Eurographics Symposium on Geometry Processing 2025
M. Attene and S. Sellán
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 5

Controlling Quadric Error Simplification with Line Quadrics

Hsueh-Ti Derek Liu, Mehdi Rahimzadeh, Victor Zordan

Roblox, USA

Abstract

This work presents a method to control the output of mesh simplification algorithms based on iterative edge collapses. Traditional mesh simplification focuses on preserving the visual appearance. Despite still being an important criterion, other geometric properties also play critical roles in different applications, such as triangle quality for computations. This motivates our work to stay under the umbrella of the popular quadric error mesh simplification, while proposing different ways to control the simplified mesh to possess other geometric properties. The key ingredient of our work is another quadric error, called line quadrics*, which can be seamlessly added to the vanilla quadric error metric. We show that, theoretically and empirically, adding our line quadrics can improve the numerics and encourage the simplified mesh to have uniformly distributed vertices. If we spread the line quadric adaptively to different regions, it can easily lead to soft preservation of feature vertices and edges. Our method is simple to implement, requiring only a few lines of code change on top of the original quadric error simplification, and can lead to a variety of user controls.*

CCS Concepts

• ***Computing methodologies*** → ***Mesh models;***

1. Introduction

Quadric error simplification [GH97] is a popular way to simplify surface triangle meshes due to its efficiency and the ability to preserve mesh appearance [HG99]. However, other geometric properties also play important roles in different applications. For instance, striving for high triangle quality is critical for rendering and simulation accuracy; preserving user-specified feature edges/vertices is an important artistic control to design levels of detail based on regions of interest. Despite the importance of controlling quadric error simplification, this topic receives far less attention.

Existing variants of quadric error simplification primarily focus on improving the numerics or preserving different surface quantities. For controlling the quadric error simplification, the few existing techniques rely scaling the quadric error based on some scalar weights [LHW10,LGC*23]. However, the concept of scaling quadric errors falls short in many scenarios. The most notable one being simplifying a planar region where the quadric error is zero everywhere, thus scaling has no influence (see Fig. 1).

In this work, our goal is to augment the quadric error simplification with different user controls. Our technical ingredient is extremely simple: an additional quadric error we called the *line quadric*. Its theoretical connection to Fréchet Means makes it a convenient tool to encourage uniform decimation, which is desirable for computation and remeshing. When adding line quadrics adaptively across the surface, they can be used to preserve features, such as joint vertices for skinned meshes. In addition, line quadrics also guarantee that the linear system for solving the optimal vertex location is always well-conditioned, avoiding the need of numerical surgery. Last but not least, incorporating line quadrics is extremely simple, requiring only ten lines of code change on top of the vanilla quadric error simplification.

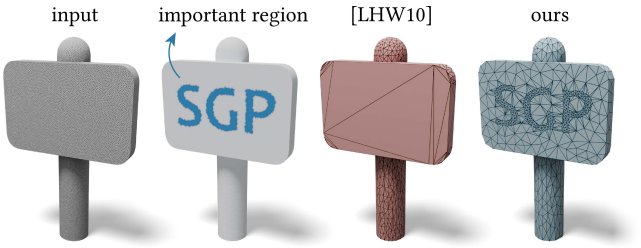


Figure 1: *Previous methods (e.g. [LHW10]) based on scaling the quadric error fail to preserve user-specified features (second) for planar regions, because scaling the initially zero quadric error has no influence. Our line quadric serves as a controllable soft constraint for the quadric error simplification, leading to a decimated mesh that preserves the feature (fourth).*

2. Related Work

Mesh simplification has been extensively studied in computer graphics to reduce the resolution of a triangle mesh while pre-

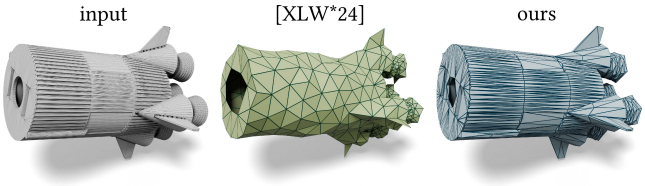


Figure 2: *Compared to clustering-based method, such as [XLW*24], our method is orders of magnitude faster. In this example, both methods decimate the mesh down to 5% of its original resolution. Our method takes less than a second, while [XLW*24] requires more than 3 minutes.*

serving the mesh appearance. Starting from prominent early examples based on optimization [HDD*92, Tur92] or decimation [SZL92], mesh simplification has been tackled from different perspectives, such as using clustering [CAD04, LT97, XLW*24, RB93] and wavelets [Sch96, PM05]. Among these techniques, perhaps the most widely used approach is to iteratively collapse mesh edges [Hop96]. Compared to alternatives, such as clustering [XLW*24], edge collapse is efficient to decimate a mesh in seconds to milliseconds and can be robust to a variety of defects (e.g., non-manifolds, multi-components, and open surfaces) [LZY24] (see Fig. 2). Among the family of edge collapse algorithms, using the *quadric error metric* [GH97, GZ05] to prioritize the order of edge collapses has been demonstrating *excellent* simplification results. This quadric error edge collapse has been extended extensively to handle surface attributes [GH98, Hop99], preserve topology [DEGN99], avoid self-intersections [GBK03], increase robustness [Lin00, CPW*23], bound the maximum geometric error [BF05], maintain segmentation boundaries [GV24], and improve the decimated triangle quality [CGG*03, TK20]. In addition, different error metrics have been proposed to maintain different geometry-related quantities, including volume/area [LT98], acoustic [LFZ15], and spectral properties [LLT*20]. We refer readers to [BKP*10] for a more comprehensive discussion on its variants and extensions.

On the application side, we focus on controlling quadric error simplification, similar to the works on scaling the error metric, as in [LHW10, LGC*23]. However, they may lead to unexpected behavior on e.g. planar regions where the error is zero to begin with. On the technical side, the closest methods to our approach are the *boundary quadric* [GH98], the *probabilistic quadric* [TK20], and the *dihedral plane quadric* [CGG*03] (adopted by [CCC*08]). They share a similar spirit as our approach, adding another quadric, to improve the decimation results. The boundary quadric [GH98] is added to the boundary edges of a triangle mesh to improve boundary preservation, but not the interior of the mesh, thus sharing similar numerical issues as the original edge quadric error [GH97]. Probabilistic quadrics [TK20] factor in the uncertainty in positions and normals, leading to a more robust behavior on noisy meshes. The dihedral plane quadric [CGG*03, CCC*08] is added to each interior edge to improve triangle quality. On planar regions, dihedral plane quadrics encourage optimal vertices to locate on the *incenter* of the triangle, while we encourage them to locate at the *centroid*. We demonstrate that our method yields better decimation results over these methods in terms of triangle quality and visual preservation (see Fig. 9).

We introduce line quadrics to *softly* preserve featured vertices, leading to improvements over hard constraints (see Fig. 5) and the method based on scaling [LHW10] (see Fig. 1). The connection of our method to Fréchet Means makes our approach an effective tool to encourage uniform decimation (see Sec. 4.3), producing better regularization behavior compared to previous regularizers [CGG*03, CCC*08, TK20]. Our method only requires a few lines of code change to benefit applications, such as skinned mesh simplification and subdivision remeshing.

3. Background

Our method offers user-controls on the quadric error edge collapse algorithm [GH97]. Here, we briefly overview the background of the quadric error metric.

Given a surface $\mathcal{M}$ embedded in $\mathbb{R}^3$, let $\mathbf{p} \in \mathcal{M}$ be a point on $\mathcal{M}$ with (unit) normal vector $\hat{\mathbf{n}}$. The tangent plane at $\mathbf{p}$ can be defined by all points $\mathbf{x} \in \mathbb{R}^3$ that satisfy

$$\hat{\mathbf{n}}^\top (\mathbf{x} - \mathbf{p}) = 0. \tag{1}$$

The quadric error measures the squared point-to-plane distance between $\mathbf{x}$ and the tangent plane at $\mathbf{p}$, which can be computed as

$$E(\mathbf{x}) = \left(\hat{\mathbf{n}}^\top (\mathbf{x} - \mathbf{p})\right)^2 \tag{2}$$

$$= \mathbf{x}^\top \mathbf{A}\mathbf{x} + 2\mathbf{b}^\top \mathbf{x} + c, \tag{3}$$

A *quadric* Q refers to the 4-by-4 matrix assembled from the above expression

$$Q = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{b} \\ \hline \mathbf{b}^\top & c \end{array}\right] = \left[\begin{array}{c|c} \hat{\mathbf{n}}\hat{\mathbf{n}}^\top & -\hat{\mathbf{n}}\hat{\mathbf{n}}^\top \mathbf{p} \\ \hline (-\hat{\mathbf{n}}\hat{\mathbf{n}}^\top \mathbf{p})^\top & \mathbf{p}^\top \hat{\mathbf{n}}\hat{\mathbf{n}}^\top \mathbf{p} \end{array}\right] \tag{4}$$

This quadric Q gives us the complete information to compute the quadric error $E(\mathbf{x}) = [\mathbf{x}, 1]^\top Q [\mathbf{x}, 1]$ with matrix multiplications in the homogeneous coordinate.

Triangle Quadrics The quadric error can be generalized to discrete triangulated 2-manifolds by defining the quadric error E_{ijk} on the plane of each triangle ijk

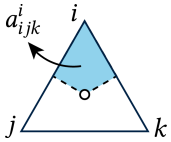
$$E_{ijk}(\mathbf{x}) = \mathbf{x}^\top \mathbf{A}_{ijk}\mathbf{x} + 2\mathbf{b}_{ijk}^\top \mathbf{x} + c_{ijk}, \tag{5}$$

with the corresponding *triangle quadric* Q_{ijk} as

$$Q_{ijk} = \left[\begin{array}{c|c} \mathbf{A}_{ijk} & \mathbf{b}_{ijk} \\ \hline \mathbf{b}_{ijk}^\top & c_{ijk} \end{array}\right] \tag{6}$$

derived from the face normal $\hat{\mathbf{n}}_{ijk}$ and a location of a point on the plane, such as one of the corner vertices $\mathbf{v}_i$.

Vertex Quadrics For each vertex i, the *vertex quadric error* E_i is defined as the weighted summation of the triangle quadric errors E_{ijk} from its one-ring triangles ijk



$$E_i(\mathbf{x}) = \sum_{ijk \in \mathcal{N}_i} a^i_{ijk} E_{ijk}(\mathbf{x}) \tag{7}$$

$$= \mathbf{x}^\top \left(\sum_{ijk \in \mathcal{N}_i} a^i_{ijk} \mathbf{A}_{ijk}\right)\mathbf{x} + 2\left(\sum_{ijk \in \mathcal{N}_i} a^i_{ijk} \mathbf{b}_{ijk}^\top\right)\mathbf{x} + \left(\sum_{ijk \in \mathcal{N}_i} a^i_{ijk} c_{ijk}\right)$$

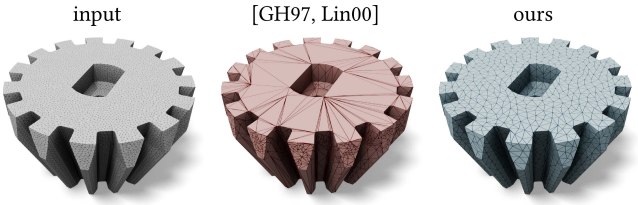


Figure 3: *Although using a robust numerical solver (e.g. [LT98]) can resolve the numerical issue from the quadric error simplification [GH97], it does not change the decimation priority, leading to random edge collapses in planar regions and low-quality triangulations (middle). Sprinkling a little bit of our line quadrics ($w_i = 10^{-4}$) not only resolves the numerical issue, it further encourages uniform decimation within planar regions (right).*

We use $\mathcal{N}_i$ to denote the one-ring triangles of the vertex i and a^i_{ijk} is a portion of the vertex area at i coming from face ijk. A common choice of area is the *barycentric area* (see inset) which simply sets $a^i_{ijk} = a_{ijk}/3$ to be one-third of the face area a_{ijk} [MDSB02]. This derivation gives rise to the definition of *vertex quadric* Q_i as a weighted summation of its one-ring triangle quadrics Q_{ijk}

$$Q_i = \sum_{ijk \in \mathcal{N}_i} a^i_{ijk} Q_{ijk} \tag{8}$$

where the summation between quadrics is the component-wise summation for all elements in the triplets.

Edge Quadrics The quadric error metric E_{ij} for each edge ij is defined as the summation of vertex quadric errors of its endpoints

$$E_{ij}(\mathbf{x}) = E_i(\mathbf{x}) + E_j(\mathbf{x}) \tag{9}$$

As we can see in Eq. 7 that summing up quadric errors leads to a summation of quadrics, this leads to the definition of an *edge quadric* Q_{ij} as

$$Q_{ij} = Q_i + Q_j. \tag{10}$$

[GH97] suggest simplifying a triangle mesh by iteratively collapsing the edge with the smallest edge quadric error $E_{ij}(\mathbf{x}^\star)$ which is the value of E_{ij} evaluated at the optimal location $\mathbf{x}^\star$ that minimizes E_{ij}. Let $Q_{ij} = [\mathbf{A}_{ij}, \mathbf{b}_{ij}; \mathbf{b}_{ij}^\top, c_{ij}]$, the optimal location $\mathbf{x}^\star$ can be computed solving a linear system obtained from setting $\nabla E_{ij} = 0$

$$\mathbf{A}_{ij}\mathbf{x}^\star = -\mathbf{b}_{ij} \tag{11}$$

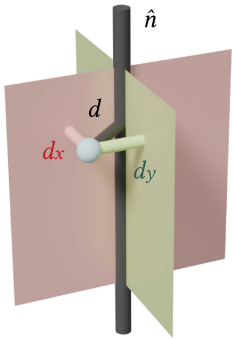
If the matrix $\mathbf{A}_{ij}$ is invertible and well-conditioned, one can solve for $\mathbf{x}^\star$ with standard linear solvers, such as Cholesky decomposition. If not, numerical surgeries, such as singular value decomposition [Lin00], are required to solve for $\mathbf{x}^\star$.

In subsequent iterations, instead of recomputing the edge quadric with Eq. 10, [GH97] recommend using the Q_{ij} as the vertex quadric for the newly inserted vertex. Suppose we collapse an edge ij to a vertex i', the quadric of the new vertex $Q_{i'} = Q_{ij}$ is simply the edge quadric before the collapse, and $Q_{i'}$ will then be used to update the quadrics for its one-ring edges. This definition allows the quadric error to "memorize" all the plane information on the *input* mesh, instead of from the current mesh. In practice, compared to recomputing the edge quadric Q_{ij} from scratch (the so-called "memoryless" implementation), such an accumulation often leads to a more efficient and desirable decimation.

4. Method

Our objective is to stay under the umbrella of quadric error simplification while adding user controls for different situations. Our core ingredient is an extra quadric error named *line quadrics*. In this section, we will introduce the definition of line quadric, its theoretical properties, and the full implementation on how to easily incorporate this change to the original quadric error mesh simplification method [GH97].

4.1. Line Quadrics



Given a surface $\mathcal{M} = (\mathbf{V}, \mathbf{F})$ discretized as a triangle mesh with a set of vertices $\mathbf{V}$ and faces $\mathbf{F}$. We define the line quadric at each vertex i as the squared point-to-line distance where the line passes through vertex i with its direction parallel to the vertex normal. To construct the line quadric, we draw inspiration from the *Pythagorean theorem* which states that the squared point-to-line distance d can be computed by summing up the squared distances from the point to two orthonormal planes $d^2 = d_x^2 + d_y^2$ passing through the line (see inset).

Specifically, given a vertex i with position $\mathbf{v}_i$, we first define the vertex normal $\hat{\mathbf{n}}_i$ as the face area weighted normal

$$\mathbf{n}_i = \frac{\sum_{ijk \in \mathcal{N}_i} a_{ijk} \hat{\mathbf{n}}_{ijk}}{\| \sum_{ijk \in \mathcal{N}_i} a_{ijk} \hat{\mathbf{n}}_{ijk} \|}, \tag{12}$$

where $\mathcal{N}_i$ denotes the neighboring faces of vertex $i \in \mathbf{V}$. Then we use the Gram–Schmidt process to construct two orthonormal vectors $\mathbf{x}_i, \mathbf{y}_i \perp \hat{\mathbf{n}}_i$ that are orthonormal to the vertex normal $\hat{\mathbf{n}}_i$. These two bases $\mathbf{x}_i, \mathbf{y}_i$ allow us to construct line quadrics as a summation of two plane quadrics as

$$\boxed{Q_i^l = Q_i^{\mathbf{x}} + Q_i^{\mathbf{y}}} \tag{13}$$

where

$$Q_i^{\mathbf{x}} = \left[\begin{array}{c|c} \mathbf{x}_i\mathbf{x}_i^\top & -\mathbf{x}_i\mathbf{x}_i^\top\mathbf{v}_i \\ \hline -(\mathbf{x}_i\mathbf{x}_i^\top\mathbf{v}_i)^\top & \mathbf{v}_i^\top\mathbf{x}_i\mathbf{x}_i^\top\mathbf{v}_i \end{array}\right] \tag{14}$$

$$Q_i^{\mathbf{y}} = \left[\begin{array}{c|c} \mathbf{y}_i\mathbf{y}_i^\top & -\mathbf{y}_i\mathbf{y}_i^\top\mathbf{v}_i \\ \hline -(\mathbf{y}_i\mathbf{y}_i^\top\mathbf{v}_i)^\top & \mathbf{v}_i^\top\mathbf{y}_i\mathbf{y}_i^\top\mathbf{v}_i \end{array}\right] \tag{15}$$

4.2. Implementation

We incorporate the line quadric to the quadric error simplification by simply adding them when constructing the decimation priority

queue (see Sec. 3). Specifically, our augmented vertex quadric $\tilde{Q}_i$ consists of the original vertex quadric Q_i in Eq. 8 and our line quadric Q_i^l

$$\tilde{Q}_i = Q_i + w_i a_i Q_i^l \tag{16}$$

where $w_i > 0$ is a user-specified positive weight on the line quadric for vertex i, $a_i = \sum_{ijk} a_{ijk}^i$ is the *barycentric vertex area* mentioned in Sec. 3. Implementing this only requires a few lines of code changes on top of a quadric error edge collapse implementation. The simplicity of our method enables us to provide the full implementation for high reproducibility:

```
# V: (#V, 3) matrix of vertex locations
# N: (#V, 3) matrix of vertex normals
# a: (#V,) vector of barycentric vertex areas
# w: (#V,) vector of line quadric weights per vertex
# Q: (#V, 4, 4) tensor of vertex quadrics

def compute_line_quadric(vi, ni):
    # compute orthogonal bases using Gram Schmidt
    xi = GramSchmidt(ni)
    xi = xi / norm(xi)
    yi = cross(xi, ni)

    # compute plane quadrics
    px = [xi[0], xi[1], xi[2], -dot(xi,vi)]
    py = [yi[0], yi[1], yi[2], -dot(yi,vi)]
    Qx = outer(px, px)
    Qy = outer(py, py)
    return Qx + Qy
...
# when constructing vertex quadrics ...
for each vertex i:
    Ql_i = compute_line_quadric(V[i,:], N[i,:])
    Q[i,:,:] += w[i] * a[i] * Ql_i
```

where the `GramSchmidt` stands for the (modified) Gram-Schmidt process [Bjö94] which computes a vector `xi` that is orthogonal to `ni`. Our implementation ensures that the vertex quadric $\tilde{Q}_i$ is always well-conditioned when $w_i > 0$. This is because, by construction, the vertex quadric is always a summation of three non-parallel plane quadrics. This leads to a well-conditioned system when solving for the optimal vertex location (see Eq. 11), because three planes only have a single intersection point. This property allows one to regularize quadric error simplification by simply adding line quadrics, without the need of other numerical tools, such as [Lin00]. If the weight w_i is small, it can guarantee numerical robustness without sacrificing feature preservation (see Fig. 3). Empirically, we recommend setting $w_i < 10^{-1}$ (our default is $w_i = 10^{-3}$) to avoid deteriorating visual preservation (see Fig. 4). If one is interested in applying a higher weight for adaptive simplification (see Fig. 5), we refer readers to Sec. 4.4.1 for a small variant of our implementation. In practice, we may encounter edge cases where neither the line quadrics nor the original quadric error [GH97] are well-defined due to degenerated elements. In such cases, we simply replace them with a tiny amount (10^{-6}) of probabilistic point quadric in [TK20] to avoid numerical issues. In terms of the performance, as our method only changes the precomputation about vertex quadrics, we enjoy the same complexity and efficient runtime as the original quadric error simplification.

4.3. Connection to Fréchet Means

Incorporating line quadrics encourages the simplified mesh to have a uniform distribution of vertices:

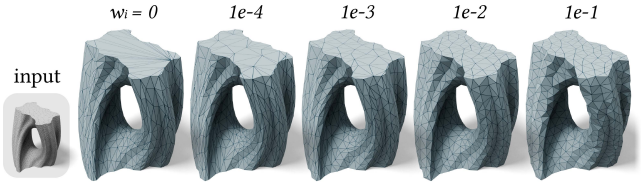


Figure 4: *Increasing the line quadric weights w_i leads to more uniform triangulations, but may lead to feature degradation in order to hit the target resolution (right). When $w_i = 0$ (left), our method falls back to the original quadric error simplification [GH97].*

Lemma 4.1 *Quadric error simplification with line quadrics places vertices uniformly on planar regions.*

The above lemma can be easily illustrated once we understand a few concepts. On a planar region, our augmented vertex quadric $\tilde{Q}_i$ in Eq. 16 is equivalent to measuring squared point-to-point distances in 2D. Such a "dimensionality reduction" comes from the fact that those quadric lines are orthogonal to the quadric plane by definition. Thus, the optimal vertex location after minimizing $\tilde{Q}_i$ always lies on the plane, making line quadrics become a point-to-point squared distance measure in 2D on planar regions. As discussed in Sec. 3, the (memory) quadric error simplification uses *addition* to accumulate vertex quadrics after each edge collapse. This implies that the optimal vertex location computed from Eq. 11 later on in the simplification minimizes the sum of squared point-to-point distances.

This property coincides with the definition of *Fréchet means*, which is a generalization of centroids to a metric space. Specifically, given a distance measure $d(\mathbf{x}, \mathbf{v}_i) \in \mathbb{R}_+$ between two points $\mathbf{x}, \mathbf{v}_i$, Fréchet mean $\mathbf{m}$ is the minimizer of

$$\mathbf{m} = \arg\min_{\mathbf{x}} \sum_i d^2(\mathbf{x}, \mathbf{v}_i). \tag{17}$$

where $\mathbf{v}_i$ are all points in the space. This definition is precisely the outcome of minimizing the line quadric error which uses the *memory* implementation on planar regions.

This connection to Fréchet means, when "translated" to practical behavior, means that the vertex locations after simplification will be placed at a weighted average location where the weights depend on the vertex area and the user-specified line quadric weight. When applying uniform line quadric weights, this leads to a triangulation with a uniform vertex distribution, which often has high triangle quality. For non-planar meshes, line quadrics still encourage uniform decimation by striking a balance between Fréchet means and curvature preservation from the original vertex quadric (see Fig. 4).

4.4. Variants of Line Quadrics

Line quadrics in Sec. 4.1 serves as our foundation for controlling the quadric error simplification. To broaden the type of controls, we describe a few variants in this section.

4.4.1. Extreme Weights

Adding our line quadrics to a simplification method can be perceived as adding a *regularizer* to adjust the decimation result (see

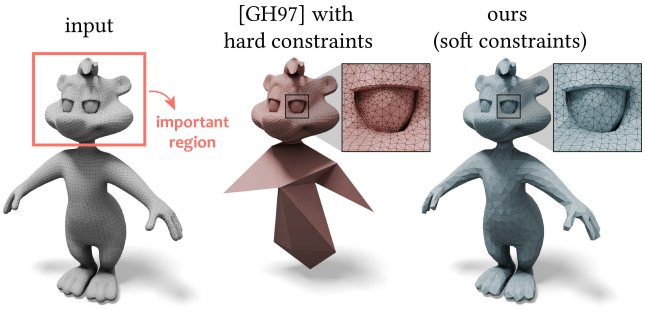


Figure 5: *Given a mesh with important regions specified by an user (left), naively treating important vertices as hard constraints leads to overly dense vertices around the important part (middle). Our method utilizes line quadrics to softly preserve important vertices, striking a balance between vertex preservation and visual quality.*

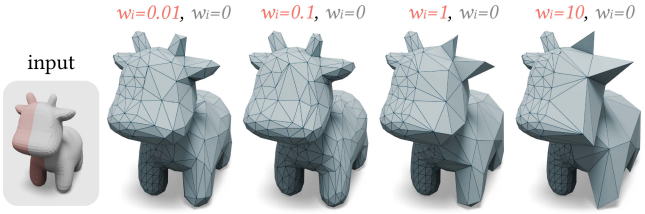


Figure 6: *We illustrate the influence of applying extremely high line quadric weights by putting high weights w_i on the left half of the mesh (left, red) and zero weight on the right half (left, blue). We can observe vertices augmented with line quadrics (left half) lead increased "importance" in the priority queue, resulting in decimation happened later on in the simplification process. In this example, we simplify the mesh down to the same number of vertices in total. Such an importance control with w_i leads to more and more adaptive triangulation when the difference in w_i between left and right increases. Note that this implementation applies a mixture of addition and scaling, explained in Sec. 4.2.*

Eq. 16). However, just like other regularizers, when the regularizer outweighs the original error metric (by applying high w_i), it may deteriorate the visual appearance.

This motivates the first variant of our method designed specifically for high line quadric weights w_i, such as for highly adaptive simplification in Fig. 5. We recommend adding a small amount of line quadrics e.g. $w_i = 0.01$, and then applying scaling to the entire augmented quadric $\tilde{Q}_i$ in Eq. 16. Adding line quadrics before scaling can resolve the issue of direct scaling [LHW10] for planar regions (see Fig. 1) because every edge will have non-zero quadric error. The result after weighting still depends on the local curvature variations, but because extreme weights dominate the decimation priority, we can still achieve effective user controls (see Fig. 6, 7).

4.4.2. Edge Quantities

Our line quadrics is convenient to work with quantities defined on vertices. In some applications, when one wants to specify importance in terms of edge quantities, the same spirit of defining line quadrics can be generalized to edges, leading to an idea similar to the method by [GH98] for preserving boundary edges

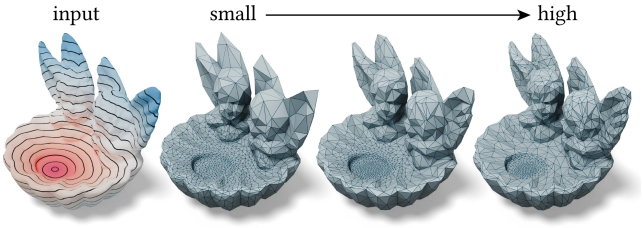


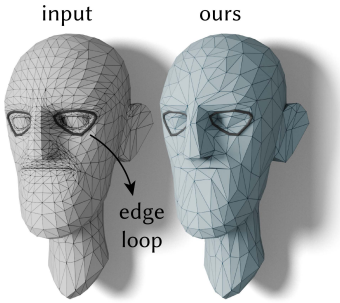
Figure 7: *We illustrate the influence of applying extreme weights that is proportional to the exponential of negative geodesic distances ($e^{-\alpha d}$), scaled by a constant factor α. In this figure, we show that by increasing α, our method can lead to adaptive triangulations that are denser near the source, and vice versa.*

and [CGG*03] for improving triangle quality. Specifically, one can compute the plane that goes through the edge ij whose plane normal $\hat{\mathbf{n}}_{ij}^{\perp}$ is orthogonal to the face area weighted edge normal $\hat{\mathbf{n}}_{ij}$

$$\hat{\mathbf{n}}_{ij} = \frac{\sum_{ijk\in\mathcal{N}_{ij}} a_{ijk}\hat{\mathbf{n}}_{ijk}}{\|\sum_{ijk\in\mathcal{N}_{ij}} a_{ijk}\hat{\mathbf{n}}_{ijk}\|}. \tag{18}$$

Given a plane normal $\hat{\mathbf{n}}_{ij}^{\perp}$ and one of the end vertices $\mathbf{v}_i$ (or $\mathbf{v}_j$), one can build the so-called dihedral plane quadric Q_{ij}^n with the formula in Eq. 4. This Q_{ij}^n can then be added to the vertex quadric

$$\tilde{Q}_i = Q_i + \sum_{ij\in\mathcal{N}_i} w_{ij}\frac{a_{ij}}{2}Q_{ij}^n \tag{19}$$



where $w_{ij} > 0$ is the user-specified weight and we use $ij \in \mathcal{N}_i$ to denote the one-ring edges of vertex i. a_{ij} is the barycentric edge area which is computed by summing up one third of the triangle areas incident to edge ij. Some implementations (e.g., [CGG*03]) replace a_{ij} with the cosine of dihedral angle. With the dihedral plane quadrics Q_{ij}^n, we can (softly) preserve important edges during simplification, such as the edge loops in the inset.

An interesting observation is that, this dihedral plane quadric can also be used to regularize quadric error simplification, just like the line quadric. But, empirically, we notice that this does not achieve as ideal results as line quadrics. As shown in Fig. 9, we apply the same weight w_i to line quadrics and dihedral plane quadrics (implemented by [CCC*08]) uniformly across the mesh. Line quadrics outperform the dihedral plane quadrics in geometric error/quality measures. Therefore, we still use line quadrics for regularizing and controlling the simplification, and reserve dihedral plane quadrics only for edge-specific controls.

5. User Controls & Applications

In this section, we demonstrate how our line quadrics can benefit quadric error simplification (see Fig. 8) and be used to tailor the simplification results for different situations.

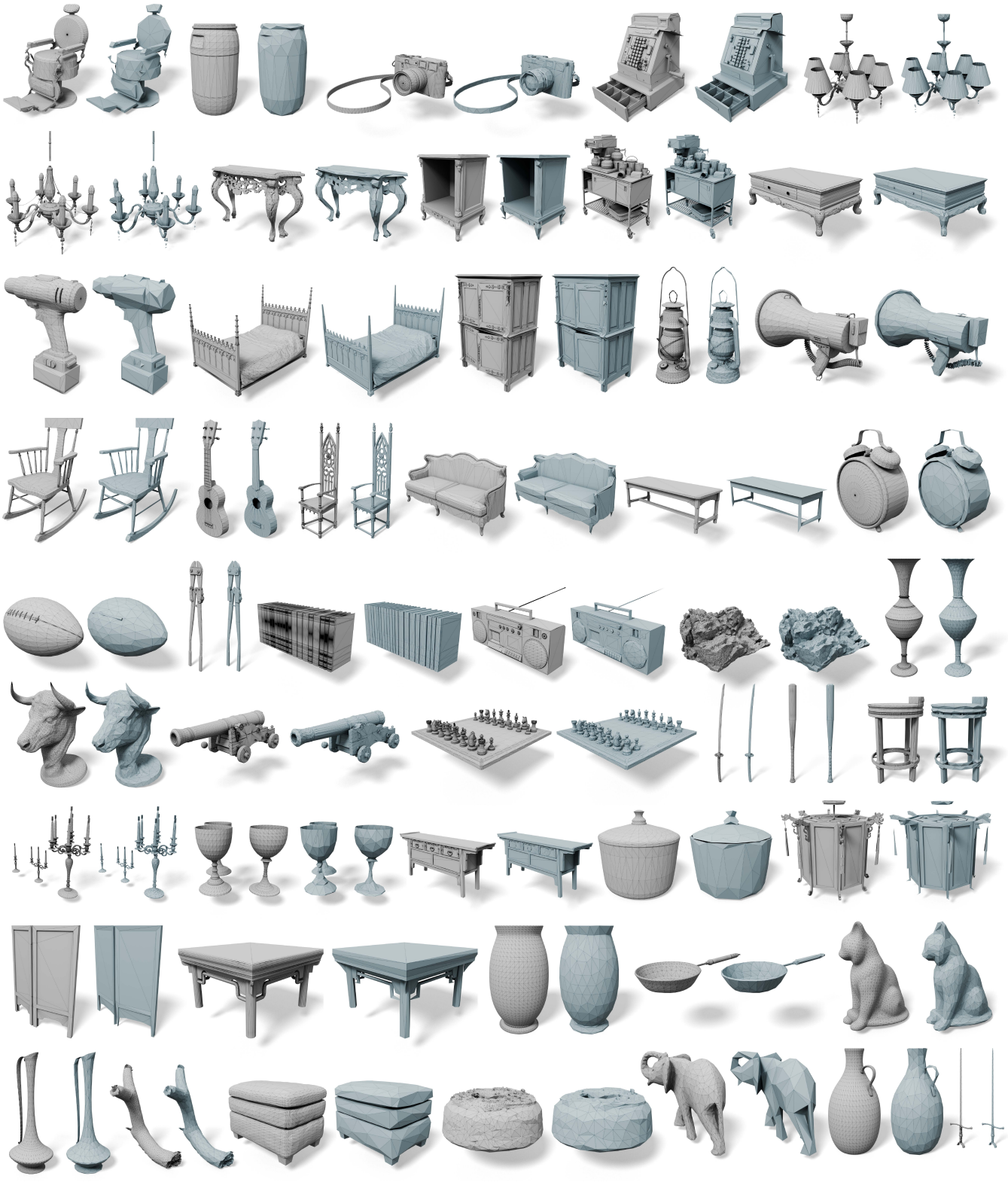


Figure 8: *Our line quadric can be seamlessly incorporated to other mesh simplification framework e.g. [LZY24] for robust simplification. We demonstrate that this combination can decimate the entire PolyHaven dataset down to 10% of their original resolution.*

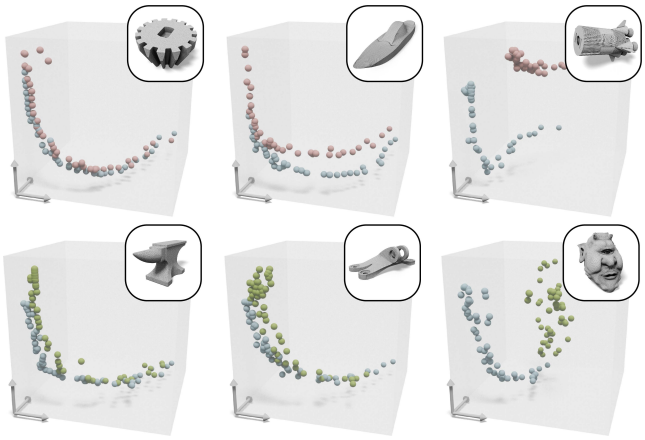


Figure 9: *We compare our method against the dihedral plane quadrics [CGG*03] (implemented in MeshLab [CCC*08]) and the probabilistic quadrics [TK20] by showing that our method contains* Pareto optimal *solutions measured by the Hausdorff distance, average point-to-mesh distance, and the triangle quality. Each point is a simplification result from one of the line quadric (or standard deviation) parameters. The point coordinates are the normalized metrics to the range of* $0, 1$ *(note that we flip the sign of triangle quality so that the lower is better). One can observe that our results (blue dots) are on the Pareto front (closer to the origin) compared to the probabilistic (green) and the dihedral plane quadrics (red).*

Figure 10: *Compared to the probabilistic quadrics [TK20], we demonstrate that our method leads to better results by showing that our method contains* Pareto optimal *solutions measured by the Hausdorff distance (Haus), average point-to-mesh distance (P2M), and the triangle quality (Tri.Q).*

5.1. Planar Simplification

The immediate benefit of adding our line quadrics is to resolve the numerical issue and improve triangle quality when decimating planar regions. It is well-known that planar regions will lead to numerical issues in quadric error simplification because there are infinite solutions which can minimize the vanilla quadric error (see Sec. 3). Early attempts treat this as a numerical problem and tackle them with a more robust solver, such as the singular value decomposition [Lin00]. This alleviates the numerical issue, but it does not improve the decimation quality as the simplification method will still pick edges randomly for planar regions (see Fig. 3). In contrast, adding a little bit of line quadrics, meaning small w_i, to all vertices is guaranteed to remove the numerical issue (see Sec. 4.2) and encourages uniform decimation (see Sec. 4.3).

Another previous method to improve planar region simplification is the *probabilistic quadric* [TK20] which treats each quadric as a sample from a Gaussian distribution. This is equivalent to minimize the quadric error with the generalized Tikhonov regularization (see Sec. 3 in [TK20]). A comparison between probabilistic quadrics and our method is not trivial because putting a high weight on probabilistic or line quadrics leads to a better triangle quality, at the cost of increased geometric error. Since an equivalence mapping between the amount of Gaussian noise and our line quadric weight does not exist, we instead demonstrate that our parameter space contains *Pareto optimal* solutions with respect to common geometric metrics, including the average point-to-mesh distance, Hausdorff distance, and the triangle quality (see the Appendix C of [LKC*20]). Specifically, we perform a parameter sweep on different line quadric weights for our method and different standard deviations for [TK20] (ranging from $[10^{-4}, 10^{1}]$). We show that our method leads to Pareto optimal results under all metrics (see Fig. 9). We also show a qualitative example in Fig. 10.

5.2. Subdivision Remeshing

Remeshing a mesh to have subdivision connectivity is a key ingredient to multiresolution shape analysis, such as wavelet compression [GSS99] and multiresolution modeling [BK04]. Recently, subdivision remeshing has also been applied to generate training data for neural networks [LKC*20] or facilitate neural network architectures defined on triangle meshes [HLG*22, YDS*23]. Although different applications have slightly different definitions for a "good" subdivision remeshing result, a common objective is to (1) preserve the input geometry in order to minimize the area/angle distortion, and (2) have uniform triangulations for better training outcomes. In Fig. 11, we demonstrate that our line quadric can immediately serve as a tool to strike a balance between feature preservation and uniform triangulation, leading to more desirable remeshing outcomes.

5.3. Attribute-Aware Simplification

Applying different weights w_i in Eq. 16 to different vertices allows us to specify important vertices that need to be maintained "softly" throughout the decimation. Compared to, for instance, treating these vertices as hard constraints, our method can be viewed as applying soft constraints to strike a better balance between feature and appearance preservation (see Fig. 5).

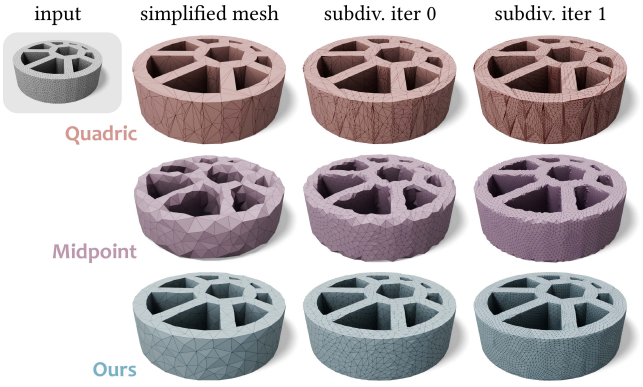


Figure 11: *If we use the quadric error [GH97] alone to remesh a surface to have Loop subdivision connectivity, as in [LKC*20], this lead to bad quality triangles on planar regions (top). If we use mid-point edge collapse (e.g., [LZBCJ21]), it can achieve uniform triangulation, but at the cost of missing sharp features (middle). In contrast, our method regularizes the quadric error simplification and leads to a feature-preserving uniform triangulation.*

This type of control is useful for decimating meshes with surface attributes that cannot be inferred from the geometry alone. For instance, in the context of simplifying skinned meshes, preserving vertices near joints is important to pose the simplified character afterwards. Previous works "translate" the skinning information to actual geometries by deforming the skinned mesh based on user-provided poses [TGBE16, LS09] or sampled poses from user-provided joint information [DR05]. These posed geometries can then be used to define an error metric that simultaneously minimizes the error against all these posed meshes. In contrast, our method loosens the input requirement by only requiring skinning information, without any example poses or joint information. In Fig. 12, we compute a set of line quadric weights based on the skinning weights. Specifically, we define the weight of a vertex i as a heuristic function $w_i = c(1 - s_i)$, where s_i is the maximum skinning weight value at a vertex and $c = 0.15$ is a global scaling term to make sure that $w_i \leq 0.1$. Intuitively, this weight definition assigns higher weights on near joint vertices which are shared by multiple bones, and vice versa. With such a definition, we can preserve vertices near joints in order to realize target animations on the decimated mesh (see Fig. 13).

6. Conclusion & Future Work

Our work focuses on a practical aspect of quadric error simplification – user controls. Mesh simplification has been widely used in different applications, which desire different properties on the simplified meshes. Our proposed line quadrics and its variants enable one to control triangle quality, preserve featured vertices/edges, and simplify a mesh adaptively. We demonstrate a handful of situations that favor different user controls. However, at the presence of noise, line quadrics cannot serve as a denoiser because it respects the vertex normals of the input mesh (see Fig. 14). Future explorations on more applications could understand the limits of line quadrics and inspire new ways to control simplification.

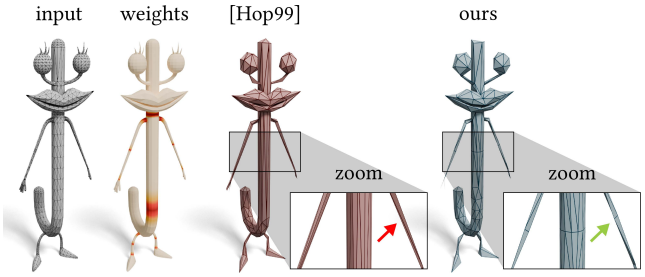


Figure 12: *Given a skinned mesh (first), we specify high line quadric weights near joint vertices according to the skinning information (second). Traditional method, such as [Hop99], fails to consider skinning weights, leading to decimating near joint vertices (third). In contrast, our method better preserves joint vertices, making it more suitable for downstream deformations.*

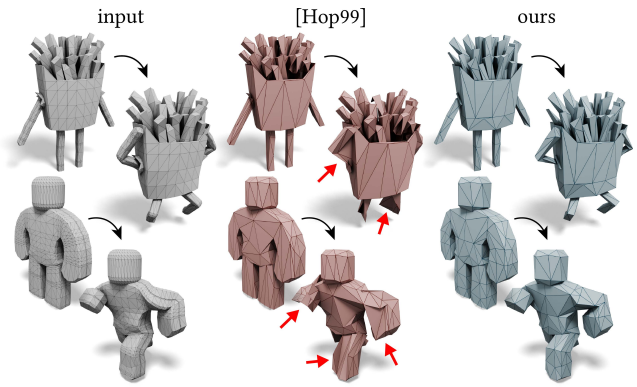


Figure 13: *When simplifying the character meshes in the rest post (left), we add line quadrics for near joint vertices, inferred from skinning weights. We can observe that our method (blue) preserves joint vertices and lead to better deformation results (right). In contrast, a variant of quadric error simplification that preserves attributes [Hop99] fails to preserve joint vertices (red) and results in suboptimal deformation.*

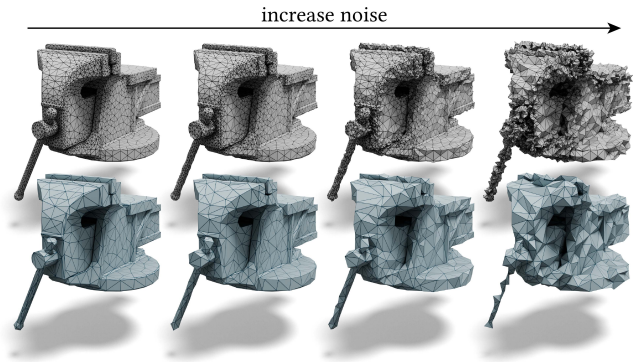


Figure 14: *In the presence of minor noise, mesh simplification with our line quadric could achieve reasonable results. However, at the presence of high noise (right) where the vertex normal become extremely noisy, line quadric can not serve as a denoising technique.*

References

[BF05] BOROUCHAKI H., FREY P.: Simplification of surface mesh using hausdorff envelope. *Computer methods in applied mechanics and engineering 194*, 48-49 (2005), 4864–4884. 2

[Bjö94] BJÖRCK Å.: Numerics of gram-schmidt orthogonalization. *Linear Algebra and Its Applications 197* (1994), 297–316. 4

[BK04] BOTSCH M., KOBBELT L.: A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2004), pp. 185–192. 7

[BKP*10] BOTSCH M., KOBBELT L., PAULY M., ALLIEZ P., LÉVY B.: *Polygon Mesh Processing*. A K Peters, 2010. 2

[CAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *ACM Trans. Graph. 23*, 3 (2004), 905–914. 2

[CCC*08] CIGNONI P., CALLIERI M., CORSINI M., DELLEPIANE M., GANOVELLI F., RANZUGLIA G., ET AL.: Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference* (2008), vol. 2008, Salerno, pp. 129–136. 2, 5, 7

[CGG*03] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: BDAM - batched dynamic adaptive meshes for high performance terrain visualization. *Comput. Graph. Forum 22*, 3 (2003), 505–514. 2, 5, 7

[CPW*23] CHEN Z., PAN Z., WU K., VOUGA E., GAO X.: Robust low-poly meshing for general 3d models. *ACM Trans. Graph. 42*, 4 (2023), 119:1–119:20. 2

[DEGN99] DEY T., EDELSBRUNNER H., GUHA S., NEKHAYEV D.: Topology preserving edge contraction. *Publications de l'Institut Mathématique 66* (1999). 2

[DR05] DECORO C., RUSINKIEWICZ S.: Pose-independent simplification of articulated meshes. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics, SI3D 2005, April 3-6, 2005, Washington, DC, USA* (2005), Lastra A., Olano M., Luebke D. P., Pfister H., (Eds.), ACM, pp. 17–24. 8

[GBK03] GUMHOLD S., BORODIN P., KLEIN R.: Intersection free simplification. *Int. J. Shape Model. 9*, 2 (2003), 155–176. 2

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997, Los Angeles, CA, USA, August 3-8, 1997* (1997), Owen G. S., Whitted T., Mones-Hattal B., (Eds.), ACM, pp. 209–216. 1, 2, 3, 4, 8

[GH98] GARLAND M., HECKBERT P. S.: Simplifying surfaces with color and texture using quadric error metrics. In *9th IEEE Visualization Conference, IEEE Vis 1998, Research Triangle Park, North Carolina, USA, October 18-23, 1998, Proceedings* (1998), Ebert D. S., Rushmeier H. E., Hagen H., (Eds.), IEEE Computer Society and ACM, pp. 263–269. 2, 5

[GSS99] GUSKOV I., SWELDENS W., SCHRÖDER P.: Multiresolution signal processing for meshes. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1999, Los Angeles, CA, USA, August 8-13, 1999* (1999), Waggenspack W. N., (Ed.), ACM, pp. 325–334. 7

[GV24] GRZECZKOWICZ G., VALLET B.: Semantic edge collapse: A mesh edge collapse algorithm preserving per face semantic information. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 48* (2024), 185–192. 2

[GZ05] GARLAND M., ZHOU Y.: Quadric-based simplification in any dimension. *ACM Trans. Graph. 24*, 2 (2005), 209–239. 2

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1992, Chicago, IL, USA, July 27-31, 1992* (1992), Thomas J. J., (Ed.), ACM, pp. 71–78. 2

[HG99] HECKBERT P. S., GARLAND M.: Optimal triangulation and quadric-based surface simplification. *Comput. Geom. 14*, 1-3 (1999), 49–65. 1

[HLG*22] HU S., LIU Z., GUO M., CAI J., HUANG J., MU T., MARTIN R. R.: Subdivision-based mesh convolution networks. *ACM Trans. Graph. 41*, 3 (2022), 25:1–25:16. 7

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996* (1996), Fujii J., (Ed.), ACM, pp. 99–108. 2

[Hop99] HOPPE H.: New quadric metric for simplifying meshes with appearance attributes. In *10th IEEE Visualization Conference, IEEE Vis 1999, San Francisco, CA, USA, October 24-29, 1999, Proceedings* (1999), Ebert D. S., Gross M. H., Hamann B., (Eds.), IEEE Computer Society and ACM, pp. 59–66. 2, 8

[LFZ15] LI D., FEI Y. R., ZHENG C.: Interactive acoustic transfer approximation for modal sound. *ACM Trans. Graph. 35*, 1 (2015), 2:1–2:16. 2

[LGC*23] LIU H. D., GILLESPIE M., CHISLETT B., SHARP N., JACOBSON A., CRANE K.: Surface simplification using intrinsic error metrics. *ACM Trans. Graph. 42*, 4 (2023), 118:1–118:17. 1, 2

[LHW10] LI L., HE M., WANG P.: Mesh simplification algorithm based on absolute curvature-weighted quadric error metrics. In *2010 5th IEEE Conference on Industrial Electronics and Applications* (2010), IEEE, pp. 399–403. 1, 2, 5

[Lin00] LINDSTROM P.: Out-of-core simplification of large polygonal models. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000, New Orleans, LA, USA, July 23-28, 2000* (2000), Brown J. R., Akeley K., (Eds.), ACM, pp. 259–262. 2, 3, 4, 7

[LKC*20] LIU H. D., KIM V. G., CHAUDHURI S., AIGERMAN N., JACOBSON A.: Neural subdivision. *ACM Trans. Graph. 39*, 4 (2020), 124. 7, 8

[LLT*20] LESCOAT T., LIU H. D., THIERY J., JACOBSON A., BOUBEKEUR T., OVSJANIKOV M.: Spectral mesh simplification. *Comput. Graph. Forum 39*, 2 (2020), 315–324. 2

[LS09] LANDRENEAU E., SCHAEFER S.: Simplification of articulated meshes. *Comput. Graph. Forum 28*, 2 (2009), 347–353. 8

[LT97] LOW K., TAN T. S.: Model simplification using vertex-clustering. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics, SI3D '97, Providence, RI, USA, April 27-30, 1997* (1997), van Dam A., (Ed.), ACM, pp. 75–82, 188. 2

[LT98] LINDSTROM P., TURK G.: Fast and memory efficient polygonal simplification. In *9th IEEE Visualization Conference, IEEE Vis 1998, Research Triangle Park, North Carolina, USA, October 18-23, 1998, Proceedings* (1998), Ebert D. S., Rushmeier H. E., Hagen H., (Eds.), IEEE Computer Society and ACM, pp. 279–286. 2, 3

[LZBCJ21] LIU H.-T. D., ZHANG J. E., BEN-CHEN M., JACOBSON A.: Surface multigrid via intrinsic prolongation. *ACM Trans. Graph. 40*, 4 (2021). 8

[LZY24] LIU H. D., ZHANG X., YUKSEL C.: Simplifying triangle meshes in the wild. *CoRR abs/2409.15458* (2024). 2, 6

[MDSB02] MEYER M., DESBRUN M., SCHRÖDER P., BARR A. H.: Discrete differential-geometry operators for triangulated 2-manifolds. In *Third International Workshop "Visualization and Mathematics", VisMath 2002, Berlin, Germany, May 22-25, 2002* (2002), Hege H., Polthier K., (Eds.), Mathematics and Visualization, Springer, pp. 35–57. 3

[PM05] PEYRÉ G., MALLAT S.: Surface compression with geometric bandelets. *ACM Trans. Graph. 24*, 3 (2005), 601–608. 2

[RB93] ROSSIGNAC J., BORREL P.: Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in Computer Graphics, Methods and Applications [selection of papers from the conference held at Genoa, Italy, on June 28-July 1, 1993]* (1993), Falcidieno B., Kunii T. L., (Eds.), IFIP Series on Computer Graphics, Springer, pp. 455–465. 2

[Sch96] SCHRÖDER P.: Wavelets in computer graphics. *Proc. IEEE 84*, 4 (1996), 615–625. 2

[SZL92] SCHROEDER W. J., ZARGE J. A., LORENSEN W. E.: Decimation of triangle meshes. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1992, Chicago, IL, USA, July 27-31, 1992* (1992), Thomas J. J., (Ed.), ACM, pp. 65–70. 2

[TGBE16] THIERY J., GUY E., BOUBEKEUR T., EISEMANN E.: Animated mesh approximation with sphere-meshes. *ACM Trans. Graph. 35*, 3 (2016), 30:1–30:13. 8

[TK20] TRETTNER P., KOBBELT L.: Fast and robust QEF minimization using probabilistic quadrics. *Comput. Graph. Forum 39*, 2 (2020), 325–334. 2, 4, 7

[Tur92] TURK G.: Re-tiling polygonal surfaces. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1992, Chicago, IL, USA, July 27-31, 1992* (1992), Thomas J. J., (Ed.), ACM, pp. 55–64. 2

[XLW*24] XU R., LIU L., WANG N., CHEN S., XIN S., GUO X., ZHONG Z., KOMURA T., WANG W., TU C.: CWF: consolidating weak features in high-quality mesh simplification. *ACM Trans. Graph. 43*, 4 (2024), 80:1–80:14. 2

[YDS*23] YUAN S., DOU Y., SHI R., NI B., ZHENG Z.: Sievenet: Selecting point-based features for mesh networks. *CoRR abs/2308.12530* (2023). arXiv:2308.12530. 7