

Simplifying Textured Triangle Meshes in the Wild

HSUEH-TI DEREK LIU, Roblox, Canada

XIAOTING ZHANG, Roblox, USA

CEM YUKSEL, University of Utah & Roblox, USA

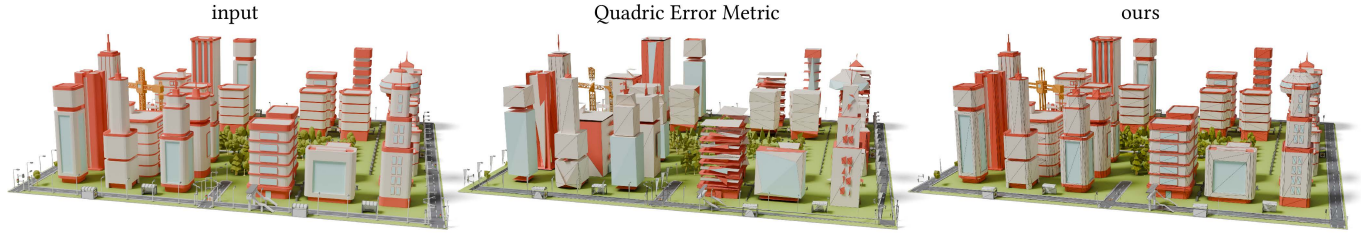


Fig. 1. Given an artist-created city scene (© mertkilić) in the wild, we simplify it down to 10% of its original resolution. Our method (right) better preserves geometrically significant components (e.g., buildings) compared to the quadric error simplification [Garland and Heckbert 1998], implemented in [Cignoni et al. 2008], (middle) which prioritizes small components (e.g., streetlamps).

This paper introduces a method for simplifying textured surface triangle meshes in the wild while maintaining high visual quality. While previous methods achieve excellent results on *manifold* meshes by using the quadric error metric, they struggle to produce high-quality outputs for meshes in the wild, which typically contain *non-manifold* elements and multiple connected components. In this work, we propose a method for simplifying these “wild” textured triangle meshes. We formulate mesh simplification as a problem of decimating *simplicial 2-complexes* to handle multiple non-manifold mesh components collectively. Building on the success of quadric error simplification, we iteratively collapse 1-simplices (vertex pairs). Our approach employs a modified quadric error that converges to the original quadric error metric for watertight manifold meshes, while significantly improving the results on wild meshes. For textures, instead of following existing strategies to preserve UVs, we adopt a novel perspective which focuses on computing mesh correspondences throughout the decimation, independent of the UV layout. This combination yields a textured mesh simplification system that is capable of handling arbitrary triangle meshes, achieving to high-quality results on wild inputs without sacrificing the excellent performance on clean inputs. Our method guarantees to avoid common problems in textured mesh simplification, including the prevalent problem of *texture bleeding*. We extensively evaluate our method on multiple datasets, showing improvements over prior techniques through qualitative, quantitative, and user study evaluations.

ACM Reference Format:

Hsueh-Ti Derek Liu, Xiaoting Zhang, and Cem Yuksel. 2025. Simplifying Textured Triangle Meshes in the Wild. *ACM Trans. Graph.* 44, 6 (December 2025), 16 pages. <https://doi.org/10.1145/3763277>

Authors' addresses: Hsueh-Ti Derek Liu, Roblox, Canada, hsuehtil@gmail.com; Xiaoting Zhang, Roblox, USA, xzhang@roblox.com; Cem Yuksel, University of Utah & Roblox, USA, cem@cemyuksel.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM. 0730-0301/2025/12-ART \$15.00

<https://doi.org/10.1145/3763277>

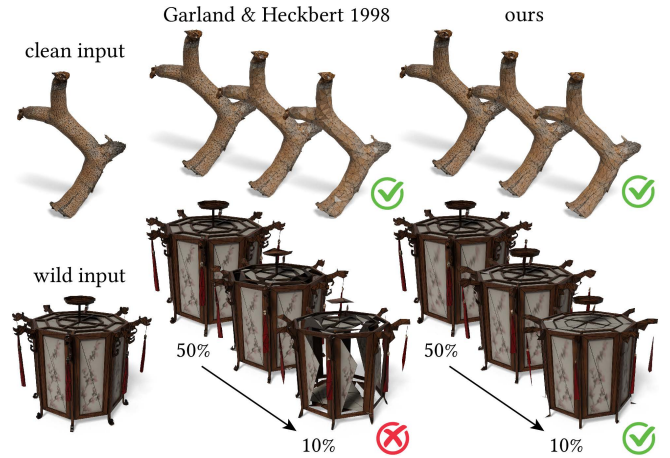


Fig. 2. Comparison between our textured mesh simplification method and a representative prior technique [Garland and Heckbert 1998] (using the implementation from [Cignoni et al. 2008]). On single component and manifold inputs, both methods produce excellent results (compare baseline, top center, vs. ours, top right). However, for challenging “wild” inputs, often characterized by non-manifold and multiple components geometry, the baseline approach frequently yields unsatisfactory results (bottom center). In contrast, our method preserves visual fidelity and geometric structure more effectively on these wild meshes (bottom right).

1 INTRODUCTION

Mesh simplification creates different levels of detail (LOD) for 3D objects while maintaining their visual fidelity. It plays a critical role in interactive graphics to achieve target performance in rendering and simulation across various hardware platforms. Its importance has been motivating decades of development with a plethora of solutions, such as the widely used QEM by Garland and Heckbert [1997]. While these solutions produce high-quality results for manifold surface inputs, we demonstrate that they often fail to generate acceptable results on meshes obtained from online repositories (see

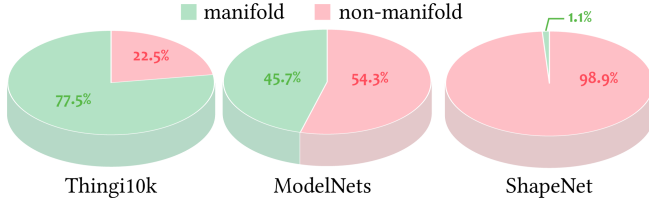


Fig. 3. Many meshes available in online repositories are non-manifold. For instance, Thingi10k [Zhou and Jacobson 2016] (a dataset consisting of 3D-printable shapes) includes 22.5% non-manifold shapes, ModelNet [Wu et al. 2015] (a dataset of CAD models) contains 54.3% non-manifold objects, and more severely 98.9% of the meshes in the ShapeNet dataset [Chang et al. 2015] (a widely used dataset for machine learning) are non-manifold.

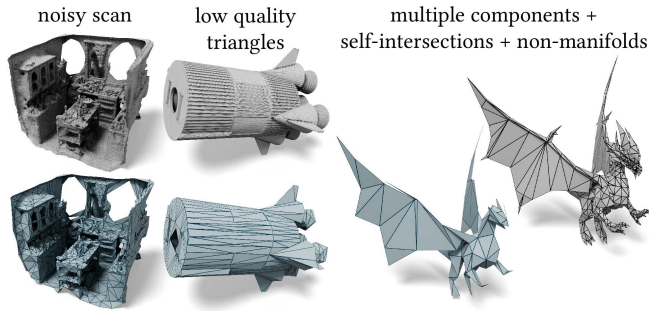


Fig. 4. We stress test our method by simplifying a variety of challenging cases encountered in the wild, including noisy 3D scans (left), low-quality triangles from CAD models (middle), and meshes with defects (right).

Fig. 1). This issue is caused by the discrepancy between existing problem formulations and real-world data characteristics. Existing LOD techniques often assume the input mesh is manifold [Dey et al. 1999]. This was a realistic assumption when, for instance during the 1990s, the goal was to simplify meshes generated by isosurfacing methods, such as marching cubes [Lorensen and Cline 1987]. However, recent advancements in 3D modeling have populated online repositories with a rich amount of manually modeled shapes. In contrast to meshes reconstructed from isosurfacing, the majority of them are non-manifold (e.g., 98.9% on ShapeNet [Chang et al. 2015] are non-manifold, see Fig. 3). This prevalence of non-manifold meshes underscores the need to revisit mesh simplification and its problem formulation to better suit contemporary datasets.

A desired mesh simplification method must be *performant* and *robust* to all types of geometric artifacts, and can *preserve surface attributes* (e.g., textures). In real-time applications such as virtual reality and video games, these (defective) meshes appear frequently during game time from user interactions (i.e., engineers collaboratively model 3D buildings). Being able to efficiently and robustly simplify all the newly created 3D content on the fly while preserving attributes is critical to achieving target performance. Such requirements also exclude expensive alternatives that do not preserve attributes (see Fig. 6), such as repairing defective meshes into manifolds and performing simplification afterward, which may take minutes to hours.

In this work, we propose a method to simplify *any* textured triangle meshes that may contain multiple components, non-manifold geometry, and even the extreme case like triangle soups (Fig. 4). We elevate the manifold restriction of prior arts to work with a soup of triangles, while guaranteeing to converge to the classic quadric error simplification results on manifold inputs. To handle textured meshes, we successively compute correspondences between the input mesh and the simplified mesh, unlike prior arts that aim at preserving UV coordinates. Our strategy reduces texture distortion and prevents texture bleeding occurs during the simplification process. This combination enables us to handle arbitrary textured meshes one can encounter today, without degraded performance on clean inputs (see Fig. 2). We provide a background (Appendix A) and analyze the limitations of existing strategies (Sec. 3). This analysis highlights issues with current tools and motivates revisiting the simplification problem for mesh data in-the-wild.

2 RELATED WORK

Our method belongs to the family of *local* decimation schemes defined on triangle meshes, aiming to preserve their surface attributes. We thus focus our discussion on prior techniques closest to ours, and refer readers to Luebke [2003] for a more comprehensive discussion.

Triangle Mesh Simplification. Mesh simplification has been extensively studied in computer graphics to reduce the resolution while preserving the appearance of 3D objects. Early attempts tried to fit low resolution polygons [Jr. and Zyda 1991], *globally* remesh the input to a lower resolution [Hoppe et al. 1992; Turk 1992], or *locally* remove each mesh element [Schroeder et al. 1992]. Since then, different formulations have been proposed, such as clustering of triangles [Cohen-Steiner et al. 2004; Xu et al. 2024] or vertices [Lindstrom 2000; Low and Tan 1997; Rossignac and Borrel 1993], greedily removing triangles [Gieng et al. 1998; Hamann 1994] or vertices [Klein 1998], simplifying a manifold envelope wrapping around the input [Chen et al. 2023b; Gao et al. 2022; Mehra et al. 2009], or optimization with a differentiable renderer [Deliot et al. 2024; Hasselgren et al. 2021; Knodt et al. 2024]. Some of these techniques have also been revisited in the context of machine learning-driven simplification [Chen et al. 2023a; Potamias et al. 2022]. Among them, perhaps the most popular strategy is the *greedy edge collapse* [Hoppe 1996] with the *quadric error metric* [Garland and Heckbert 1997]. The popularity of edge collapses is partly due to the efficiency as each edge collapse exhibits constant time complexity.

Quadric Error Metrics. The success of quadric error [Garland and Heckbert 1997] has motivated a variety of *quadrics* for measuring different geometric quantities. Lindstrom and Turk [1998] propose *volume* and *area* quadrics to measure the squared volume and area changes, respectively. Garland and Heckbert [1998] introduce the *boundary quadric* to maintain the shape boundary. Trettner and Kobbelt [2020] consider that a mesh element is a sample from a probability distribution and derive the *probabilistic quadric* to further improve its numerical behavior. Other extensions generalize the quadric error to incorporate surface attributes (e.g., textures) by defining quadric error in a higher dimension [Garland and Heckbert

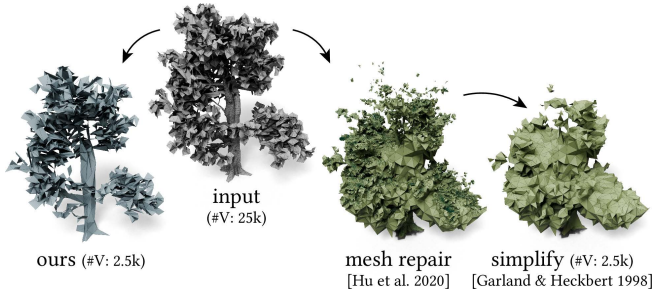


Fig. 5. Repairing a triangle mesh with [Hu et al. 2020] (third) and then simplifying the repaired model (fourth) depends on having a high quality repaired model, which is not always the case. Our method simplifies the input mesh directly and leads to a better result (first).

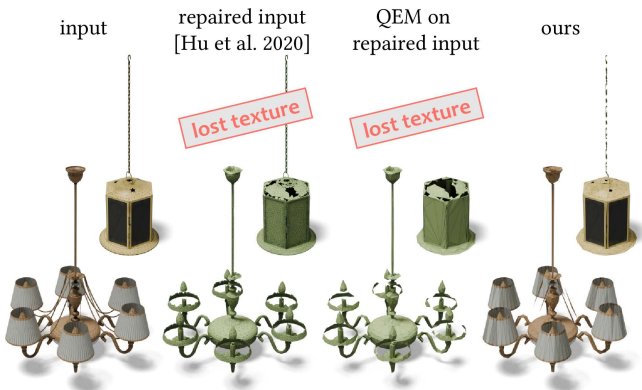


Fig. 6. Running mesh repairing [Hu et al. 2020] (second column) and then simplifying the mesh with [Garland and Heckbert 1997] (third column) leads to losses on surface attributes, such as textures, and may suffer from suboptimal results due to imperfect repairing. In contrast, our method directly simplifies the input, leading to better results (fourth column).

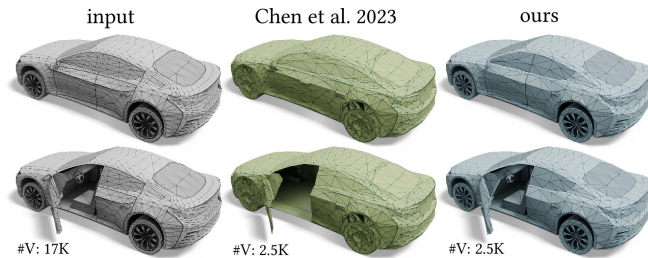


Fig. 7. Comparing to methods that construct a manifold mesh wrapper and simplify it [Chen et al. 2023b] (middle), our method directly simplifies the input and preserves interior structures (right).

1998; Garland and Zhou 2005] or by linear approximation of the attribute [Hoppe 1999]. The optimal choice of metric varies depending on the input geometry and applications. The suite of quadric error metrics provides us with several options to mix-and-match them for different tasks, including the ones beyond mesh simplification such as quadric-inspired surface reconstruction [Ju et al. 2002; Kobbelt

et al. 2001; Schaefer and Warren 2005] and filtering [Legrand et al. 2019; Vieira et al. 2010].

Topology Varying Mesh Simplification. Many of the methods mentioned above are designed for simplifying a single *manifold* triangle mesh. Another class of mesh simplification generalizes the local decimation method to triangle meshes in the wild, which may contain multiple components or non-manifolds. Garland and Heckbert [1997] reformulate the edge collapse operation as an operation to collapse any *vertex pair*, meaning one can merge two vertices even if they are not connected by an actual edge. These vertex pairs are computed based on vertex-to-vertex distance (see Fig. 15) and optimized with the quadric error metric. Schroeder [1997] identifies different cases for manifold/non-manifold vertices and removes each vertex based on a distance-to-plane metric. Popovic and Hoppe [1997] strive for maximizing the compression rate and suggest not optimizing the vertex location. In order to handle any triangle meshes, Popovic and Hoppe [1997] reformulate it as a problem of simplifying *simplicial complex*. Our method is another instance of topology varying simplification for simplicial complex with high-quality geometric approximation and is capable of transferring textures.

Another possible direction is to *repair* the mesh, i.e. convert it into a single manifold, and then simplify it. However, this depends on a repairing method to produce high-quality outputs, which is not always the case (see Fig. 5) and can lose important surface attributes (see Fig. 6). Alternatively, one can robustly create a manifold “cage” to wrap the triangle mesh and then simplify the manifold wrapper [Chen et al. 2023b; Nooruddin and Turk 2003; Portaneri et al. 2022]. However, these methods, e.g. Chen et al. [2023b], often suffer from deleting interior components which may be important for interacting with the 3D object (see Fig. 7). We thus follow the spirit of early attempts to simplify triangle meshes with vertex pair collapses.

Beyond Appearance Preservation. Mesh simplification has also been studied in the context of preserving properties related to computation and simulation, such as preserving acoustic transfer [Li et al. 2015], spectral properties [Chen et al. 2020; Keros and Subr 2023; Lescoat et al. 2020; Liu et al. 2019], intrinsic geometric quantities [Liu et al. 2023; Shoemaker et al. 2023], and the dynamic behavior of physics-based simulations [Chen et al. 2017, 2015, 2018; Kharevych et al. 2009]. When trained jointly with a machine learning model, one could even tailor-made a simplification for the underlying task, such as mesh classification [Hanocka et al. 2019; Ludwig et al. 2023]. These methods serve as fundamental building blocks to obtain a coarse approximation for a simulation. But if one aims to build a multilevel solver that requires to transfer signals across resolutions [Guskov et al. 1999], an important ingredient is to compute correspondences across the decimation hierarchy. Such a motivation has stimulated several works on computing correspondences across hierarchy by computing an *intrinsic* mapping for each local decimation [Khodakovsky et al. 2003; Lee et al. 1998; Liu et al. 2023, 2020, 2021] or by maintaining a bijective map with *extrinsic* projection [Jiang et al. 2020, 2021]. With such correspondences, one can then build hierarchical solvers for scalable computation [Liu et al. 2021; Wiersma et al. 2023; Zhang et al. 2022, 2023].

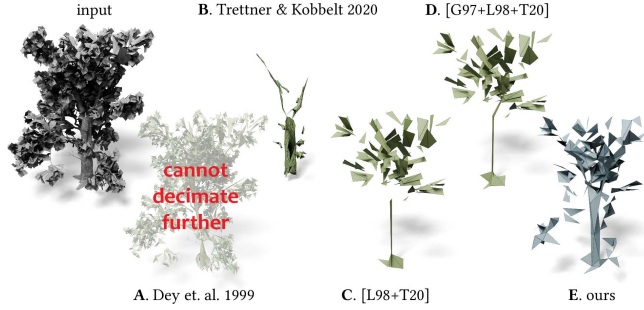


Fig. 8. Given a tree mesh with non-manifold elements and multi-connected components, we demonstrate that several off-the-shelf simplification methods and a combination of them suffer from different issues, such as failing at achieving target face counts or deleting planar components. In contrast, our method leads to perceptually better results when decimating the mesh down to 1% of the original resolution.

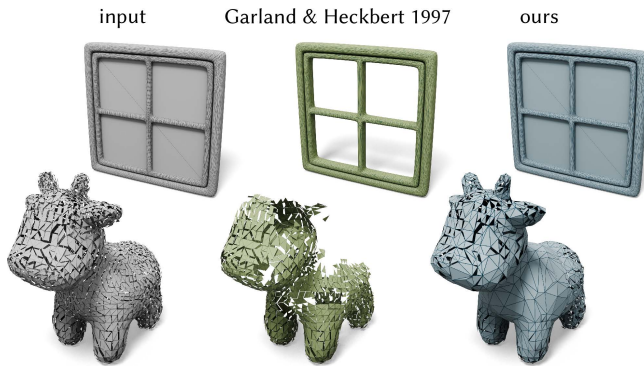


Fig. 9. Simplifying the input mesh with the edge quadric error may lead to deleting geometrically significant parts because they lead to nearly zero quadric error (green). Our method overcomes such issues by penalizing collapses that will introduce significant area change (blue).

3 PITFALLS OF QUADRIC ERROR SIMPLIFICATION

Many existing simplification methods are designed for manifold meshes. Despite being a realistic assumption for meshes extracted from marching cubes, it becomes obsolete when considering artist-created assets in today’s online repositories. The discrepancy often leads to geometric and texture issues.

3.1 Geometric Issues

Many simplification methods, such as [Dey et al. 1999], assume the input is a manifold mesh and aim at maintaining the topology throughout the simplification. However, this implicitly sets a lower bound on the coarsest mesh resolution because representing a component requires at least a single triangle (see Fig. 8 A). Not to mention that many input meshes in practice already possess topological artifacts, which immediately break implementations that assume manifold inputs.

The edge quadric error metric [Garland and Heckbert 1997; Trettner and Kobbelt 2020] suffers from area losses. As described in

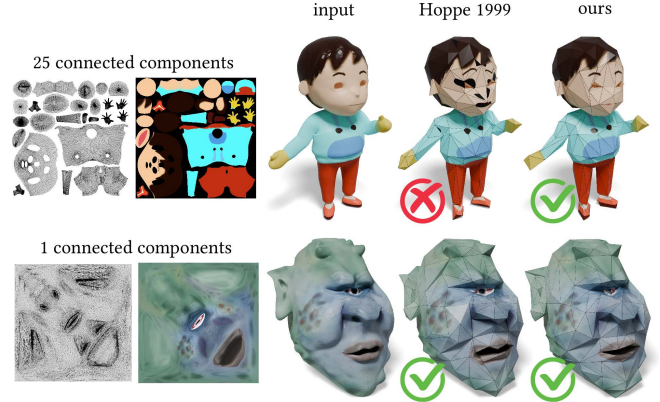


Fig. 10. Existing textured mesh simplification techniques work well when the input mesh has one or few connected components in the UV-space, a.k.a. UV islands (bottom middle). However, when deploying to meshes that have multiple UV islands, previous methods often lead to the “texture bleeding” artifact: seeing the background color of the texture image on the surface mesh (top middle). As most real-world meshes (e.g., [Maggiordomo et al. 2020]) contain multiple UV islands, this motivates our method to handle both cases more robustly (right column).

Appendix A, quadric error measures the squared distance to triangle planes. When a component is nearly *developable*, can be flattened to a plane without distortion, quadric error will report zero error when collapsing edges in the developable region, resulting in the risk of removing large components and causing significant visual difference (see Fig. 9).

A potential remedy is to incorporate the *area quadric* [Lindstrom and Turk 1998], in addition to the edge quadric error. Although it helps to preserve some large-area components, this combination (we use [L98+T20] to abbreviate the combination of area quadric [Lindstrom and Turk 1998] and the probabilistic quadric [Trettner and Kobbelt 2020]) is still prone to deleting components due to no edges to collapse between disconnected parts.

In lieu of this, Garland and Heckbert [1997] suggest constructing virtual edges for vertex pairs such that their vertex-to-vertex distance is below a small threshold. However, in Fig. 8 D, we show that this strategy, a combination of [Garland and Heckbert 1997; Lindstrom and Turk 1998; Trettner and Kobbelt 2020] (called [G97+L98+T20] for short), still does not introduce significant improvements. This is because computing distances between vertices is merely a sparse and inaccurate measure of distance between mesh components, leading to failures in constructing virtual edges between geometrically connected, but topologically disconnected components (see the inset).

3.2 Texture Issues

UV coordinates are a popular way to store the mapping to a texture space that stores surface attributes. A traditional approach to preserve mesh attributes during simplification is to minimize geometric distortion of their UVs [Garland and Heckbert 1998; Hoppe 1999].

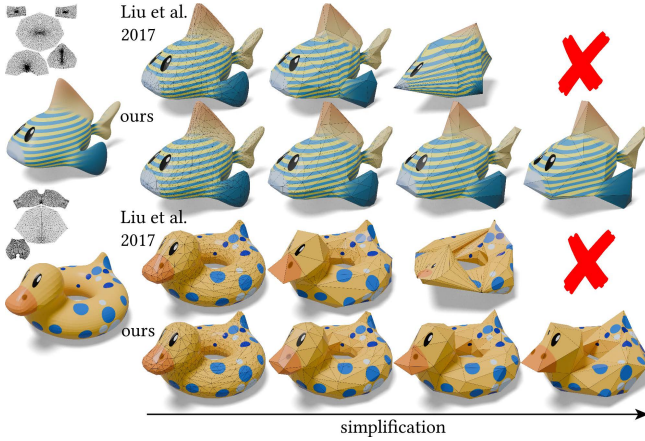


Fig. 11. An alternative method to avoid texture bleeding is to preserve the UV boundaries exactly, such as [Liu et al. 2017]. But they often lead to larger geometric distortion and have a lower bound on the resolution of the simplified mesh (top rows). Our method, in contrast, can decimate the mesh aggressively while still being able to transfer texture colors.

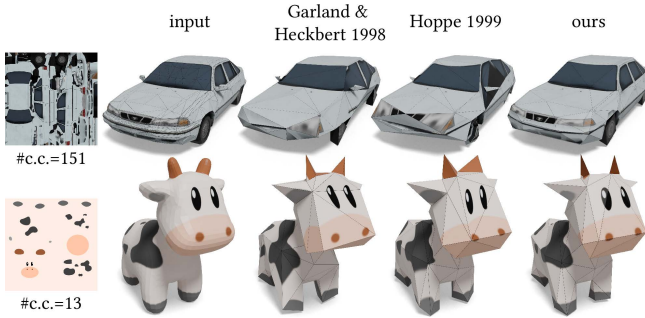


Fig. 12. Previous methods simplify a mesh while preserving its UV-coordinates for transferring texture to the simplified mesh. In addition to texture bleeding artifacts, these approaches can lead to distorted textures. Our method reduces such distortion by computing correspondences between the input and its simplified counterpart (right).

However, this strategy of *approximately* preserving UV coordinates does not perform well when the input mesh has many connected UV components, a.k.a. UV islands (see Fig. 21). This is because if the boundary of each UV island is not exactly preserved, and when the renderer interpolates texture colors near UV boundaries, distortion in the boundary curves could lead to interpolating the background color of the UV image onto the surface mesh. This behavior of rendering the background color on the mesh surface is also known as the “texture bleeding” artifact (see Fig. 10). In practice, a majority of the meshes have more than one UV islands, leading to significant texture bleeding (see Fig. 10) or texture distortions (see Fig. 12) when applying this strategy to meshes in the wild. An alternative is to maintain the boundary shape of each UV island *exactly* [Liu et al. 2017]. However, this can lead to significant geometric distortion and possibly result in the early termination of the simplification



Fig. 13. Compared to an expensive mesh optimization method with differentiable rendering that usually takes a few hours [Hasselgren et al. 2021], our method is orders of magnitude faster and achieves comparable quality.

process (see Fig. 11). On the other hand, although methods like differentiable rendering [Hasselgren et al. 2021] can potentially repair texture bleeding, such a texture optimization often leads to minutes to hours of additional computation time (see Fig. 13), making them unsuitable for real-time applications.

4 METHOD

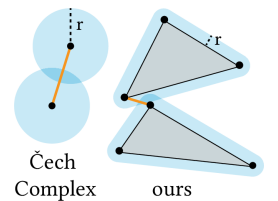
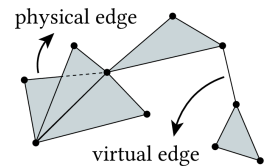
In Sec. 3, we highlight the pitfalls of the off-the-shelf methods (e.g., [Liu et al. 2017; Trettner and Kobbelt 2020]) failing at achieving satisfying results on wild meshes. We further show that direct combinations of existing techniques, including [L98+T20] and [G97+L98+T20] suffer from various issues. In this section, we present our method to address these issues in order to achieve high quality simplification on textured meshes in the wild.

4.1 Vertex Pair Collapses

Given a triangle mesh with vertices V and faces F , we first construct a set of *physical* and *virtual* edges E to connect vertex pairs. We use *physical* edges to denote the edges that are connected to at least one of the faces in F , and *virtual* edges to denote the edges without neighboring faces (see inset). The combination $M = (V, E, F)$, a.k.a. *simplicial 2-complex*, is the input to our system. We decimate the model by iteratively collapsing edges (1-simplices) in E prioritized by our error metric in Sec. 4.2.

4.1.1 Virtual Edges. Defining virtual edges is critical for decimating a wild mesh with multiple disconnected mesh components (see Fig. 14). A common heuristic is to connect vertices with small vertex-to-vertex distance. However, it often leads to suboptimal results because topologically disconnected components that are geometrically close may still have large vertex-to-vertex distance due to discretization, such as Fig. 15.

We thus take inspiration from the construction of the Čech complex to form virtual edges. In a nutshell, Čech complex forms the connectivity between points if their r -radius balls have non-empty



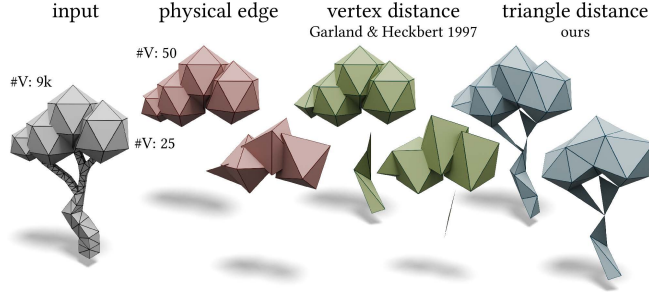


Fig. 14. We simplify an input mesh (gray) with our method that computes virtual edges based on the triangle distances (blue). Our method can merge initially disconnected parts and lead to a more compact simplified mesh compared to the method by Garland and Heckbert [1997] which constructs virtual edges with vertex distances (green) and the method that is solely based on physical edges (red).

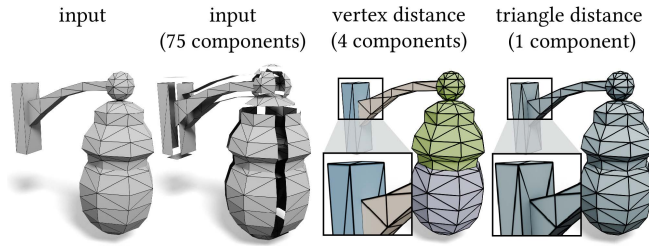


Fig. 15. Given a lamp mesh representing a single connected rigid object (first), the raw data, though, may contain several disconnected pieces from the modeling process (second). Using vertex distance (third) to infer the connectivity still leads to 4 disconnected components indicated by different colors [Garland and Heckbert 1997]. In contrast, using our triangle distance (fourth) leads to a single component. We visualize both physical and virtual edges with thick black lines in the third and fourth images.

intersection. In our case of triangle meshes, the natural generalization of the r -radius ball around a point becomes the r -offset surface around a triangle (see inset). We thus check connectivity between them by checking whether two offset surfaces have intersection. Intuitively, the construction suggests a straightforward implementation by computing triangle-to-triangle distances. If the distance between two triangles is smaller than $2r$ and the two triangles are not from the same connected component, we form a virtual edge between the two triangles by connecting their closest vertex pairs (see the inset). Because our method additionally checks connected components, our result is *not* a Čech complex, but it shares a similar spirit in the construction.

Once we have established an edge set E with virtual edges and physical edges derived from F , we use a simplicial complex data structure to represent our input (V, E, F) (detailed in Appendix C), then iteratively collapse edges (1-simplices) to decimate the mesh. In Fig. 14, we demonstrate that our triangle distance measure is more effective than the naive vertex distance for simplifying meshes with multiple components.

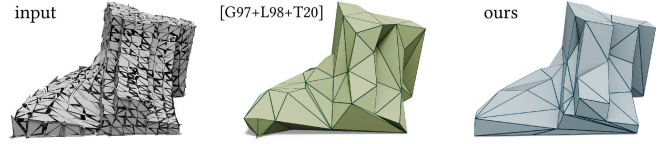


Fig. 16. Our novel quadric accumulation, the *memory* edge quadric with the *memoryless* area quadric, leads to a significantly better result in preserving sharp features compared to the straightforward combination of quadric [Garland and Heckbert 1997] and area quadric [Lindstrom and Turk 1998]. To illustrate the difference, we compare our method against such a combination, augmented with a tiny bit of probabilistic quadric [Trettner and Kobbelt 2020]) for numerical robustness and extended it with the data structure of [Popovic and Hoppe 1997] to handle non-manifolds. Given a soup of triangles (left), our method (right) leads to a better simplification result than the baseline (middle).

4.2 Error Metric

A key element in edge collapse algorithms is to define an error metric to prioritize the sequence of edge collapses. As pointed out in Sec. 3, using the popular *edge quadric error* [Garland and Heckbert 1997] or its probabilistic version [Trettner and Kobbelt 2020] suffers from deleting large planar components. Naively adding the *boundary quadric* [Garland and Heckbert 1998] or the *area quadric* [Lindstrom and Turk 1998] may lead to oversmoothing (see Fig. 16). This motivates our modified metric to handle non-manifold meshes with multiple components.

Our key observation comes from studying the oversmoothing behavior of the area quadric. When contracting a virtual edge, it can glue boundary edges (red edges in the inset) into interior edges (green edges in the inset). In such cases, the usual implementation of summing up quadrics (suggested in [Garland and Heckbert 1997]) leads to area quadrics being applied to interior edges later in the decimation. This behavior results in incorrectly high costs for removing interior edges. Collapsing an interior edge often leads to some area gain on one side of the edge and loss on the other. Although they should cancel each other and result in a small area change, area quadrics do not differentiate between the two (see Sec. A.3) and would simply square the area changes on both sides and sum up the results. This results in a high cost when decimating long interior edges and leads to overly uniform decimation.

To overcome the issue of oversmoothing, we propose a small, yet effective change in the quadric error metric accumulation. Specifically, we accumulate the edge quadric with the usual summation as proposed in [Garland and Heckbert 1997], a.k.a. the *memory* implementation, but we do not accumulate the area quadric term, a.k.a. the *memoryless* implementation. This targeted adjustment significantly improves sharp feature preservation when decimating a soup of triangles (see Fig. 16). Crucially, our method “converges” to the standard quadric error metric [Garland and Heckbert 1997] for closed manifold meshes. This ensures that our method maintains the excellent performance of [Garland and Heckbert 1997] on manifold inputs, while improving the results for meshes in the wild.

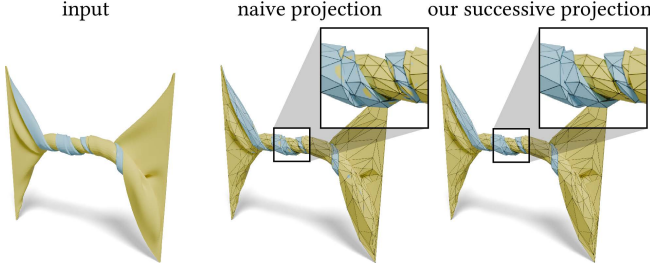


Fig. 17. On a thin shell cloth mesh with different colors (blue, yellow) on different sides, naive closest point projection between the simplified and the input meshes may lead to projecting points onto the wrong side (middle). In contrast, our successive closest point projection encourages the projection to lie on the corresponding side, leading to less projection error.

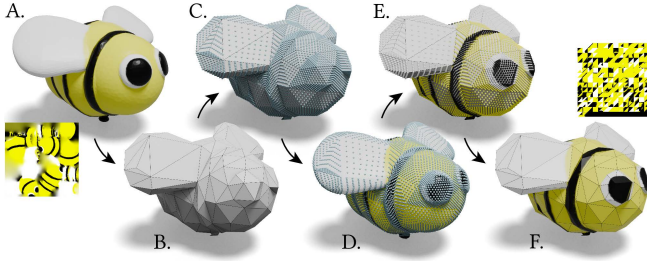


Fig. 18. Given an input mesh (A), we simplify it (B) while computing a successive map from the simplified mesh to the input. With this map, we can sample points on the simplified mesh (C), obtain correspondences to the texel locations, and transfer those points to the input with the correspondence map (D). Then we can simply look up the color information on the input (E) and generate a new texture image after simplification (F).

4.3 Texture Transfer With Successive Mapping

Previous textured mesh simplification methods preserve textures by preserving UV coordinates of the input mesh. This class of approach however suffers from issues detailed in Sec. 3.2. We take inspiration from the idea of *imposters* [Christiansen 2005] which bake a new texture to a simplified geometry via mapping. Our approach simplifies a mesh while keeping track of correspondences between the input mesh and its simplified counterpart with a *successive mapping*, and then bake a new UV map after simplification. Our method avoids *texture bleeding* (see Fig. 10) and supports more aggressive simplifications (see Fig. 11).

4.3.1 Successive Projection. Given an input mesh $M^0 = (V, E, F)$ and its coarsened counterpart $M^c = (V_c, E_c, F_c)$, our goal is to compute a function $T : M^c \rightarrow M^0$ that maps a point $p \in M^c$ on the simplified mesh to its corresponding point $T(p) \in M^0$ on the input. Then, for any point p on M^c , we can retrieve attribute values from $T(p)$ and bake new texture maps for M^c .

Inspired by [Liu et al. 2020], we compute the mapping T successively. Starting with an identity map from M^0 to itself, we successively update the map during each edge collapse. However, previous

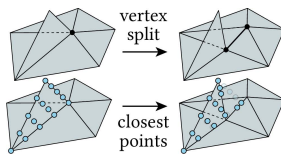


Fig. 19. Our method is applicable to any injective texture mapping method, such as mesh color textures [Yuksel 2017] and BLENDER’s smart UV.

methods [Liu et al. 2020] require the input to be a manifold mesh and without topological changes during simplification. This violates the scenario we considered, preventing us from using them.

We thus compute the mapping T by successive closest point projection within the edge one-ring neighborhoods. Specifically, we store the decimation history $\{M^0, M^1, \dots, M^c\}$ where two consecutive levels only differ by an edge collapse/vertex split. If a given point $p \in M^c$ lies within the vertex one-ring of M^c , we perform a vertex split to obtain M^{c-1} , and project to the edge one-ring of M^{c-1} (see inset). We continue the local projection within the vertex one-ring until we reach M^0 . This localized projection encourages (but does not guarantee) projecting each point to the part of the surface it comes from, making it more robust to thin shell structures (see Fig. 17). One important implementation detail is that we reuse point locations on M^c for closest point queries, but using successive maps $M^c \rightarrow M^{c-1} \rightarrow \dots \rightarrow M^0$ to identify relevant edge one-rings is crucial to our results. This ensures that the sampled surface attributes are geometrically closer to the simplified mesh M^c and avoids accumulating distortions from compositing multiple successive projections.

With the computed mapping T , we can bake any attributes from the input M^0 to the simplified mesh M^c as textures via the process described in Fig. 18. Our mapping approach is compatible with any injective texture mapping technique (see Fig. 19). In our paper, we use the *mesh color texture* [Yuksel 2017] due to its simplicity.

5 RESULTS

Our method embrace the flexibility of existing QEM variants to control the decimation process, such as weighted simplification driven by visibility [Hoppe 1997] (see Fig. 20). We further enjoy the efficiency of being an edge collapse method that we can decimate hundreds of thousands of edges in a few seconds. In the inset, we report the total runtime of decimating the meshes from the Polyhaven dataset down to 100 faces with around 56,000 texture sample points, evaluated on a MacBook with the M1 processor.

Our method shares the same asymptotic complexity, $O(n \log n)$, as previous edge collapse methods implemented with priority queues. However, our method has an extra precomputation of triangle-to-triangle distances for virtual edges, unlike previous methods which

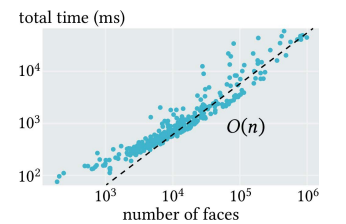


Table 1. We decimate the meshes of the Thingi10K dataset [Zhou and Jacobson 2016] down to 0.1%/1%/10% of their original resolution and report the geometric errors (Hausdorff distance, mean squared Chamfer distance) averaged across the dataset. The meshes are normalized to have unit diagonal bounding boxes to avoid bias towards large meshes.

Methods	Hausdorff $\times 10^{-1}$	Chamfer $\times 10^{-1}$
[Garland and Heckbert 1998]	1.05/0.57/0.13	2.36/1.37/0.46
[Trettner and Kobbelt 2020]	1.05/0.58/0.14	2.36/1.36/0.39
Ours	0.84/0.44/0.10	2.06/1.11/0.29

Table 2. We compute the symmetric Chamfer distance on the textures [Yuan et al. 2018] for our method and the method by Garland and Heckbert [1998]. We decimate the mesh down to 1% of the original resolution, and then report the average error from the *Real-World Textured Things* [Maggiordomo et al. 2020] and the Polyhaven datasets, respectively separated by “/”, showing that our method quantitatively achieves lower texture errors.

Methods	Textured Chamfer $\times 10^{-1}$
[Garland and Heckbert 1998]	0.18 / 0.11
Ours	0.10 / 0.09

simply decimates existing physical edges. Compared to previous UV-preserving mesh simplification (e.g., [Garland and Heckbert 1998]), our method is cheaper during simplification because we avoid the computation of high-dimensional quadrics for the UV coordinates. But we instead require an extra post-process to transfer textures. Specifically, our method requires to map each texel, represented as a barycentric point on the simplified mesh, to the input mesh by going through the decimation history. Luckily, this process is efficient because the barycentric coordinate for each texture sample will only be changed by a few edge collapses where the sample lies within the edge one-ring. This property allows us to skip computations for a majority of edge collapses when mapping texel points. From our implementation, computing correspondences for each point only involves less than 5 microseconds. When querying a large number of texels, this process can be trivially parallelized because the computation for each texel is independent.

Qualitative Evaluations. Our method offers several key advantages demonstrated throughout this paper. (1) Its virtual edge construction yields better results compared to previous method by Garland and Heckbert [1997] in Fig. 14 and Fig. 8. (2) Our enhanced error metric effectively prioritizes merging disconnected components while preventing the deletion of components with large surface areas (Fig. 2, 9). (3) The robust texture handling avoids texture bleeding, outperforming UV-preserving techniques (see Fig. 12), especially on non-manifold inputs with multiple components (Fig. 1). Integrating these components, our system demonstrates quality improvements across numerous textured models in Fig. 24, 21.



Fig. 20. Our method is built on top of edge collapse algorithms, and thus can be seamlessly integrated with existing QEM extensions, such as simplifying a mesh driven by visibility [Hoppe 1997]. In this example, we simplify the rubber duck statue based on the visible region (bottom left) from the observer on the island.



Fig. 21. We present a subset of textured mesh simplification results from [Garland and Heckbert 1998] and ours on the *Real-World Textured Things* dataset [Maggiordomo et al. 2020]. A majority of them (the first column) have multiple texture islands (second column). Using the existing method (third column) suffers from texture bleeding and large distortion. In contrast, we obtain better results (fourth column) and avoid texture bleeding.

Quantitative Geometric Evaluations. In Tab. 1, we quantitatively evaluate the geometric quality (excluding textures) of our simplification results against [Garland and Heckbert 1998; Trettner and Kobbelt 2020], demonstrating that our method leads to smaller *Hausdorff* and *mean squared Chamfer* distances on the *Thing10K* dataset [Zhou and Jacobson 2016]. In Fig. 23 we show that our method can reliably decimate every mesh in the dataset down to 0.1% of the original resolution for the entire dataset.

Quantitative Texture Evaluations. We quantitatively evaluate textured mesh simplification on the *Real-World Textured Things* [Maggiordomo et al. 2020] and the Polyhaven datasets. Specifically, we decimate the meshes in the datasets that contain a single texture image (though potentially with multiple texture islands) down to 1% of the input resolution. To avoid bias towards any specific render view, we measure the error using the average symmetric Chamfer distance on textures [Yuan et al. 2018], which measures the L2 norm of the color difference (color is normalized to [0,1]) between the closest spatial point pairs sampled on the surface (using ten thousands samples per mesh). In Tab. 2, our method consistently achieves lower errors than the method by Garland and Heckbert [1998]. We exclude the method by [Liu et al. 2017] because it struggled to achieve target resolutions as shown in Fig. 24.

User Studies. In addition to the qualitative and quantitative evaluations, we performed user studies to get perceptual opinions on our mesh simplification results. On the geometry side (see inset, top), we evaluated meshes presented in the Fig. 25, showing that our outputs were preferred, in comparison to [Trettner and Kobbelt 2020] and the combination of baselines discussed in Sec. 3. On the texture side, we conducted user study on randomly sampled meshes from the PolyHaven dataset, which contains a mixture of non-manifold textured meshes and manifold meshes extracted from isosurfacing. In the inset (bottom), we show that more than 80% of the participants agree that our method is either comparable or outperforming [Garland and Heckbert 1998]. Note that in this experiment, we further included an option to select *Similar* quality in the study to verify the claim that our performance is comparable to [Garland and Heckbert 1998] for manifold meshes. We provide more details about the studies in Appendix D.

6 DISCUSSION & CONCLUSION

We have presented a practical mesh simplification method that can achieve better results on human-created meshes that commonly found online. We have also described an effective attribute transfer technique to preserve texture quality without impacting the quality of the mesh simplification process. Our method extends the reach



Fig. 22. Our texture is based on a localized closest point projection. When the decimation introduces topological changes (e.g. deleting components), our projection may pick up colors from different components, leading to noticeable texture distortion.

of mesh simplification to gracefully handle triangle meshes with inconvenient properties, including non-manifold meshes and meshes with multiple connected components. This enhances performance by reducing the complexity of 3D models without compromising visual quality, enabling fast rendering and interactive simulation for real-time applications like online gaming, especially on low-power devices such as mobile phones and VR headsets, where computational resources are limited.

However, our current attribute transfer mechanism does not guarantee sampling from the outermost parts of the surface, potentially resulting in textures incorporating data from interior layers, which may be undesirable in certain cases (see Fig. 22). Future work could address this problem by considering external visibility information while computing the mapping from the final mesh to the input surface. Employing adaptive texturing techniques based on the color content could mitigate potential blurry textures due to insufficient texel density in high-frequency areas. Exploring different stopping criteria, instead of simply setting target triangle counts, could further enhance the practical utility when deciding which geometric resolution to use for a given application. Adding area quadrics reduces the risk of removing large area components, but future exploration to avoid deleting many small-area components (e.g., coniferous trees) could further enhance the visual perception on a wider variety of meshes. Our research contributes to the field of robust geometry processing and underscores the need to further enhance the robustness of downstream geometric algorithms, particularly because the output of our method, like the input, may contain defects such as non-manifold structures.

REFERENCES

- Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. *ShapeNet: An Information-Rich 3D Model Repository*. Technical Report arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago.
- Desai Chen, David I. W. Levin, Wojciech Matusik, and Danny M. Kaufman. 2017. Dynamics-aware numerical coarsening for fabrication design. *ACM Trans. Graph.* 36, 4 (2017), 84:1–84:15.
- Desai Chen, David I. W. Levin, Shinjiro Sueda, and Wojciech Matusik. 2015. Data-driven finite elements for geometry and material design. *ACM Trans. Graph.* 34, 4 (2015), 74:1–74:10.
- Honglin Chen, Hsueh-Ti Derek Liu, Alec Jacobson, and David I. W. Levin. 2020. Chordal decomposition for spectral coarsening. *ACM Trans. Graph.* 39, 6 (2020), 265:1–265:16.
- Jiong Chen, Hujun Bao, Tianyu Wang, Mathieu Desbrun, and Jin Huang. 2018. Numerical coarsening using discontinuous shape functions. *ACM Trans. Graph.* 37, 4

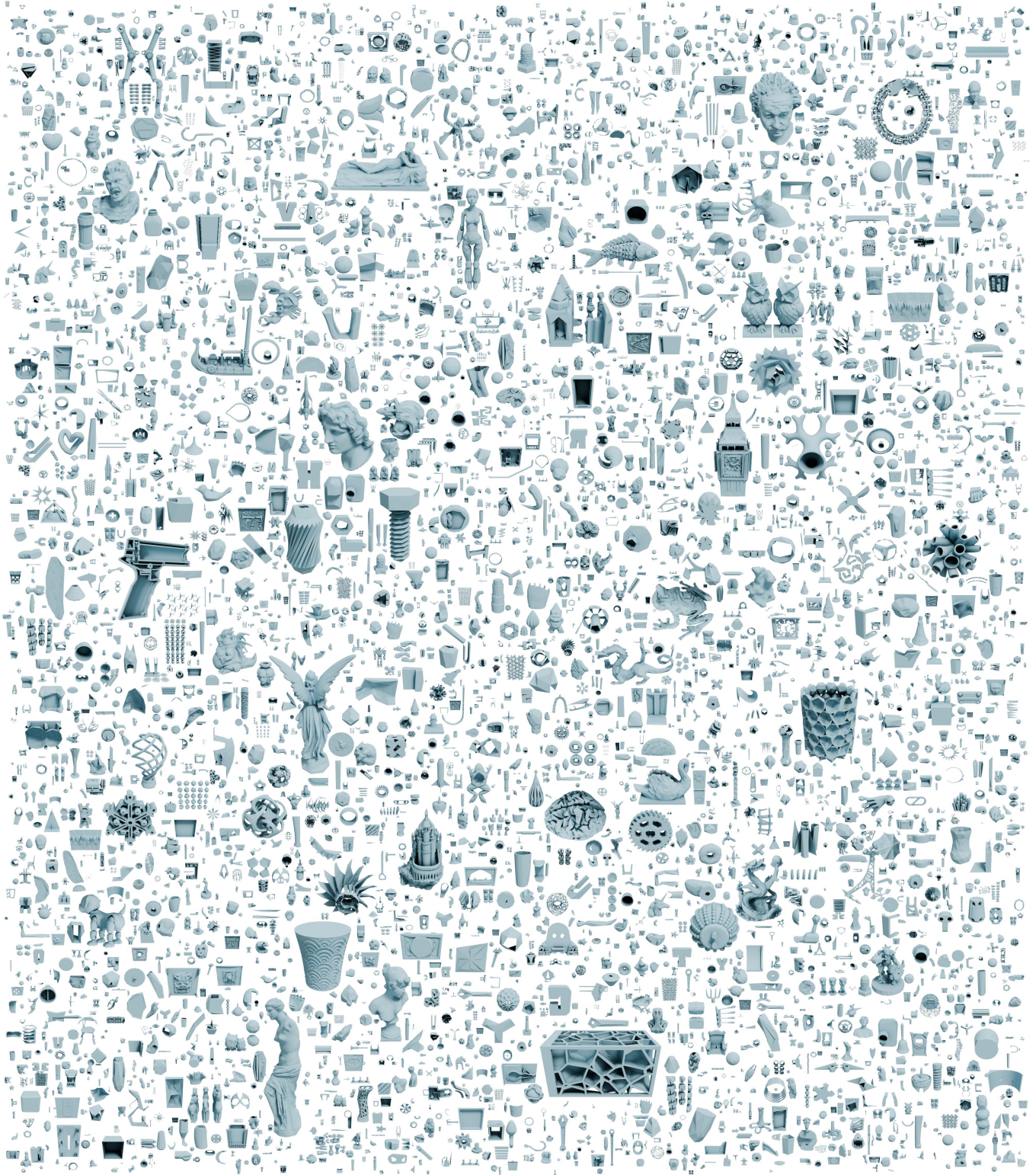


Fig. 23. Our method can robustly decimation all 10 thousand meshes from [Zhou and Jacobson 2016] down to 1% of their original resolution. Here we display a randomly selected subset of the simplified meshes.



Fig. 24. We show a subset of the textured mesh simplification results from the *PolyHaven* dataset. We can observe several benefits of our method, such as preserving large planar regions, avoiding texture bleeding, and converging to comparable quality as previous methods when the input is a clean manifold mesh. We use faded colors to denote the failure in achieving target resolutions.

- (2018), 120.
- Yun-Chun Chen, Vladimir G. Kim, Noam Aigerman, and Alec Jacobson. 2023a. Neural Progressive Meshes. In *ACM SIGGRAPH 2023 Conference Proceedings, SIGGRAPH 2023, Los Angeles, CA, USA, August 6-10, 2023*, Erik Brunvand, Alla Sheffer, and Michael Wimmer (Eds.). ACM, 84:1–84:9.
- Zhen Chen, Zherong Pan, Kui Wu, Etienne Vouga, and Xifeng Gao. 2023b. Robust Low-Poly Meshing for General 3D Models. *ACM Trans. Graph.* 42, 4 (2023), 119:1–119:20.
- Kenneth Rohde Christiansen. 2005. The use of imposters in interactive 3D graphics systems. *Department of Mathematics and Computing Science* 3 (2005), 1–8.
- Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. 2008. MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference*, Vittorio Scarano, Rosario De Chiara, and Ugo Erra (Eds.). The Eurographics Association.
- David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2004. Variational shape approximation. *ACM Trans. Graph.* 23, 3 (2004), 905–914.
- Thomas Deliot, Eric Heitz, and Laurent Belcour. 2024. Transforming a Non-Differentiable Rasterizer into a Differentiable One with Stochastic Gradient Estimation. *arXiv preprint arXiv:2404.09758* (2024).
- Tamal Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry Nekhayev. 1999. Topology preserving edge contraction. *Publications de l'Institut Mathématique* 66 (1999).
- Leila De Florian and Annie Hui. 2005. Data Structures for Simplicial Complexes: An Analysis And A Comparison. In *Third Eurographics Symposium on Geometry Processing, Vienna, Austria, July 4-6, 2005 (ACM International Conference Proceeding Series, Vol. 255)*, Mathieu Desbrun and Helmut Pottmann (Eds.). Eurographics Association, 119–128.
- Xifeng Gao, Kui Wu, and Zherong Pan. 2022. Low-poly Mesh Generation for Building Models. In *SIGGRAPH '22: Special Interest Group on Computer Graphics and Interactive Techniques Conference, Vancouver, BC, Canada, August 7 - 11, 2022*, Munkhtsetseg Nandigajav, Niloy J. Mitra, and Aaron Hertzmann (Eds.). ACM, 3:1–3:9.
- Michael Garland and Paul S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997, Los Angeles, CA, USA, August 3-8, 1997*, G. Scott Owen, Turner Whitted, and Barbara Mones-Hattal (Eds.). ACM, 209–216.
- Michael Garland and Paul S. Heckbert. 1998. 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*, David S. Ebert, Holly E. Rushmeier, and Hans Hagen (Eds.). IEEE Computer Society and ACM, 263–269.
- Michael Garland and Yuan Zhou. 2005. Quadric-based simplification in any dimension. *ACM Trans. Graph.* 24, 2 (2005), 209–239.
- Tran S. Gieng, Bernd Hamann, Kenneth I. Joy, Gregory L. Schussman, and Issac J. Trotts. 1998. Constructing Hierarchies for Triangle Meshes. *IEEE Trans. Vis. Comput. Graph.* 4, 2 (1998), 145–161.
- Igor Guskov, Wim Sweldens, and Peter Schröder. 1999. 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*, Warren N. Waggenspack (Ed.). ACM, 325–334.
- Bernd Hamann. 1994. A data reduction scheme for triangulated surfaces. *Comput. Aided Geom. Des.* 11, 2 (1994), 197–214.
- Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: a network with an edge. *ACM Trans. Graph.* 38, 4 (2019), 90:1–90:12.
- Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. 2021. Appearance-Driven Automatic 3D Model Simplification. In *32nd Eurographics Symposium on Rendering, EGSR 2021 - Digital Library Only Track, Saarbrücken, Germany, June 29 - July 2, 2021*, Adrien Bousseau and Morgan McGuire (Eds.). Eurographics Association, 85–97.
- Hughes Hoppe. 1996. 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*, John Fujii (Ed.). ACM, 99–108.
- Hughes Hoppe. 1997. View-dependent refinement of progressive meshes. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997, Los Angeles, CA, USA, August 3-8, 1997*, G. Scott Owen, Turner Whitted, and Barbara Mones-Hattal (Eds.). ACM, 189–198.
- Hughes Hoppe. 1999. 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*, David S. Ebert, Markus H. Gross, and Bernd Hamann (Eds.). IEEE Computer Society and ACM, 59–66.
- Hughes Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1992. 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*, James J. Thomas (Ed.). ACM, 71–78.
- Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast tetrahedral meshing in the wild. *ACM Trans. Graph.* 39, 4 (2020), 117.
- Zhongshi Jiang, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2020. Bijective projection in a shell. *ACM Trans. Graph.* 39, 6 (2020), 247:1–247:18.
- Zhongshi Jiang, Ziyi Zhang, Yixin Hu, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2021. Bijective and coarse high-order tetrahedral meshes. *ACM Trans. Graph.* 40, 4 (2021), 157:1–157:16.
- Michael J. DeHaemer Jr. and Michael Zyda. 1991. Simplification of objects rendered by polygonal approximations. *Comput. Graph.* 15, 2 (1991), 175–184.
- Tao Ju, Frank Losasso, Scott Schaefer, and Joe D. Warren. 2002. Dual contouring of hermite data. *ACM Trans. Graph.* 21, 3 (2002), 339–346.
- Alexandros Dimitrios Keros and Kartic Subr. 2023. Spectral Coarsening with Hodge Laplacians. In *ACM SIGGRAPH 2023 Conference Proceedings, SIGGRAPH 2023, Los Angeles, CA, USA, August 6-10, 2023*, Erik Brunvand, Alla Sheffer, and Michael Wimmer (Eds.). ACM, 22:1–22:11.
- Liliya Kharevych, Patrick Mullen, Houman Owhadi, and Mathieu Desbrun. 2009. Numerical coarsening of inhomogeneous elastic materials. *ACM Trans. Graph.* 28, 3 (2009), 51.
- Andrei Khodakovskiy, Nathan Litke, and Peter Schröder. 2003. Globally smooth parameterizations with low distortion. *ACM Trans. Graph.* 22, 3 (2003), 350–357.
- Reinhard Klein. 1998. Multiresolution representations for surfaces meshes based on the vertex decimation method. *Comput. Graph.* 22, 1 (1998), 13–26.
- Julian Knodt, Zherong Pan, Kui Wu, and Xifeng Gao. 2024. Joint UV Optimization and Texture Baking. *ACM Trans. Graph.* 43, 1 (2024), 2:1–2:20.
- Leif Kobbelt, Mario Botsch, Ulrich Schwaner, and Hans-Peter Seidel. 2001. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001, Los Angeles, California, USA, August 12-17, 2001*, Lynn Pocock (Ed.). ACM, 57–66.
- Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence C. Cowsar, and David P. Dobkin. 1998. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, Orlando, FL, USA, July 19-24, 1998*, Steve Cunningham, Walt Bransford, and Michael F. Cohen (Eds.). ACM, 95–104.
- Hélène Legrand, Jean-Marc Thiery, and Tamy Boubekeur. 2019. Filtered Quadrics for High-Speed Geometry Smoothing and Clustering. *Comput. Graph. Forum* 38, 1 (2019), 663–677.
- Thibault Lescoat, Hsueh-Ti Derek Liu, Jean-Marc Thiery, Alec Jacobson, Tamy Boubekeur, and Maks Ovsjanikov. 2020. Spectral Mesh Simplification. *Comput. Graph. Forum* 39, 2 (2020), 315–324.
- Dingzeyu Li, Yun (Raymond) Fei, and Changxi Zheng. 2015. Interactive Acoustic Transfer Approximation for Modal Sound. *ACM Trans. Graph.* 35, 1 (2015), 2:1–2:16.
- Peter Lindstrom. 2000. 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*, Judith R. Brown and Kurt Akeley (Eds.). ACM, 259–262.
- Peter Lindstrom and Greg Turk. 1998. 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*, David S. Ebert, Holly E. Rushmeier, and Hans Hagen (Eds.). IEEE Computer Society and ACM, 279–286.
- Hsueh-Ti Derek Liu, Mark Gillespie, Benjamin Chislett, Nicholas Sharp, Alec Jacobson, and Keenan Crane. 2023. Surface Simplification using Intrinsic Error Metrics. *ACM Trans. Graph.* 42, 4 (2023), 118:1–118:17.
- Hsueh-Ti Derek Liu, Alec Jacobson, and Maks Ovsjanikov. 2019. Spectral coarsening of geometric operators. *ACM Trans. Graph.* 38, 4 (2019), 105:1–105:13.
- Hsueh-Ti Derek Liu, Vladimir G. Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. 2020. Neural subdivision. *ACM Trans. Graph.* 39, 4 (2020), 124.
- Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. 2021. Surface multigrid via intrinsic prolongation. *ACM Trans. Graph.* 40, 4 (2021), 80:1–80:13.
- Songrun Liu, Zachary Ferguson, Alec Jacobson, and Yotam I. Gingold. 2017. Seamless: seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Trans. Graph.* 36, 6 (2017), 216:1–216:15.
- William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987, Anaheim, California, USA, July 27-31, 1987*, Maureen C. Stone (Ed.). ACM, 163–169.
- Kok-Lim Low and Tiow Seng Tan. 1997. Model Simplification Using Vertex-Clustering. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics, S3D '97, Providence, RI, USA, April 27-30, 1997*, Andy van Dam (Ed.). ACM, 75–82, 188.
- Ingmar Ludwig, Daniel Tyson, and Marcel Campen. 2023. HalfedgeCNN for Native and Flexible Deep Learning on Triangle Meshes. *Comput. Graph. Forum* 42, 5 (2023), i–viii.
- David Luebke. 2003. *Level of detail for 3D graphics*. Morgan Kaufmann.
- Andrea Maggioridomo, Federico Ponchio, Paolo Cignoni, and Marco Tarini. 2020. *Real-World Textured Things: A repository of textured models generated with modern photo-reconstruction tools*. *Comput. Aided Geom. Des.* 83 (2020), 101943.
- Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Ashurst Gooch, and Niloy J. Mitra. 2009. Abstraction of man-made shapes. *ACM Trans. Graph.* 28, 5 (2009), 137.

- Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. 2002. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. In *Third International Workshop "Visualization and Mathematics", VisMath 2002, Berlin, Germany, May 22-25, 2002 (Mathematics and Visualization)*, Hans-Christian Hege and Konrad Polthier (Eds.). Springer, 35–57.
- David E. Muller and Franco P. Preparata. 1978. Finding the Intersection of two Convex Polyhedra. *Theor. Comput. Sci.* 7 (1978), 217–236.
- Fakir S. Nooruddin and Greg Turk. 2003. Simplification and Repair of Polygonal Models Using Volumetric Techniques. *IEEE Trans. Vis. Comput. Graph.* 9, 2 (2003), 191–205.
- Jovan Popovic and Hugues Hoppe. 1997. Progressive simplicial complexes. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997, Los Angeles, CA, USA, August 3-8, 1997*, G. Scott Owen, Turner Whitted, and Barbara Mones-Hattal (Eds.). ACM, 217–224.
- Cédric Portaneri, Mael Rouxel-Labbé, Michael Hemmer, David Cohen-Steiner, and Pierre Alliez. 2022. Alpha wrapping with an offset. *ACM Trans. Graph.* 41, 4 (2022), 127:1–127:22.
- Rolandos Alexandros Potamias, Stylianos Ploumpis, and Stefanos Zafeiriou. 2022. Neural Mesh Simplification. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 18562–18571.
- Jarek Rossignac and Paul Borrel. 1993. 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] (IFIP Series on Computer Graphics)*, Bianca Falcidieno and Tosiyaasu L. Kunii (Eds.). Springer, 455–465.
- Scott Schaefer and Joe D. Warren. 2005. Dual Marching Cubes: Primal Contouring of Dual Grids. *Comput. Graph. Forum* 24, 2 (2005), 195–201.
- William J. Schroeder. 1997. A topology modifying progressive decimation algorithm. In *8th IEEE Visualization Conference, IEEE Vis 1997, Phoenix, AZ, USA, October 19-24, 1997, Proceedings*. IEEE Computer Society and ACM, 205–212.
- William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. 1992. 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*, James J. Thomas (Ed.). ACM, 65–70.
- Randy Shoemaker, Sam Sartor, and Pieter Peers. 2023. Intrinsic Mesh Simplification. *CoRR* abs/2307.07115 (2023). <https://doi.org/10.48550/ARXIV.2307.07115> arXiv:2307.07115
- Philip Trettner and Leif Kobbelt. 2020. Fast and Robust QEF Minimization using Probabilistic Quadrics. *Comput. Graph. Forum* 39, 2 (2020), 325–334.
- Greg Turk. 1992. 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*, James J. Thomas (Ed.). ACM, 55–64.
- Antônio Wilson Vieira, Armando Alves Neto, Douglas G. Macharet, and Mario Fernando Montenegro Campos. 2010. Mesh Denoising Using Quadric Error Metric. In *SIBGRAPI 2010, Proceedings of the 23rd SIBGRAPI Conference on Graphics, Patterns and Images, Gramado, Brazil, August 30 2010-September 3, 2010*. IEEE Computer Society, 247–254.
- Ruben Wiersma, Ahmad Nasikun, Elmar Eisemann, and Klaus Hildebrandt. 2023. A Fast Geometric Multigrid Method for Curved Surfaces. In *ACM SIGGRAPH 2023 Conference Proceedings, SIGGRAPH 2023, Los Angeles, CA, USA, August 6-10, 2023*, Erik Brunvand, Alla Sheffer, and Michael Wimmer (Eds.). ACM, 1:1–1:11.
- Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D ShapeNets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 1912–1920.
- Rui Xu, Longdu Liu, Ningna Wang, Shuangmin Chen, Shiqing Xin, Xiaohu Guo, Zichun Zhong, Taku Komura, Wenping Wang, and Changhe Tu. 2024. CWF: Consolidating Weak Features in High-quality Mesh Simplification. arXiv:2404.15661 [cs.GR]
- Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. 2018. PCN: Point Completion Network. In *2018 International Conference on 3D Vision, 3DV 2018, Verona, Italy, September 5-8, 2018*. IEEE Computer Society, 728–737.
- Cem Yuksel. 2017. Mesh color textures. In *Proceedings of High Performance Graphics, HPG 2017, Los Angeles, CA, USA, July 28 - 30, 2017*. ACM, 17:1–17:11.
- Jiayi Eris Zhang, Jérémie Dumas, Yun (Raymond) Fei, Alec Jacobson, Doug L. James, and Danny M. Kaufman. 2022. Progressive Simulation for Cloth Quasistatics. *ACM Trans. Graph.* 41, 6 (2022), 218:1–218:16.
- Jiayi Eris Zhang, Jérémie Dumas, Yun (Raymond) Fei, Alec Jacobson, Doug L. James, and Danny M. Kaufman. 2023. Progressive Shell Quasistatics for Unstructured Meshes. *ACM Trans. Graph.* 42, 6, Article 184 (2023).
- Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).

A BACKGROUND

Our method simplifies a triangle mesh by iterative edge collapses. We prioritize the sequence of edge collapses by our modified *quadric error metric*. Here, we briefly introduce the quadric error metric [Garland and Heckbert 1997] and some variants that are relevant to our method.

A.1 Quadric Error Metric

Let $\mathbf{p} \in \mathcal{M}$ be a point on a surface $\mathcal{M}$ embedded in $\mathbb{R}^3$ and $\hat{\mathbf{n}}$ be the (unit) normal vector at $\mathbf{p}$. The tangent plane at $\mathbf{p}$ is given by all points $\mathbf{x} \in \mathbb{R}^3$ that satisfy

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

The quadric error $E(\mathbf{x})$ measures the squared distance from any point $\mathbf{x}$ to the tangent plane at $\mathbf{p}$, which can be computed as

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

$$= (\mathbf{x} - \mathbf{p})^\top \hat{\mathbf{n}} \hat{\mathbf{n}}^\top (\mathbf{x} - \mathbf{p}) \quad (3)$$

$$= \mathbf{x}^\top \mathbf{A} \mathbf{x} + 2\mathbf{b}^\top \mathbf{x} + c, \quad (4)$$

with

$$\mathbf{A} = \hat{\mathbf{n}} \hat{\mathbf{n}}^\top, \quad \mathbf{b} = -\mathbf{A} \mathbf{p}, \quad c = \mathbf{p}^\top \mathbf{A} \mathbf{p} \quad (5)$$

A *quadric* Q refers to the triplet $(\mathbf{A}, \mathbf{b}, c)$, which are quantities derived by the plane equation $\mathcal{P}$ with a unit normal $\hat{\mathbf{n}}$ and a point $\mathbf{p}$ on the tangent plane:

$$Q := (\mathbf{A}, \mathbf{b}, c) = \mathcal{P}(\hat{\mathbf{n}}, \mathbf{p}), \quad (6)$$

This quadric Q gives us the complete information to compute the quadric error $E(\mathbf{x})$.

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

$$E_{ijk}(\mathbf{x}) = \mathbf{x}^\top \mathbf{A}_{ijk} \mathbf{x} + 2\mathbf{b}_{ijk}^\top \mathbf{x} + c_{ijk}. \quad (7)$$

This defines the *triangle quadric* Q_{ijk} as

$$Q_{ijk} = (\mathbf{A}_{ijk}, \mathbf{b}_{ijk}, c_{ijk}) = \mathcal{P}(\mathbf{n}_{ijk}, \mathbf{v}_i) \quad (8)$$

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

$$E_i(\mathbf{x}) = \sum_{ijk \in \mathcal{N}_i} a_{ijk}^i E_{ijk}(\mathbf{x}) \quad (9)$$

$$= \sum_{ijk \in \mathcal{N}_i} a_{ijk}^i (\mathbf{x}^\top \mathbf{A}_{ijk} \mathbf{x} + 2\mathbf{b}_{ijk}^\top \mathbf{x} + c_{ijk}) \quad (10)$$

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

We use $\mathcal{N}_i$ to denote the one-ring triangles of the vertex i and a_{ijk}^i 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 $a_{ijk}^i = a_{ijk}/3$ to be one-third of the face area a_{ijk} [Meyer et al. 2002]. This derivation gives rise to the definition of *vertex quadric* Q_i as a weighted summation of its one-ring triangle quadrics Q_{ijk}

$$Q_i = \sum_{ijk \in \mathcal{N}_i} a_{ijk}^i Q_{ijk} \quad (11)$$

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

$$E_{ij}(\mathbf{x}) = E_i(\mathbf{x}) + E_j(\mathbf{x}) \quad (12)$$

As we can see in Eq. 9 that summing up quadric errors leads to a summation of quadrics, this leads to the definition of an *edge quadric* Q_{ij} as

$$Q_{ij} = Q_i + Q_j. \quad (13)$$

A.2 Quadric Error Edge Collapses

Garland and Heckbert [1997] suggest simplifying a triangle mesh by iteratively collapsing the edge with the smallest edge quadric error $E_{ij}(\mathbf{x}^*)$ which is the value of E_{ij} evaluated at the optimal location $\mathbf{x}^*$ that minimizes E_{ij} . Specifically, given an edge ij , edge collapse is performed by replacing this edge with a new vertex i' that is located at $\mathbf{x}^*$. Let $Q_{ij} = (\mathbf{A}_{ij}, \mathbf{b}_{ij}, c_{ij})$, the optimal location $\mathbf{x}^*$ can be computed solving a linear system obtained from setting $\nabla E_{ij} = 0$

$$\mathbf{A}_{ij} \mathbf{x}^* = -\mathbf{b}_{ij} \quad (14)$$

If the matrix $\mathbf{A}_{ij}$ is invertible and well-conditioned, one can solve for $\mathbf{x}^*$ with standard linear solvers, such as Cholesky decomposition. If not, one can use the singular value decomposition to solve for $\mathbf{x}^*$ more robustly [Lindstrom 2000] or use the probabilistic quadric as a regularization [Trettner and Kobbelt 2020].

In subsequent iterations, instead of recomputing the edge quadric with Eq. 13, Garland and Heckbert [1997] 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, such an accumulation often leads to a more efficient and desirable decimation. But some applications may favor not accumulating quadrics [Hoppe 1999].

We use the term “memory” to denote the implementation that accumulates the quadric error metric throughout the decimation and “memoryless” to denote the implementation that recomputes the quadric at every iteration.

A.3 Area Quadrics

The quadric error metric has stimulated several variants to serve different purposes (see Sec. 2). Among them, the *area quadric* is related to our decimation metric. The area quadric measures the *squared area change* for each edge collapse. Lindstrom and Turk [1998] show that the squared area change for each edge ij can be written as a quadratic expression, thus giving birth to the area quadric Q_{ij}^A

$$Q_{ij}^A = \sum_{ab \in N_{ij} \cap \partial M} \left([s_{ab}]_{\times}^T [s_{ab}]_{\times}, -[s_{ab}]_{\times} t_{ab}, t_{ab}^T t_{ab} \right) / 2 \quad (15)$$

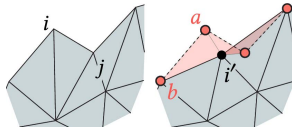
with

$$s_{ab} = v_b - v_a \quad t_{ab} = v_a \times v_b \quad (16)$$

where ab are ij 's one-ring edges that are also on the mesh boundary ∂M and $[\cdot]_{\times}$ is the *cross product matrix* defined as

$$[x]_{\times} := \begin{bmatrix} 0 & -x_2 & x_1 \\ x_2 & 0 & -x_0 \\ -x_1 & x_0 & 0 \end{bmatrix} \quad (17)$$

This area quadric Q_{ij}^A is derived from summing up the squared area (computed with cross products) from each boundary edge ab to the newly inserted vertex i' (see inset). This implies that this area quadric is merely an approximation to the actual area change because this measure ignores the cancelation between the area gain and the loss. But this proxy maintains the favorable quadratic expression and can be seamlessly incorporated into the quadric-based mesh simplification.



B THEORETICAL GUARANTEES

No Texture Bleeding. Our method guarantees no texture bleeding. This is because each texel (represented as a barycentric point) on the simplified mesh is guaranteed to be projected onto a face of the input mesh. This ensures that no texel will pick up the background color of the input texture, thus avoiding texture bleeding.

Manifold In, Manifold Out. Our simplification framework offers flexibility regarding topological changes. The vanilla configuration allows topological changes, such as merging components, to strive for high quality simplification (see Fig. 24). Alternatively, for applications requiring strict topology preservation on known manifold inputs, our method can be augmented with validity checks (e.g., those detailed in Appendix C of [Liu et al. 2020]), to reject edges that violate them. Including these checks ensures that our decimation maintains the manifold structure.

Convergence to QEM on Closed Manifolds. Our error metric converges to the edge quadric error metric [Garland and Heckbert 1997] when the input mesh is a closed manifold mesh. This property is guaranteed because we only augment the area quadric if the local edge one-ring contains boundary edges (as discussed in Sec. 4.2). Thus, for a closed manifold mesh without boundary edges, no area

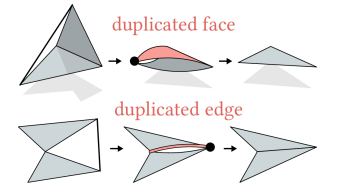
quadric will be included in the decimation, and our method converges to QEM up to some minor implementation differences, such as how to handle tied quadric error. This design decision ensures that our method improves LOD on meshes in the wild, without sacrificing the already amazing performance of QEM on manifolds.

C IMPLEMENTATION

In addition to the proposed improvements over the edge topology, the error metric, and the texture-preserving simplification part, reproducing our method requires changes in the implementation (e.g., data structures) to properly handle non-manifold meshes. Here, we focus on discussing the differences in implementation compared to the standard approach, such as in [Garland and Heckbert 1997], and omit the aspects that remain the same (e.g. using a priority queue to prioritize edge costs, employing a linear solve to determine the optimal vertex positions, etc.).

C.1 Simplicial Complex Data Structure

Many off-the-shelf mesh simplification methods are implemented using the *half-edge data structure* [Muller and Preparata 1978]. However, such a data structure is primarily designed to handle manifold meshes with implicit assumptions e.g. each edge is shared by one or two faces, making it inapplicable to defective meshes.



Several data structures for manipulating simplicial complexes have been proposed to handle general simplices (e.g., high dimensional simplices) and reduce the memory footprint [Floriani and Hui 2005]. However, in our case, these more generic data structures are unnecessary. We only perform edge collapses on simplicial 2-complexes. Thus, instead of using general-purpose data structures, we use a list of lists (similar to [Popovic and Hoppe 1997]) to represent the simplicial complex. Specifically, we construct three lists to store the *star* information for each (1) vertex to its one-ring edges, (2) vertex to its one-ring faces, and (3) edge to its one-ring faces. For each edge collapse, we use these lists to gather neighboring simplices and update their connectivity correspondingly. If a collapse leads to duplicated or degenerated simplices (see inset), we simply remove them from the corresponding lists.

C.2 Successive Texture Transfer

To support the texture transfer described in Sec. 4.3, for each edge collapse, we additionally store the local edge one-ring mesh before the collapse and the local vertex one-ring mesh after the collapse for closest point successive projection. This is because successive projections only requires updating texture samples within the one-ring neighborhoods, we can avoid redundant computations by skipping samples outside the one-ring (by checking the corresponding face of each barycentric point) and only perform the closest point projection within the one-ring meshes. In practice, the number of triangles within the one-ring is small, we did not observe significant efficiency improvement if one build spatial hierarchies for the local one-ring meshes.

D USER STUDY DETAILS

We performed two user studies to gather perceptual opinions on our mesh simplification results. Specifically, the participants were instructed to perform the user study within the context of a common online game scenario where the players are using low-end devices (e.g., mobile phones) to play online video games with other players. Due to the high performance requirements (e.g., frame rates, bandwidth), the 3D assets in the game have to satisfy a fixed budget (e.g., the total number of vertices in the scene). Under this situation, we asked participants to fill out online surveys about their preferences among a variety of simplified 3D assets obtained from our method and baselines.

One of the studies evaluated the quality of textured mesh simplification. For each response, we randomly selected a mesh from the PolyHaven dataset, simplified with our method and the textured QEM by Garland and Heckbert [1998], and asked the participant to select their preferences among (1) our method, (2) [Garland and Heckbert 1998], and (3) comparable (see Fig. 24 for some examples in the dataset). We included *comparable* as one of the options because our method “converges” to QEM when the input is a closed manifold mesh and this can avoid users randomly picking one out of comparable outputs. The results from the two methods are presented in an arbitrary order in the survey to avoid the order bias. From the 591 responses we collected, 81.9 % of them indicated that our method is either comparable or outperforming the baseline, suggesting that our method leads to better simplification results (see Sec. 5 for more details).

The second study solely focused on the quality of geometry (excluding textures) after simplification. We presented results from the method by [Trettner and Kobbelt 2020] and our invented baseline [G97+L98+T20] mentioned in Sec. 3. Participants were asked to select their favorite mesh simplification results out of the 10 user-created non-manifold meshes presented in the paper (see Fig. 25 for the collection) representing a wide range of meshes from organic shapes, 3D scenes, and man-made objects. From the total 690 responses (69×10), 82.5% of them favored our results, 10.4% favored the results from [Trettner and Kobbelt 2020], and 7.1% favored [G97+L98+T20] (see Sec. 5).

For these two studies, we invited 73 participants (51 male, 21 female, 1 non-binary individual, age ranging from 11-50, 75% graduate students and 25% working professionals including engineers and artists) to conduct the user study.

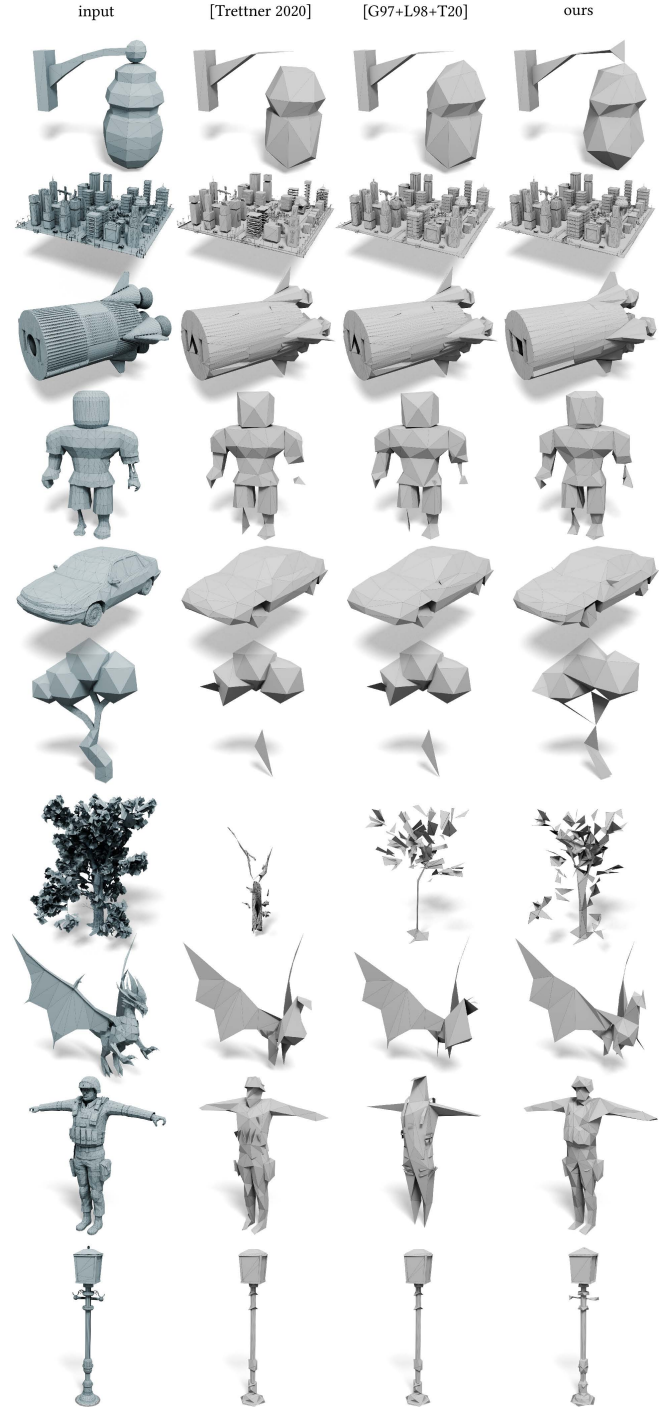


Fig. 25. We conducted a user study to evaluate the perceptual preference across different mesh simplification techniques, including (1) ours, (2) [Trettner and Kobbelt 2020], and (3) [G97+L98+T20] a combination of [Garland and Heckbert 1997; Lindstrom and Turk 1998; Trettner and Kobbelt 2020]. The figure contains all the meshes used in the user studies. Note that in the user study, the order of different methods is randomly permuted to avoid order bias, but in this figure we display them in a consistent order for clarify.