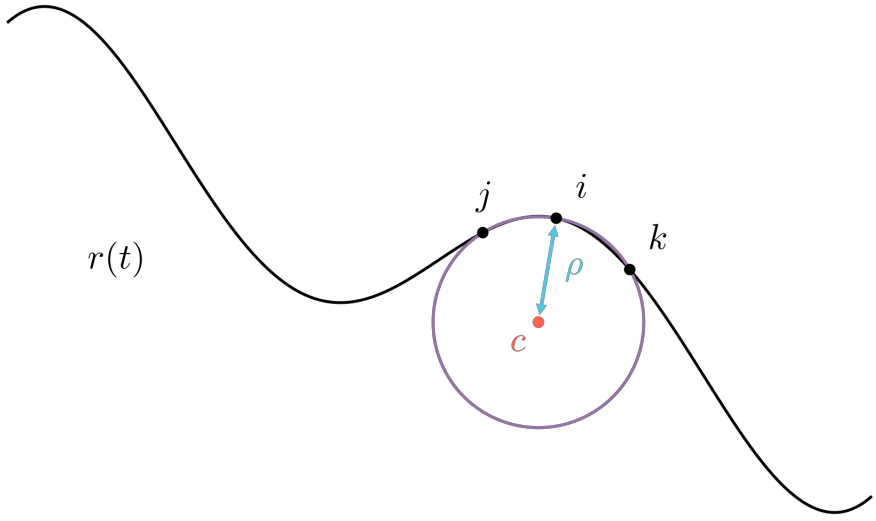


Figure 2.4: Osculating circle defined by the points i, j, k.

As we take the limit of ϵ, δ independently approaching 0, we get c, ρ defined for a circle that aligns perfectly with $r(t)$ at i. The *curvature* κ of a circle is the reciprocal of its radius, $\kappa = \frac{1}{r}$. So formally, we define the curvature of $r(t)$ at i as

$$\kappa(i) = \lim_{\epsilon,\delta \to 0} \frac{1}{\rho}. \tag{2.1}$$

To put it broadly, more sharply curved arcs correspond to smaller values of ρ and thus larger values of κ. Curves closer to straight lines have κ values that approach 0. If the center of the osculating circle c is on the positively-oriented side of $r(t)$, then κ has a positive value — if c is on the negatively-oriented side, then κ will be negative.

This is only the very basics of curvature in $\mathbb{R}^2$, but it is all we need to get to work on a differential surface. Consider a surface $X \subset \mathbb{R}^3$ that is oriented and differentiable. We can select a point on the surface $i \in X$, and since our surface is oriented, the normal vector $\vec{n}(i)$ is defined as well. Orthogonal to $\vec{n}(i)$ is a circle's worth of possible tangent vectors $\vec{t}(i)$. For now we can just pick any arbitrary $\vec{t}(i)$. By taking the cross product $\vec{n}(i) \times \vec{t}(i)$, we get a third vector orthogonal to both. Let (x_i, y_i, z_i) be the coordinates of i. Taking the dot product of $\vec{t_i} \times \vec{n}_i$ with the displacement vector $\langle x - x_i, y - y_i, z - z_i \rangle$ gives the equation of a plane through i:

$$\vec{t_i} \times \vec{n}_i \cdot \langle x - x_i, y - y_i, z - z_i \rangle = 0. \tag{2.2}$$

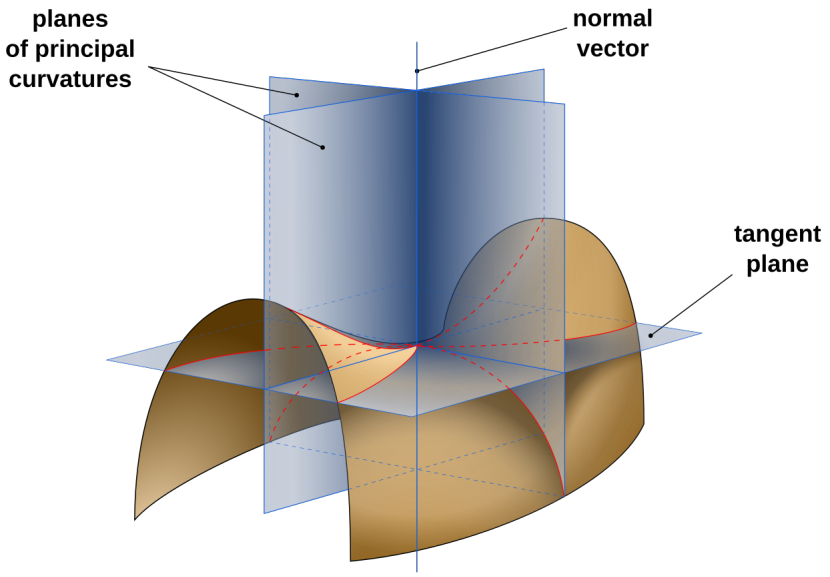


Figure 2.5: A differential surface with normal planes [Gab06].

We refer to this as a *normal plane* at i, denoted $N_i(\vec{t_i})$. The normal plane creates a "cross-section" with $N_i \cap X$ (called the *normal section*), which is a planar curve. Just like finding curvature in $\mathbb{R}^2$, we define an osculating circle to get κ_i. Finally we denote the curvature value associated with a particular normal plane N_i as κ_{N_i}. For the sake of simplicity, we will just use κ_N and assume we are talking about the point i.

As mentioned, there is a circle's worth of tangent vectors at i, and thus a circle's worth of normal sections. Of all the different values of κ_N, we denote the maximum value as κ_1 and the minimum value as κ_2. These are the *principal curvatures* at i. Using the principal curvatures we can define the *Gaussian curvature* at i.

Definition 2.2.1. The *Gaussian curvature* K at a point i on an oriented, differentiable surface $X \subset \mathbb{R}^3$ is the product of the principal curvatures at i,

$$K_i = \kappa_1 \kappa_2.$$

If exactly one of the principal curvatures is 0, then i is a parabolic point; if both are 0, it is a planar point. In either case, $K_i = 0$. When the principal curvatures have matching signs, then $K_i > 0$. Otherwise, when the signs are opposite, $K_i < 0$. Positive curvature indicates that i is an elliptic point (a local bowl or dome), and negative curvature indicates that i is a hyperbolic point.

2.2.2 Curvature on Discrete Surfaces

Having covered the basics of differential curvature, we can start defining discrete curvature. Consider an oriented geometric simplicial complex (1.1.8) Y in $\mathbb{R}^3$. If we use the same approach as with differential curvature, we immediately run into an issue — any point on a face of Y will have zero curvature, which does not give us much to work with. At each vertex of Y, the surface bends between adjacent planar faces; this is where discrete curvature will be concentrated.

Borrowing from differential curvature, we can compute curvature at a vertex i via a *Gauss map*. For a smooth, oriented surface X and the unit sphere S^2, the differential Gauss map is the map $f : X \to S^2$ defined by taking the point $i \in X$ to the unit normal at i — that is, the point at the tip of $\vec{n}(i)$ is a point on the surface of the unit sphere. To extract Gaussian curvature at i, we take a sequence of simple, orientable, compact regions $R \subseteq X$ containing i and form the ratio area

$$\text{avg. curvature}(R) = \frac{\text{area}(\bar{R})}{\text{area}(R)},$$

where $\bar{R}$ is the region $\vec{n}(R)$ on S^2. Note that area(R) and area$(\bar{R})$ are *signed* areas. As area(R) tends to zero while still maintaining $i \in R$, we get

$$\lim_{\text{area}(R)\to 0} \frac{\text{area}(\bar{R})}{\text{area}(R)} = K_i,$$

thus defining Gaussian curvature at i [Cas96]. The discrete Gauss map $f : Y \to S^2$ can be defined in a similar way, instead using face normals incident to $i \in Y$ to define a spherical polygon on S^2.

We start by defining the set of triangles σ containing vertex i as the *neighborhood* of i, denoted $\mathcal{F}_i$. Essentially, $\mathcal{F}_i$ is a simplicial complex where each facet contains i. For each $\sigma_m \in \mathcal{F}_i$, define the *face normal* of σ_m as $\vec{n}(\sigma_m)$. Since Y is oriented, we can cyclically order the triangles around i as $\sigma_1, \sigma_2, \ldots, \sigma_n$. Just as in the differential Gauss map, we can define the discrete Gauss map $f : \mathcal{F}_i \to S^2$ by $f(\sigma_m) = \vec{n}(\sigma_m)$, the unit face normal of σ_m. This gives us a set of points $s_1, \ldots, s_n$ on the surface of the unit sphere. If we then use the ordering of the face normal vectors to create a geodesic edge between each consecutive s_m as well as an edge from s_n to s_1, we create a *spherical polygon* Ω with the vertices $s_1, \ldots, s_n$. We can determine the interior by the ordering of the vertices. Let α_m be the interior angle in Ω corresponding to the vertex s_m.

Theorem 1. *[TMF22] The area of a spherical polygon Ω with n vertices and interior angles $\alpha_1, \alpha_2, \ldots, \alpha_n$ is*

$$\text{area}(\Omega) = (2-n)\pi + \sum_{m=1}^{n} \alpha_m.$$

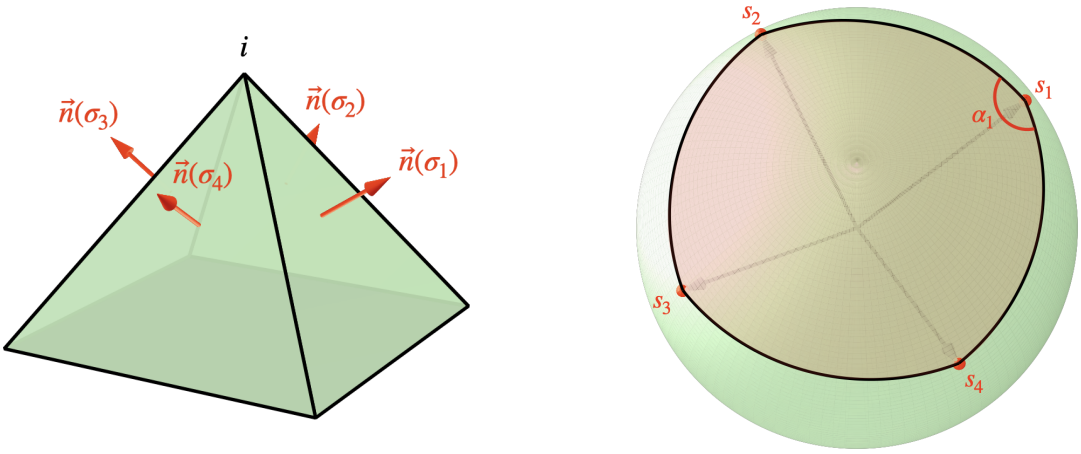


Figure 2.6: The discrete Gauss map corresponding to $\mathcal{F}_i$.

Using this theorem, we can compute the area of the spherical polygon associated with the vertex $i \in Y$. We then say that the discrete Gaussian curvature K_i is equal to the area of its spherical polygon via the Gauss map. While this method of finding K_i follows some intuition from the differential setting, there is a much easier way to compute it.

Definition 2.2.2. Let i be a vertex in a triangle σ, and let θ_m^i be the angle in σ corresponding to i. We define *angle defect* as

$$d(i) = 2\pi - \sum_{\sigma_m \in \mathcal{F}_i} \theta_m^i.$$

Angle defect becomes an easy analog for curvature at a glance. For a mesh with the vertex i, $\mathcal{F}_i$ that is concave or convex, $d(i) > 0$. If the neighborhood is instead a saddle point, then $d(i) < 0$. Of course when $\mathcal{F}_i$ is flat, then we have $d(i) = 0$.

Theorem 2. *The discrete Gaussian curvature of i is equal to the angle defect of i.*

Proof. We will show that $\text{area}(\Omega) = d(i)$ by constructing a planar quadrilateral whose interior angles encode the spherical interior angle α_m and the mesh interior angle θ_m^i. Start with a mesh M with the vertex set V, and vertex $i \in V$. Let there be n triangles $\sigma \in \mathcal{F}_i$, and consider the triangles $\sigma_{m-1}, \sigma_m, \sigma_{m+1}$ with the ordering $\sigma_{m-1} < \sigma_m < \sigma_{m+1}$. Denote the shared edge of σ_{m-1} and σ_m as E_1 and the shared edge of σ_m and σ_{m+1} as E_2.

Next, we define two planes from the cross products (see Eq. 2.2) of face normals $\vec{n}(\sigma_{m-1}) \times \vec{n}(\sigma_m)$ and $\vec{n}(\sigma_m) \times \vec{n}(\sigma_{m+1})$, denoted N_1 and N_2 respectively. The intersection of N_1 and E_1 forms a right angle, as does the intersection of N_2 and E_2. This

is because N_1's normal direction, $\vec{n}(\sigma_m) \times \vec{n}(\sigma_{m+1})$, is parallel to E_1, and thus N_1 is perpendicular to E_1. The same holds for N_2 and E_2. We know that the angle between E_1 and E_2 is θ_i associated with σ_m.

Recall that on the discrete Gauss map, the tips of $\vec{n}(\sigma_{m-1}), \vec{n}(\sigma_m)$, and $\vec{n}(\sigma_{m+1})$ become vertices for a spherical polygon on S^2. Because N_1 and N_2 are defined by cross products including $\vec{n}(\sigma_m)$, the two planes will intersect at $\vec{n}(\sigma_m)$, meaning that the angle at the intersection point α_m is the same as the interior angle incident to s_m from the spherical polygon. We can then use N_1, N_2, E_1, and E_2 to construct a planar quadrilateral.

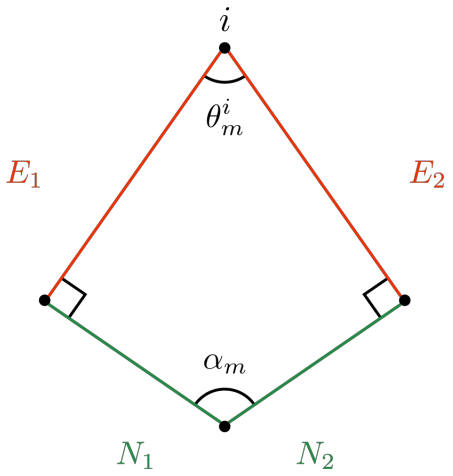


Figure 2.7: The planar quadrilateral formed by E_1, E_2, N_1, and N_2.

The sum of the interior angles of a quadrilateral is 2π, so we can solve for α_m like so:

$$\frac{\pi}{2} + \frac{\pi}{2} + \theta_m^i + \alpha_m = 2\pi \implies \alpha_m = \pi - \theta_m^i.$$

Plugging in $\pi - \theta_m^i$ to Thm. 1, we get the following:

$$\begin{aligned}
\text{area}(\Omega) &= (2-n)\pi + \sum_m^n \pi - \theta_m^i \\
&= 2\pi - \pi n + \pi n - \sum_m^n \theta_m^i \\
&= 2\pi - \sum_m^n \theta_m^i \\
&= d(i).
\end{aligned}$$

Since Ω was chosen such that $\text{area}(\Omega) = K_i$ where K_i is the discrete Gaussian curvature at i, we conclude $K_i = d(i)$. □

Theorem 3. Discrete Gauss–Bonnet Theorem. *For a simplicial complex with the set of vertices V, set of edges E, and set of faces F,*

$$\sum_{i \in V} d(i) = 2\pi\chi,$$

where the Euler characteristic $\chi = |V| - |E| + |F|$.

That is, the discrete Gauss–Bonnet theorem guarantees that the sum of the angle defects for every vertex $i \in V$ for a mesh is equal to $2\pi\chi$, meaning as vertices are removed from a mesh, the total curvature is conserved as the collapse of an edge does not change the value of χ. With χ invariant, $2\pi\chi$ is also invariant, and thus so is $\sum_{i \in V} d(i)$. This is a powerful result that lets us define an error metric, which we do in Section 2.3.3.

2.3 Intrinsic Curvature Error

A simplification algorithm relies on some metric of "cost", or the degree to which the removal of some vertex, edge, or face impacts the similarity between a fine and coarsened mesh. For example, QEM uses quadric matrices at each vertex to compute how different a mesh becomes when two of its vertices are contracted together. These particular computations are based on vertex positions, which are of course extrinsic properties of a mesh. Our intrinsic simplification algorithm must instead depend on an intrinsic property. Thankfully, discrete Gaussian curvature is entirely intrinsic.

In the previous section, we discussed the discrete Gauss–Bonnet Theorem, which states that the sum of all vertex Gaussian curvatures is a constant $2\pi\chi$. In other words, total Gaussian curvature is conserved as vertices are removed.

Intrinsic curvature error (ICE) utilizes discrete Gaussian curvature alongside intrinsic Delaunay triangulations to guide simplification [Liu+23]. The following Sections 2.3.1, 2.3.2, and 2.3.3 are expositions on the original paper from Liu et al.

2.3.1 Calculating Error in 2D

The purpose of an error metric is to measure how different a coarsened mesh is from its corresponding fine mesh. With ICE, we assign a distribution of non-negative mass on the set of vertices. When a vertex is removed, we use *optimal transport* to quantify the effort of redistributing the mass to neighboring vertices [PC19]. We use a *Karcher mean* to encode the center of mass of all the fine vertices contributing the mass of a coarse vertex [Kar14]. The cost of removing a vertex is determined by how much mass needs to be redistributed, and how far that mass must travel for redistribution.

We start with a distribution of mass $m : V \to \mathbb{R}_{\geq 0}$ such that each vertex i has the associated mass m_i. We also assign a vector $\vec{t_i}$ to keep track of i's *center of mass*, or the weighted average of where i's mass comes from. Since m_i is initialized with no other contributors, we set $\vec{t_i} = \vec{0}$ for each i.

Let i be the vertex we want to remove, and let $\mathcal{N}_i$ be the set of vertices j such that $ij \in E$. (We also refer to $\mathcal{N}_i$ as the "neighborhood" of i.) As i is removed, a fraction

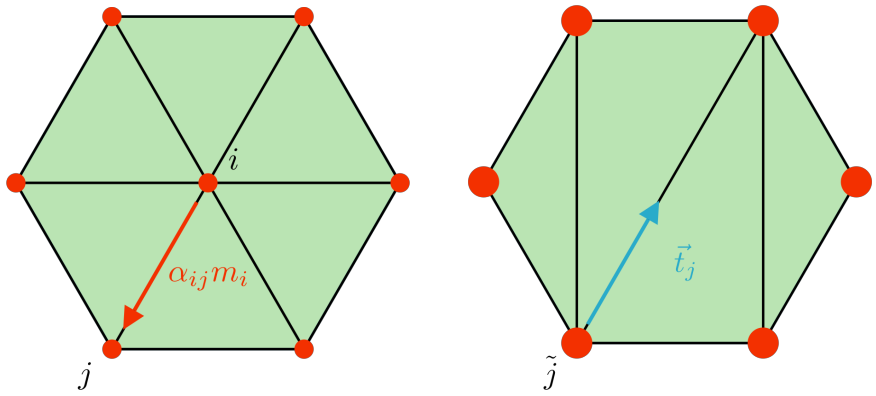


Figure 2.8: When i is removed from the mesh, a portion of its mass m_i gets redistributed to j. Then, $\vec{t}_j$ is updated to reflect where j received its mass.

of its mass m_i is added to each neighboring m_j. Let $\tilde{m}_j$ be the updated mass of j, and α_{ij} as the fraction of m_i added to m_j such that $\sum_{j\in\mathcal{N}_i}\alpha_{ij} = 1$. More precisely,

$$\tilde{m}_j = \alpha_{ij}m_i + m_j.$$

How we determine α_{ij} will be discussed in greater detail in 2.3.3. After updating the mass, we also want to update $\vec{t}_j$ to reflect how much m_i contributed to $\tilde{m}_j$. We denote this new value as $\tilde{t}_j$.

Let $\vec{e}_{ji}$ be the vector pointing from j to i along the edge ij, with $||\vec{e}_{ji}|| = l_{ij}$. With this, we compute

$$\tilde{t}_j = \frac{\alpha_{ij}m_i(\vec{t}_i + \vec{e}_{ji}) + m_j\vec{t}_j}{\alpha_{ij}m_i + m_j},$$

giving us both of the updated values $\tilde{m}_j$ and $\tilde{t}_j$. Note that $\vec{t}_i$ is also incorporated in the above equation — this allows us to consider the overall traversal of mass over several iterations. Finally, the cost associated with removing i, denoted C_i, is defined as

$$C_i = \sum_{j\in\mathcal{N}_i}\tilde{m}_j||\tilde{t}_j||. \tag{2.3}$$

2.3.2 Calculating Error in 3D

In the 2-dimensional setting, we have the flexibility to add tangent vectors as needed. Tangent vectors at each vertex in the 3-dimensional setting are defined in their own *tangent spaces*, so they cannot be added as directly. Instead, we *parallel transport* tangent vectors from a vertex i to another vertex j.

Using the method described in Liu et al., we define $\mathcal{T}_i$ as the set of tangent vectors at i. The direction of a vector $\vec{t}_i \in \mathcal{T}_i$ is defined as the normalized angle

$$\phi := \frac{2\pi\theta}{\Theta} \in [0, 2\pi),$$

where Θ is the sum of the angles incident to i, and θ is the angle of $\vec{t_i}$ relative to an arbitrary yet fixed reference edge ij_0. We can then encode $\vec{t_i}$ as the complex number

$$||\vec{t_i}||e^{\imath} \in \mathbb{C},$$

where $\imath$ is the imaginary unit. The angular coordinate $\phi_{ij} \in [0, 2\pi)$ corresponds to the outgoing edge from i to j. We can define $\vec{e}_{ij} \in \mathcal{T}_i$ as the tangent vector in the direction ϕ_{ij} and with the magnitude l_{ij} (the length of edge ij). At vertex j, the direction ϕ_{ij} corresponds to $\phi_{ji} + \pi$.

Thus, we can define the parallel transport of vectors in the space $\mathcal{T}_i$ to the space $\mathcal{T}_j$ via the rotation

$$\mathbf{R}_{ij} := e^{\imath((\phi_{ji}+\pi)-\phi_{ij})}.$$

Let $\hat{t}_j$ be the vector $\vec{t_i}$ parallel transported to vertex j. From our rotation, we have $\hat{t}_j = \mathbf{R}_{ij}\vec{t_i} + \vec{e}_{ji}$. Our cost computation for the updated $\tilde{t}_j$ on a surface is then

$$\tilde{t}_j = \frac{\alpha_{ij} m_i \hat{t}_j + m_j \vec{t_j}}{\alpha_{ij} m_i + m_j}.$$

[Liu+23]

2.3.3 Incorporating Curvature

Recall the definition of discrete Gaussian curvature K at a vertex, which by Theorem 2 equals the angle defect $d(i)$ (Definition 2.2.2). By the Gauss–Bonnet Theorem, the sum of all K at each vertex of a mesh is

$$\sum_{i \in V} K_i = 2\pi\chi,$$

where χ is the Euler characteristic. Since local flattening preserves χ, the curvature K_i that was at i must be redistributed across the updated curvatures of its neighbors, summing to K_i. This conservation motivates treating K as the 'mass' in the transport problem.

One snag is that K is not necessarily positive, while our mass is assumed to be positive when calculating cost — having negative mass values would lead to negative cost. Our coarsening algorithm would then prioritize removing high (negative) curvature vertices over vertices with low curvature. Moreover, having a negative mass would result in updated tangent vectors pointing away from sources of contributing mass. There are a couple things we do to avoid these issues.

First, we define the convex weights

$$\alpha_{ij} = \frac{|\tilde{K}_j - K_j|}{\sum_{l \in \mathcal{N}_i} |\tilde{K}_l - K_l|}$$

to determine what proportion of curvature is distributed from i to each of its neighbors $j \in \mathcal{N}_i$. By taking the absolute value, we guarantee that α_{ij} is nonnegative and

$\sum_{j \in \mathcal{N}_i} \alpha_{ij} = 1$. Next, we split positive and negative curvatures into two "channels," defining $K_i^+ := \max(K_i, 0)$ and $K_i^- := -\min(K_i, 0)$. Each vertex now has two mass values and two corresponding center of mass vectors, denoted $K_i^+, \vec{t_i^+}$ and $K_i^-, \vec{t_i^-}$. The final cost computation for i is then just the sum of the errors using equation 2.3:

$$C_i = \sum_{j \in \mathcal{N}_i} (\tilde{K}_j^+ ||\tilde{t}_j^+|| + \tilde{K}_j^- ||\tilde{t}_j^-||).$$

This way, we keep our mass values positive while still considering both positive and negative curvatures.

2.4 Simplification Using Intrinsic Curvature Error

Now that we are equipped with a method to quantify error associated with a vertex removal, we can construct the rest of the coarsening algorithm. First we execute an initialization step where the starting cost C_i is computed and assigned for each vertex $i \in V$. All vertices are then stored in a priority queue, keyed by their respective cost C_i. As the main loop of our algorithm, we run the following scheme until the target vertex count is met or there are no valid vertices to remove.

1. Flatten the lowest cost vertex i.
2. Remove i from the mesh.
3. Flip the remaining edges in $\mathcal{N}_i$ back to Delaunay as needed.

Pop the vertex i with the lowest cost C_i from our queue. We perform a *local flattening* operation on i's neighborhood $\mathcal{N}_i$, where we move i such that $K_i = 0$ while keeping the lengths of the boundary edges of $\mathcal{N}_i$ fixed. Doing this requires us to scale each edge ij by some factor, which can be computed via Springborn's CETM [SSP08]. Next, compute the updated curvatures of $j \in \mathcal{N}_i$.

With i locally flattened, we can now cleanly remove it from the mesh. Start by iteratively flipping the edges containing i until $\deg(i) = 3$ (i.e. i is included in exactly 3 edges). Then, remove i and its remaining edges from the mesh. Finally, flip the edges of the one-ring of the just-removed vertex i as needed to restore the intrinsic Delaunay property.

Chapter 3

Results and Conclusion

Having defined intrinsic triangulations and an intrinsic coarsening method using ICE, we can compare the results with the extrinsic coarsening method via QEM. The results from the testing done by Liu et al. will be reiterated here, alongside some potential applications of ICE. Additionally, we will discuss future directions for implementation and research.

3.1 Using the Results from ICE

Analysis from Liu et al. tests for accuracy and computation times on ICE-coarsened meshes using several common surface algorithms. Intrinsic coarsening gives a particularly impressive result for single-source geodesic distance algorithms (similar to the heat-based algorithm described in Section 1.2.1). On a mesh with 20,000 vertices, the coarsened mesh with 1,000 vertices (a 99.95% reduction) sees a 4880x speedup in computation times while only incurring 1.5% relative error [Liu+23, Section 8.5.2].

Liu et al. also finds that ICE maintains a higher relative accuracy for PDE-based algorithms compared to running the algorithm on a QEM-coarsened mesh. In one instance, a cloth mesh is reduced to a vertex count of 500 and is used for a physics simulation which involves solving a Poisson problem. The coarsened mesh computed via ICE had smoother isolines compared to the QEM-coarsened mesh, indicating a higher accuracy to the ground truth. When computing approximations for single-source geodesic distance, ICE meshes saw an approximately 4x reduction in relative error [Liu+23, Section 8.5].

3.2 Future Directions

Such a substantial increase in computation speed with minimal relative error is obviously highly sought after for nearly every geometry processing pipeline. Graph Neural Networks (GNNs), for example, train on mesh data in order to learn a mesh's "key features." Zeke Dawson's *Use of Adversarial Graph Neural Networks for MRI Super-Resolution* demonstrates how a GNN can be used to upscale low-resolution MRI inputs — one key step in training the network is reducing the input size by coarsening

the input mesh via QEM [Daw25]. While QEM is effective in reducing input size (and can be computed in parallel), the higher accuracy of an intrinsically coarsened mesh may provide more optimal training data. The incorporation of ICE coarsening as pre-processing for mesh training data seems promising in both classification and generative neural networks.

Bibliography

[AC06] Ergun Akleman and Jianer Chen. "Insight for Practical Subdivision Modeling with Discrete Gauss–Bonnet Theorem". In: *Proceedings of the 4th International Conference on Geometric Modeling and Processing*. GMP'06. Pittsburgh, PA: Springer-Verlag, 2006, pp. 287–298. ISBN: 354036711X. DOI: 10.1007/11802914_20. URL: https://doi.org/10.1007/11802914_20.

[Cas96] James Casey. *Exploring Curvature*. ger. Braunschweig: Vieweg, 1996. ISBN: 3528064757. URL: http://bvbr.bib-bvb.de:8991/F?func=service&doc_library=BVB01&doc_number=007389615&line_number=0001&func_code=DB_RECORDS&service_type=MEDIA.

[Cra25] Keenan Crane. *Discrete Differential Geometry: An Applied Introduction*. class notes for Discrete Differential Geometry. Pittsburg, Pennsylvania: Carnegie Mellon University, Spring 2025.

[CWW13] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. "Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow". en. In: *ACM Transactions on Graphics* 32.5 (Sept. 2013), pp. 1–11. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/2516971.2516977. URL: https://dl.acm.org/doi/10.1145/2516971.2516977 (visited on 03/29/2026).

[Daw25] Zeke Dawson. "Use of adversarial graph neural networks for MRI super-resolution". eng. Portland, Oregon, 2025.

[Gab06] Eric Gaba. *Minimal Surface Curvature Planes*. Licensed under CC BY-SA 4.0. Wikimedia Commons. 2006. URL: https://commons.wikimedia.org/wiki/File:Minimal_surface_curvature_planes-en.svg.

[GH97] Michael Garland and Paul S. Heckbert. "Surface Simplification Using Quadric Error Metrics". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216. ISBN: 0897918967. DOI: 10.1145/258734.258849. URL: https://doi.org/10.1145/258734.258849.

[HD06] Øyvind. Hjelle and Morten Dæhlen. *Triangulations and applications*. eng. Mathematics and visualization. Berlin: Springer-Verlag, 2006. ISBN: 354033260X. URL: http://catdir.loc.gov/catdir/enhancements/fy0824/2006928289-t.html.

[Kar14] Hermann Karcher. "Riemannian Center of Mass and So Called Karcher Mean". In: (2014). arXiv: 1407.2087 [math.HO]. URL: https://arxiv.org/abs/1407.2087.

[Lee11] John M. Lee. *Introduction to Topological Manifolds*. eng. Second. Graduate Texts in Mathematics ; 202. New York, NY: Springer, 2011 - 2011. ISBN: 9781441979391. URL: http://catdir.loc.gov/catdir/enhancements/fy1318/2011287064-t.html.

[Liu+23] Hsueh-Ti Derek Liu et al. "Surface Simplification using Intrinsic Error Metrics". en. In: *ACM Transactions on Graphics* 42.4 (Aug. 2023), pp. 1–17. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3592403. URL: https://dl.acm.org/doi/10.1145/3592403 (visited on 03/24/2026).

[PC19] Gabriel Peyré and Marco Cuturi. "Computational Optimal Transport: With Applications to Data Science". In: *Foundations and Trends® in Machine Learning* 11 (Feb. 2019), pp. 355–206. DOI: 10.1561/2200000073.

[PS20] Paolo Pellizzoni and Gianpaolo Savio. "Mesh Simplification by Curvature-Enhanced Quadratic Error Metrics". en. In: *Journal of Computer Science* 16.8 (Aug. 2020), pp. 1195–1202. ISSN: 1549-3636. DOI: 10.3844/jcssp.2020.1195.1202. URL: http://thescipub.com/abstract/10.3844/jcssp.2020.1195.1202 (visited on 09/22/2025).

[Ros97] Jarek Rossignac. "Geometric Simplification and Compression". en. In: (1997).

[SC+19] Nicholas Sharp, Keenan Crane, et al. "GeometryCentral: A modern C++ library of data structures and algorithms for geometry processing". In: (2019).

[SGC21] Nicholas Sharp, Mark Gillespie, and Keenan Crane. "Geometry Processing with Intrinsic Triangulations". en. In: *ACM SIGGRAPH 2021 Courses*. Virtual Event USA: ACM, Aug. 2021, pp. 1–1. ISBN: 978-1-4503-8361-5. DOI: 10.1145/3450508.3464592. URL: https://dl.acm.org/doi/10.1145/3450508.3464592 (visited on 04/04/2026).

[Sha+19] Nicholas Sharp et al. *Polyscope*. www.polyscope.run. 2019.

[She02] Jonathan Richard Shewchuk. "What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures". In: *11th International Meshing Roundtable, {IMR} 2002*. Ithaca, United States, 2002. URL: https://hal.science/hal-04614934.

[Shu08] Jerry Michael Shurman. *Multivariable differential calculus*. eng. Portland, Oregon: Jerry Shurman, 2008.

[SSP08] Boris Springborn, Peter Schröder, and Ulrich Pinkall. "Conformal Equivalence of Triangle Meshes". en. In: *ACM Transactions on Graphics* 27.3 (Aug. 2008), pp. 1–11. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/1360612.1360676. URL: https://dl.acm.org/doi/10.1145/1360612.1360676 (visited on 04/22/2026).

[TMF22] I Todhunter, M A, and F R S. *Spherical Trigonometry*. en. 5th ed. Outlook Verlag, 2022.

[Wee85] Jeffrey R. Weeks. *The shape of space : how to visualize surfaces and three-dimensional manifolds*. eng. Monographs and textbooks in pure and applied mathematics ; 96. New York: M. Dekker, 1985. ISBN: 082477437X. URL: http://www.ulb.tu-darmstadt.de/tocs/71865004.pdf.