Micro-Mesh Construction

ANDREA MAGGIORDOMO, University of Milan, Italy
HENRY MORETON, NVIDIA, USA
MARCO TARINI, University of Milan, Italy

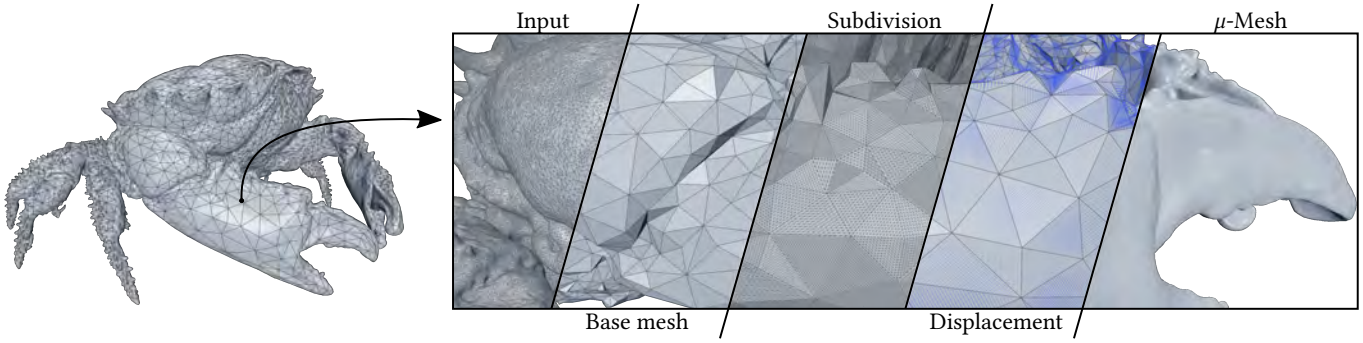


Fig. 1. A model automatically constructed using our algorithm, the Dark Fingered Reef Crab from https://threedscans.com. The input triangle mesh of 2,141,606 triangles is remeshed to a directly renderable micro-mesh representation of 21,312 base-triangles, with ray tracing accelerated by dedicated hardware. The micro-mesh is expanded on-demand to 2,223,780 displaced μ-triangles and achieves a 15:1 compression ratio.

Micro-meshes (μ-meshes) are a new structured graphics primitive supporting a large increase in geometric fidelity, without commensurate memory and run-time processing costs, consisting of a base mesh enriched by a displacement map. A new generation of GPUs supports this structure with native hardware μ-mesh ray-tracing, that leverages a self-bounding, compressed displacement mapping scheme to achieve these efficiencies.

In this paper, we present an automatic method to convert an existing multi-million triangle mesh into this compact format, unlocking the advantages of the data representation for a large number of scenarios. We identify the requirements for high-quality μ-meshes, and show how existing re-meshing and displacement-map baking tools are ill-suited for their generation. Our method is based on a simplification scheme tailored to the generation of high-quality *base meshes*, optimized for tessellation and displacement sampling, in conjunction with algorithms for determining *displacement vectors* to control the direction and range of displacements. We also explore the optimization of μ-meshes for texture maps and the representation of boundaries.

We demonstrate our method with extensive batch processing, converting an existing collection of high-resolution scanned models to the micro-mesh representation, providing an open-source reference implementation, and, as additional material, the data and an inspection tool.

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

Authors' addresses: Andrea Maggiordomo, University of Milan, Dipartimento di Informatica, via Celoria 18, Milan, Italy; Henry Moreton, NVIDIA, Santa Clara, California, USA; Marco Tarini, University of Milan, Dipartimento di Informatica, via Celoria 18, Milan, Italy.

0730-0301/2023/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Additional Key Words and Phrases: Micro-meshes, Remeshing, Mesh simplification, Displacement mapping, Mesh compression

ACM Reference Format:
Andrea Maggiordomo, Henry Moreton, and Marco Tarini. 2023. Micro-Mesh Construction. *ACM Trans. Graph.* 1, 1 (May 2023), 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Displaced micro-meshes (abbreviated μ-meshes) are a recently introduced polygonal representation for 3D models [NVIDIA 2022], designed for efficient GPU rendering, particularly ray tracing. μ-meshes are natively ray traced and greatly reduce mesh size and the size and construction time of acceleration structures.

A μ-mesh is a special type of displacement-mapped mesh, consisting of a coarse *base* triangular mesh and a set of scalar displacement values modeling the high-frequency details. At rendering time, each base-mesh face is subdivided into μ-triangles and the resulting μ-vertices are displaced along interpolated directions (details of the μ-mesh structure are provided in Sec. 1.1).

Hardware is available that natively supports the rendering of μ-meshes, unlocking unprecedented triangle counts in interactive applications such as videogames and virtual reality, and, ultimately improving the quality of real-time rendering.

The construction of μ-meshes is an unexplored area, whereas traditional hi-res meshes can be produced with a variety of well-established approaches, including manual digital sculpting, 3D acquisition techniques, subdivision surfaces, and more. To allow digital artists to produce μ-meshes, asset authoring pipelines would have to be devised anew or modified, incorporating new tools and modeling approaches.

In this paper, we explore the problem of the automatic production of a high-quality μ-mesh from a traditional high-resolution mesh. This is intended both as a bridge from traditional representations

to μ-meshes and as a tool in new potential asset creation pipelines designed to directly produce μ-meshes.

μ-Meshes share many characteristics with other forms of displacement-mapped surfaces, and a natural solution would be to resort to a pipeline of existing tools to convert an input mesh into a displacement-mapped surface (including tools for mesh decimation, displacement map baking, semi-regular remeshing, and so on). However, the μ-mesh scenario poses stringent requirements, and specific objectives (see Sec. 1.2). The primary motivation of the present work is to automatically construct a satisfactory μ-mesh, something very challenging with existing tools (see experiments in Sec. 10.8 and 10.9).

1.1 Background: the μ-mesh structure

In this section, we summarize aspects of the μ-mesh representation, the output of our method. For further details, see the white paper [NVIDIA 2022].

Base mesh. The base mesh of a μ-mesh is a triangle mesh with position and displacement vectors defined at the mesh vertices. Each base mesh face is divided into a regular grid of 4^k μ-triangles, specified by a per-face subdivision level k (Fig. 2-b). Note that each unmodified base mesh triangle edge is composed of 2^k segments.

Micro-displacements. At each μ-vertex the base mesh positions and direction vectors are barycentrically interpolated. The direction vector is then scaled by a per-μ-vertex scalar displacement value and added to the interpolated position to produce the displaced μ-vertex (Fig. 2d).

Water tightness. Bit-exact water-tightness between displaced base faces is achieved through shared base mesh vertex positions and displacement vectors, and duplication of displacement values on shared edges, plus by using a per base face edge decimation control bit to reduce the number of triangle edge segments by one half along the associated edge. This imposes a constraint on adjacent subdivision levels, they may differ by no more than one. We discuss how to satisfy this requirement in Sec. 6 (Fig. 2c).

Encoding. The scalar displacements associated with each base mesh face are stored in a micro-map (μ-map), a structure for storing per μ-vertex attributes. Each displacement μ-map contains $\left(2^k+1\right)\left(2^{k-1}+1\right)$, 11-bit normalized displacement values. The μ-map is laid out for efficiency of access and to facilitate compression beyond the quantization to 11 bits.

Rendering. The μ-mesh is designed with ray tracing in mind, allowing μ-triangles to be created as needed during rendering. By construction, the displaced surface is bounded by the hull formed by the base mesh face and the face defined by the tips of the displacement vectors; we refer to this as a *prismoid*. (Fig. 3). The bounding nature of the prismoid is exploited in the construction of the ray tracer's bounding volume hierarchy (BVH) as well as during the actual tracing of rays.

Texture mapping support. μ-Meshes can optionally support texture mapping by storing texture coordinates at base mesh vertices (as in any standard mesh) and linearly interpolating their values using the base mesh barycentric coordinates of the hit point. μ-meshes may also implement Mesh Colors with a μ-map of colors [Yuksel et al. 2010].

Multiresolution. μ-Meshes support four, watertight levels of detail. HW ray queries may include a LOD bias where all μ-meshes encountered during the trace are treated as though their subdivision level is reduced by up to three, one sixty-fourth the μ-triangles. Shader code can exploit the μ-mesh structure to arbitrarily reduce LOD, all the way to the base mesh itself. As long as LOD is uniformly reduced by lowering the subdivision level, the resulting mesh remains bit-exact watertight.

1.2 Problem statement and objectives

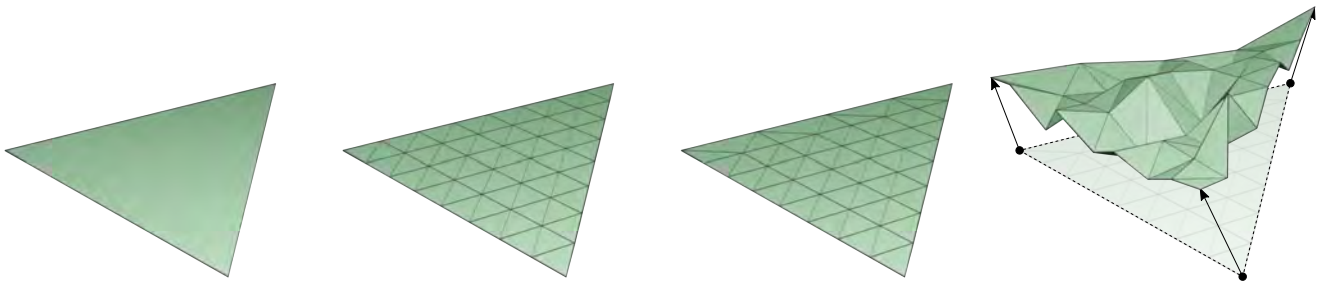
Motivation: why μ-meshes. The μ-mesh representation is designed to exploit the inherent coherence of high-quality geometry and possess qualities that significantly reduce the impact of using extremely detailed models. The primary value of μ-meshes is their extreme compactness in video RAM, while maintaining high-efficiency ray traced or rasterized rendering. The compression ratios mitigate the storage and transmission time costs of detailed models, and the intrinsic LOD amplifies these benefits by largely obviating the need to maintain multiple, redundant LODs. Additionally, because each μ-mesh face serves as a good bounding structure for its constituent μ-triangles, the BVH size and build time are greatly reduced when compared to an equivalent triangle-based model (build time improvements of 7-15× and size savings of 5-20× are reported in [NVIDIA 2022]). These strengths and the emerging hardware support make μ-meshes a rendering primitive that could make possible a dramatic increase in attainable geometric complexity.

To realize this benefit, a reliable construction method is needed, tailored to the specific needs of the μ-mesh structure. We want to produce an *accurate* and *efficient* μ-mesh representation of an input surface expressed as a high-resolution triangle mesh. With this high-level objective we see the need for trade-offs among a number of conflicting goals, due to the specifics of the μ-mesh representation:

Goal: Coarseness of base mesh. μ-Meshes are more memory efficient the coarser the base mesh. More geometric detail is placed in the scalar displacements with μ-triangles generated on demand. The current μ-mesh design supports high amplification factors, more than 1000 to 1. As a consequence, we seek a base mesh that is as much as three orders of magnitude more coarse than the input surface. However, as we have stated, this increased coarseness may conflict with other desirable properties. Note that a wider range of levels of detail may be provided by a coarser base mesh with higher levels of subdivision.

Goal: Reprojectability. To reproduce the input surface, the μ-mesh must hit it with the set of rays defined by the μ-mesh's interpolated displacement vectors. This requirement is both difficult to fulfill and to test for satisfaction. Still, the desire for reprojectability pulls on our algorithmic choices in multiple phases where it is in tension with the other objectives. Ideally, the displacement direction rays are locally orthogonal where they strike the input surface.

Goal: Isotropy. The regular grid of the subdivided μ-mesh means that their μ-triangles inherit their shape and aspect ratio (prior to displacement). A consequence is that roughly equilateral-shaped base triangles will tend to more efficiently sample the final surface. To preserve sampling efficiency we favor low aspect ratio base



(a) Base-mesh triangle (b) Regular subdivision into μ-triangles (c) Optional tessellation modification (d) μ-vertex displacement

Fig. 2. The μ-mesh geometric primitive. A base triangle is first refined using regular 1-to-4 subdivision, then border edges are optionally decimated to control the triangulation density of μ-mesh complexes while ensuring water-tightness, and finally, the refined geometry is displaced along linearly interpolated directions sampled at base vertices. Scalar micro-vertex displacements are normalized in $[0, 1]$.

triangles. Conversely, less accuracy is gained from samples taken by long, thin base triangles.

Goal: Minimal prismoid volume. Since the prismoid envelopes the represented surface (and μ-triangles), and because the base triangle area grows with mesh coarsening, the length of displacement vectors is an important factor in reducing prismoid volume. Shorter displacement vectors are desirable for two reasons: First, shorter vectors mean greater accuracy from a fixed number of scalar displacement bits, 11 in this case. Second, short displacement vectors mean the prismoid serves as a tighter bounding hull, improving culling and rendering performance. Since the prismoid is an envelope, we penalize a base mesh farther from the surface.

Finally, our method needs to be efficient, scalable, and robust, if it is to convert real-world, high-resolution models.

2 RELATED WORK

Efficient representations for high-resolution geometries. μ-Meshes belong to a long-lived line of research seeking methods and data structures to lower the resources (computing power and memory storage) for representing and rendering complex surfaces rich in geometric detail.

Progressive schemes are adaptive multi-resolution representations that control the tessellation density dynamically across the surface, significantly reducing the transition artifacts. Pioneered by Hoppe with the Progressive Mesh data structure [Hoppe 1996], these schemes encode the sequence of operations performed by an edge-collapse decimation of the high-resolution mesh and smoothly transition across resolutions by traversing the decimation sequence (forward to coarsen the mesh, and reverse to refine it). Early approaches encoded transitions at triangle granularity [De Floriani et al. 1998; Hoppe 1996; Puppo 1998], while more recent multi-resolution schemes adopt patch-granularity at a few thousand triangles to take advantage of the support of modern GPUs [Cignoni et al. 2003, 2004; Sander and Mitchell 2005; Yoon et al. 2004]. This trend culminated in Nanite [Karis et al. 2021], with its LOD hierarchy composed of 128 triangle clusters. *Normal Meshes* [Guskov et al. 2000] are similar to μ-Meshes in spirit, as they also represent the high-resolution surface by subdivision and scalar displacement. In a Normal Mesh, new vertices are displaced along the normal direction of the local frame of the subdivided surface *at the previous level*, making it a less ideal fit with hardware implementation.

Alternative approaches designed around fast GPU tessellation schemes and displacement mapping have also been proposed [Boubekeur and Schlick 2008; Lorenz and Döllner 2008; Schwarz and Stamminger 2009]. These approaches represent a high-resolution surface with displacement values stored in a texture image (e.g., height-fields), and adaptively refine and displace a low-resolution base (or control) mesh according to the desired level. Typically, the connectivity and density of the resulting finely tessellated mesh are determined by simple subdivision schemes that might be efficiently implemented in hardware or dedicated shader code. However, these works share the need to decode them into simple triangles so that a hardware-consumable ray tracing acceleration structure can be built. This results in significant BVH construction costs and a loss of compactness.

Remeshing and mesh simplification. One central task of our proposal is to construct the base mesh that serves as the starting point of tessellation and displacement. This can be seen as an instance of the remeshing or mesh-simplification problem, where the task is to optimize or coarsen the triangulation of a surface while preserving the original surface appearance. The literature on these methods is vast and spans several decades. We refer the reader to an appropriate survey [Khan et al. 2022; Luebke 2001].

A common trend, which we also follow, is to optimize the meshing through a sequence of carefully selected local operations [Hoppe et al. 1993] such as edge collapses, splits, flips, vertex smoothing [Botsch and Kobbelt 2004; Cohen et al. 1998; Garland and Heckbert 1997, 1998; Garland and Zhou 2005; Hoppe 1999; Hu et al. 2017; Lindstrom and Turk 2000; Surazhsky et al. 2003]. When remeshing is done to reduce the triangle count of the input mesh, the focus is on faithfully reproduction of the original surface, and state-of-the-art methods determine the decimation sequence and vertex spatial locations by approximating the geometric error produced by the mesh simplification. Most methods rely on vertex quadrics [Garland and Heckbert 1997] to estimate the geometric error and compute optimal vertex placements for the decimated mesh. Quadrics have been later extended to incorporate additional appearance information other than the surface geometry, such as colors [Garland and Heckbert

1998] and texture coordinates [Garland and Zhou 2005; Hoppe 1999]. Other methods incorporate texture-deviation terms to minimize texturing artifacts [Cohen et al. 1998], or visual differences [Lindstrom and Turk 2000], to guide the simplification; however, these methods are quite costly as they require rendering texture patches or the entire object *during* the simplification process. An alternative to the error quadrics is to compute the bidirectional Hausdorff distance on local patches to guide the remeshing operations [Hu et al. 2017].

Our goal to produce isotropic triangulations is the central focus of many semi-regular remeshing techniques. Among others, techniques include Centroidal Voronoi Tessellation (CVT) [Alliez et al. 2005; Liu et al. 2009; Yan et al. 2009] to field-guided methods [Jakob et al. 2015], to bounds to local remeshing operations [Hu et al. 2017] disallowing the production of unfavorably shaped triangles. This latter method shares similarities with our own but is not focused on other objectives, such as coarseness.

All the above methods are general-purpose, whereas ours is designed around the specific needs and goals dictated by our objective (Sec. 1.2). This gives it a clear advantage, as exemplified by the direct comparison we offer in Sec. 10.8 against a popular tool in this category [Cignoni et al. 2008].

Displacement map generation. Displacement mapping, i.e. the idea of representing a high-resolution surface by combining a coarse surface representation with displacement values sampling the small-scale surface detail, has a long history [Cook 1984]. The task of producing the values and storing them in a texture image can be seen as a form of texture baking [Cignoni et al. 1999; Lee et al. 2000], an operation commonly supported, today, by common 3D modeling suites [Adobe 2021; Blender Development Team 2022; Help 2021; Marmoset 2022].

The displacement values can consist of either vector or scalar data. Using scalars is clearly more efficient, but requires solving intricate problems.

One is the need to ensure "re-projectability", i.e., that the intended surface is expressible as a warped heights-field defined over the coarse representation. Several solutions have been proposed to address this problem [Collins and Hilton 2002; Lee et al. 2000], often tailored around specific cases, such as displacement of subdivision surfaces [Panozzo et al. 2011]. Our solution, defined for triangular meshes, compares favorably with similarly aimed existing method [Microsoft 2022] that combines displacement mapping and base-mesh creation (see Sec. 10.8).

Another closely related problem is the determination of the directions to use for the displacement. Studies focused on this subproblem already observed that good solutions coincide with the visibility directions [Tarini et al. 2003], and our solution, enunciated in Sec. 4, follow this route.

A recurring geometrical sub-problem that emerges in this context and other contexts is relative to the cone of viewing directions from which all the faces surrounding a vertex are visible without local occlusions. [Aubry and Löhner 2008; Cohen et al. 1997; Kim et al. 1995], Our algorithm (Sec. 4.1 can be seen as a more efficient way to determine if this cone is empty, and, if not, to find its centroid. [Cohen et al. 1997] resort to an approximate solution based on linear programming, which assumes that a solution exists; [Aubry and Löhner 2008] propose a brute-force solution which has $O(n^4)$ complexity in the number of faces; [Jiang et al. 2020] solve the optimization problem formulation directly with a constrained quadratic programming solver; both strategies are impractical to integrate into the iterative coarsening scheme that we use to generate a low-resolution base mesh.

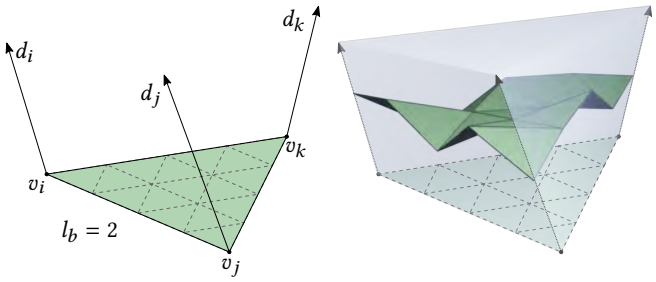


Fig. 3. A single base face, with per-vertex displacement vectors, corresponds to a bilinear prismoid that bounds the surface of the represented μ-mesh.

Bijective shells [Jiang et al. 2020] and high-order tetrahedral meshes [Jiang et al. 2021] have been recently proposed as a geometric framework for which attribute transfer and displacement map synthesis are primary applications. Their generation employs a similar approach to ours, based on local remeshing operations, but the input surface must satisfy strong assumptions to guarantee the existence of the bijection (manifold, self-intersection free) that are generally violated by many meshes in practice. We offer a comparison with [Jiang et al. 2020] in Sec. 10.9.

Displacement map and texture maps. One additional issue with any displacement map surface, which we face in Sec. 9.2, is that if the surface has additional textures (e.g. for storing colors), the displacement negatively contributes to the distortion of the texture map. This problem was observed in the past [McGuire and Whitson 2008; Zirr and Ritschel 2019], and existing solutions include compensating for the deformation by using indirect textures: a 2D offset field is computed and stored, and applied to texture accesses to counter the distortion introduced by the displacements. Our solution is tailored to the μ-mesh scenario, and, working on texture coordinates alone, conforms to the targeted data structure.

3 OVERVIEW

Our algorithm for constructing a μ-mesh representation of a given high-resolution input mesh M_I is structured in multiple phases.

In the first phase, we construct the base mesh M_B by coarsening M_I (Sec. 5). To do so, we adapt existing mesh iterative decimation approaches to pursue an appropriate trade-off between the goals of coarseness, isotropy, small displacement volume, and, especially, reprojectability (which we translate into characteristics we impose on the face normals on M_B).

A central sub-problem we have to address is to determine the per-vertex displacement *directions* on M_B (that is, the displacement vectors, including their magnitude). This is crucial for reprojectability, and we interleave this computation during the coarsening to ensure that M_B admits valid directions.

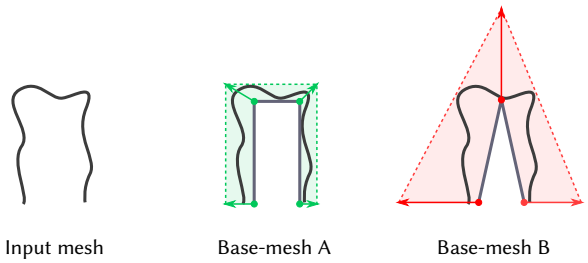


Fig. 4. In this example, using base-mesh A is drastically preferable to base-mesh B, as it results in a smaller displacement volume, and smaller quantization and aliasing issues. One way to detect this situation is to observe that the top vertex of base-mesh B has a much lower visibility value.

Ideally, the interpolated displacement directions should be as orthogonal as possible to both their origin on the base mesh and their intersection point on the input surface. Because the latter is expensive to enforce, we use a simpler proxy; we only consider the base mesh geometry and ensure that the interpolated displacement directions and local surface normals are consistently oriented. Because this criterion is also used to drive the coarsening, we discuss it in the next section (Sec. 4).

The next phase is to determine subdivision levels for each face of M_B pursuing the objectives of accuracy and memory parsimony, while abiding by the constraints of the μ-mesh structure (Sec. 6).

Next (Sec. 7), we produce the scalar displacement value for each μ-vertex, by ray-casting over M_I (similar to a standard displacement map baking). This also determines an initial estimation of the required magnitude of the displacement vectors.

Finally, we strive to minimize the displacement volume, redefining the magnitude of the displacement vectors (Sec. 8).

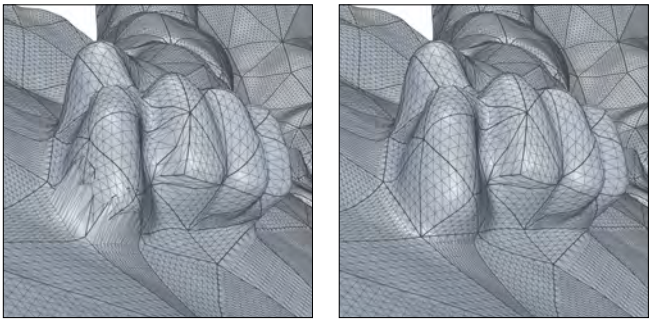
4 DETERMINATION OF DISPLACEMENT DIRECTIONS

In our context, the choice of displacement directions is critical. In this section, we show an effective criterion to guide this choice, that depends only on the base-mesh M_B. We employ this criterion both as guidance for the construction of a good M_B, and as a way to specify the actual displacement directions over it.

Similarly to standard per-vertex normals, displacement directions are unit vectors defined at mesh vertices, that point outward from the surface. This similitude would suggest that displacement directions can be constructed the same way as per-vertex normals are, e.g. with an area-weighted average of per-face normals. Doing so, however, often results in noticeable artifacts, exemplified in Fig. 5, showing that more care must be used in selecting the appropriate directions.

The criterion that consistently worked best in our scenario is, for a given base-mesh vertex v, to pick the direction that maximizes the orthogonality with *all* the base-mesh faces incident to v; specifically, the direction that maximizes the minimal orthogonality (measured as the dot product between the face normal and the direction d). We denote this maximal value as the *visibility* of the vertex v.

Input mesh (2M triangles)



(a) Vertex normals (b) Optimal visibility

Fig. 5. Choosing the wrong displacement directions can lead to visible geometric artifacts when the high-resolution surface is re-sampled by ray-casting along interpolated vectors. Left: results obtained with naive vertex normals (weighted average of the surrounding face normals). Right: results from the same base mesh using the optimal visibility directions computed as detailed in Sec. 4.1.

Formally, given a vertex v, let $\mathcal{N}$ be the set of face normals of all its adjacent faces; the visibility value $V(v)$ of v is given by

$$V(v) = \max_{\mathbf{d}\in\Omega} \left(\min_{\mathbf{n}\in\mathcal{N}} (\mathbf{d}\cdot\mathbf{n}) \right), \tag{1}$$

Ω being the set of unit directions. The maximizer of (1), $\mathbf{d}_{max}$, defines the displacement direction for vertex v, while the visibility value, which ranges from -1 (when $\mathcal{N}$ covers the entire Ω) to +1 (when the faces adjacent to v are all co-planar), serves a predictor of the quality of the μ-mesh in the region around v. Crucially, we restrict ourselves to strictly positive values, as negative values imply artifacts such as vanishing interpolated displacement directions inside base-mesh faces. Also, (1) does not necessarily admit a unique maximizer for negative values.

A problem equivalent to (1) emerged in a different context in [Jiang et al. 2020], where it is reformulated in a way that can be solved using Quadratic Programming (QP). This approach proved too inefficient for our case, where this task occurs inside nested loops (as we verify with the experimental comparison reported in Sec. 10.9). Instead, we devise the following algorithm.

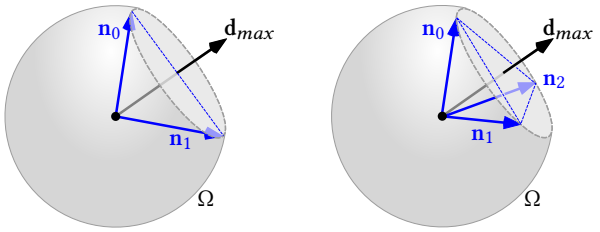


Fig. 6. A graphical representation of the solution in (3).

4.1 Finding the optimal visibility direction

Given $\mathcal{N}$, our algorithm finds $k = V(v)$ and $\mathbf{d}_{max}$ when k is strictly positive or detects when that is not the case, terminating with a failure. It has a similar structure to Welzl's algorithm [1991] for computing the minimum enclosing disk of a set of 2D points (the main difference is the necessity for us to detect failure cases).

We seek the maximal value k and a unit vector $\mathbf{d}_{max}$ subject to

$$\mathbf{d}_{max} \cdot \mathbf{n}_i \geq k \tag{2}$$

for all $\mathbf{n}_i$ in $\mathcal{N}$.

When $|\mathcal{N}| = 1$, the solution is trivially $k = 1$, and $\mathbf{d}_{max}$ is the only element in $|\mathcal{N}|$.

Otherwise, we use an iterative algorithm. At any point, we keep an active subset $\mathcal{N}' \subseteq \mathcal{N}$ with $|\mathcal{N}'| = 2$ or 3, and we derive a tentative solution as the normal with the largest k among all the normals fulfilling (2) in the equality sense for all $\mathbf{n}_i \in \mathcal{N}'$, disregarding the rest of $\mathcal{N}$. Specifically, $\mathbf{d}_{max}$ is the normalization of (see Fig. 6)

$$\mathbf{d}_{tmp} = \begin{cases} (\mathbf{n}_0 + \mathbf{n}_1) & \text{when } |\mathcal{N}'| = 2 \\ (\mathbf{n}_1 - \mathbf{n}_0) \times (\mathbf{n}_2 - \mathbf{n}_0) & \text{when } |\mathcal{N}'| = 3, \end{cases} \tag{3}$$

flipped if it has a negative dot product with any arbitrarily chosen $\mathbf{n}_i \in \mathcal{N}'$, and $k = \mathbf{d}_{max} \cdot n_i$ (in the rare cases when $k = 0$ the algorithm returns a failure).

We start by setting $\mathcal{N}'$ as two arbitrary elements of $\mathcal{N}$. At each iteration, we find the tentative solution ($\mathbf{d}_{max}$,k) for $\mathcal{N}'$, and check whether that solution fulfills (2) also for all other elements in $\mathcal{N}$; if so, the algorithm terminates, returning the tentative solution. Otherwise, the active subset $\mathcal{N}'$ is updated with the inclusion of an infringing $\mathbf{n}_i$, as follows.

We consider any non-strict subset of $\mathcal{N}'$ with cardinality 1 or 2, (a total of three possibilities, when $|\mathcal{N}'| = 2$, or six, when $|\mathcal{N}'| = 3$), and insert $\mathbf{n}_i$ into this subset, obtaining a new potential active set $\mathcal{N}''$ with cardinality 2 or 3. For each possibility, we solve with (3) over $\mathcal{N}''$ and test if the solution also verifies (2) for the (zero, one or two) elements of $\mathcal{N}'$ excluded from $\mathcal{N}''$. If no valid choice of $\mathcal{N}''$ is found, we determined that no $k > 0$ solution exists and the algorithm fails; otherwise, we set $\mathcal{N}'$ to the valid $\mathcal{N}''$ with the smallest cardinality and proceed with the next iteration.

For robustness to numerical precision issues, we test (2) within a small numerical tolerance (that is, a small constant value is subtracted from k).

Validity. In a spirit similar to [Welzl 1991], our algorithm stems from the observation that the optimal solution can always be identified by imposing up to three constraints in (2) to be fulfilled as equalities. This is because any choice of ($\mathbf{d}_{max}, k$) that fulfills more than three constraints in (2) as equalities can be equivalently identified by imposing *any* three of them as equalities. Moreover, it is easy to verify that, when $\mathcal{N}'$ has two elements, then no possible choice that fulfills (2) for both elements can improve the tentative solution; when $\mathcal{N}'$ has three elements, and no pair produces a tentative solution satisfying (2) for the third, then no choice that fulfills (2) for all three elements can improve the tentative solution; this implies that when a tentative solution is found that verifies (2) for every element in $\mathcal{N}$, it must be optimal.

Termination guarantee. Even if an element is initially excluded from $\mathcal{N}'$ it can be reintroduced in a later iteration, the algorithm is guaranteed to terminate, as the value for k necessarily decreases at each iteration.

Estimation of algorithmic complexity. We do not have an a priori bound on the number of iterations, beyond the $O(|\mathcal{N}|^3)$ number of possible candidate subsets. However, from empirical testing over hundreds of millions of problem instances, we found that the number of iterations surpassed $|\mathcal{N}|$ in fewer than 10^{-5} of the cases, and never reached twice $|\mathcal{N}|$.

5 BASE MESH GENERATION BY COARSENING

We generate the base mesh by performing a sequence of local coarsening operations on M, *edge-collapses* [Hoppe et al. 1993], progressively turning it into M_B. An edge-collapse contracts an edge into a vertex, which is positioned in some appropriate position, removing its two adjacent faces from M. Our coarsening framework follows the well-established mesh decimation approach first presented in [Garland and Heckbert 1997], adapting it to our scenario.

Initialization. As normal, we associate to each vertex v_i of M a quadric error function Q_i [Garland and Heckbert 1997] that is initialized, for each vertex, as the sum of the quadric functions measuring the squared distances from the planes of each face adjacent to v_i. We also associate to each vertex v_i a normal $\mathbf{n}_i$, thus defining a local tangent plane at each vertex.

5.1 Modified edge-collapse

Isotropy of M_B is crucial for us, therefore we modify the edge collapse operation toward fulfilling this goal, by tweaking the position of the vertex generated by every collapse to make the resulting faces more regularly shaped, compatible with other objectives. Specifically, given an edge $e = (v_i, v_j)$, we first compute the edge quadric Q_e as usual as the weighted average of the two vertex quadrics Q_i and Q_j. We find the tangent smoothing position $\mathbf{p}$ that improves the aspect ratio of the triangles after the collapse; after [Botsch and Kobbelt 2004], we define $\mathbf{p}$ as the barycenter of e (the average position of the union of the stars of v_i and v_j), and, we project $\mathbf{p}$ on the tangent plane associated with either v_i or v_j; we pick the one yielding a lower value of $\mathrm{Q}_e(\mathbf{p})$. Then, we define a "smoothing" quadric that measures the squared distance from $\mathbf{p}$:

$$\mathrm{Q}_s(x) = \|\mathbf{x} - \mathbf{p}\|_2^2, \tag{4}$$

and we combine it with the original quadric using a weight λ

$$\mathrm{Q}'_e = \mathrm{Q}_e + \lambda \mathrm{Q}_s. \tag{5}$$

We take the minimizer of Q'_e as the position for the vertex generated by the collapse. The effect of Q_s is to pull the resulting vertex toward $\mathbf{p}$, without forcing it in areas resulting in an excessive geometric distortion. In our experiments, we use $\lambda = 0.1$.

5.2 Operation cost evaluation

At the core of mesh coarsening techniques is the strategy to evaluate potential operations, in order to select the ones to perform, and, crucially, the order of execution. In our case, for a given potential collapse of edge e, we simulate it and evaluate a set of *costs* estimating its adverse effect on our objectives. Specifically, if v is the vertex position resulting from the collapse, and f_i the faces sharing v in the potential configuration after the collapses:

- $C_\mathrm{g}(e)$ estimates the geometric error increase, and is evaluated by applying the combined error quadric Q_e at the position of v;
- $C_\mathrm{n}(e)$ measures the maximum deviation from the normal of f_i to its original normal in the initial mesh, computed as their dot product (or 0 if negative);
- $C_\mathrm{a}(e)$ estimates the impact on the shape of triangles, and is measured as the worst aspect ratio (see below) of any f_i;
- $C_\mathrm{v}(e)$ estimates the impact on the reprojectability as the visibility of $V(v)$ (equation 1) after the collapse (or 0 if negative).

C_g is best at 0, larger values denoting more geometric discrepancy being introduced; C_n, C_a, and C_v are factors between 0 (worst) to 1 (best).

C_g and C_n contribute to the geometrical similarity between M and M_B; C_n also prevents the creation of folded configurations, where a triangle of M survives in M_B with an incoherently oriented normal.

C_v is crucial in ensuring reprojectability and drastically reduces the displacement volume. For example, it discourages the edge collapses that would turn configuration A into configuration B in Fig. 4.

C_a favors the creation of quasi-equilateral triangles. After [Field 2000], we quantify the aspect-ratio of a triangle T as twice the ratio of the inradius to the circumradius, resulting in a number from 0 for degenerate triangles to 1 for perfectly equilateral triangles:

$$\mathrm{aspect}(T) = \frac{16 \cdot \mathrm{area}(T)^2}{(s_0 + s_1 + s_2) \cdot s_0 \cdot s_1 \cdot s_2} \tag{6}$$

(but 0 if any s_i is 0), where s_i are the edge lengths of T.

Aggregate cost. We aggregate the costs of collapsing edge e into a single scalar value $C_\mathrm{tot}(e)$, with

$$C_\mathrm{tot}(e) = \frac{C_\mathrm{g}(e)}{C_\mathrm{n}(e)^{\omega_\mathrm{n}} \cdot C_\mathrm{a}(e)^{\omega_\mathrm{a}} \cdot C_\mathrm{v}(e)^{\omega_\mathrm{v}}} \tag{7}$$

The exponents $C_{\mathrm{n,a,v}}$ are constants controlling the relative importance of the penalty terms; for our experiments, we always use $\omega_\mathrm{n} = 0.1$, $\omega_\mathrm{a} = 0.5$, and $\omega_\mathrm{v} = 0.5$.

Disallowed operations. If $C_{\mathrm{n,a,v}}$ is 0, the aggregated cost diverges to ∞ and the operation is never performed. Additionally, we set a bound to the geometric error $C_\mathrm{g}^\mathrm{max}$, which we default to the square of 0.01 times the of the bounding box diagonal of M, and also disallow operations with an associated cost C_g exceeding $C_\mathrm{g}^\mathrm{max}$.

5.3 Operation prioritization strategy

At any given step, we have a potential operation for each edge of the current mesh, except for the ones which would compromise two-manifoldness [Dey et al. 1999]. Ideally, we would like to perform operations in reverse order of aggregated costs, but this is costly to compute.

We employ two strategies: at the beginning of the process, we employ an *approximate* randomized strategy, which is faster but only approximates the ideal order. When the number of faces drops below a threshold (we always used 1 million), and operation order becomes more crucial, we switch to a slower, more *accurate* strategy, that guarantees the ideal order of collapses.

The rationale is that, as is well-known from previous literature [Puppo 1998], several sequences of local simplification operations lead to the same final mesh, and so the order of operations is less crucial the furthest the current mesh is from the final base mesh.

In the *accurate strategy*, we score all potential operations with their costs and store them in a priority queue (implemented as a heap). Operations are extracted from the top of the queue and performed in succession. Each edge collapse resulting in the creation of vertex v invalidates all previously scored operations at edges in the entire 2-star of v (including edges at its boundaries). Invalidated operations are flagged (so that they are ignored and discarded when they reach the top of the queue), and substituted with new operations, which are reevaluated for costs and reinserted in the queue. This strategy is costly because each performed operation requires the evaluation of an average of 42 potential operations.

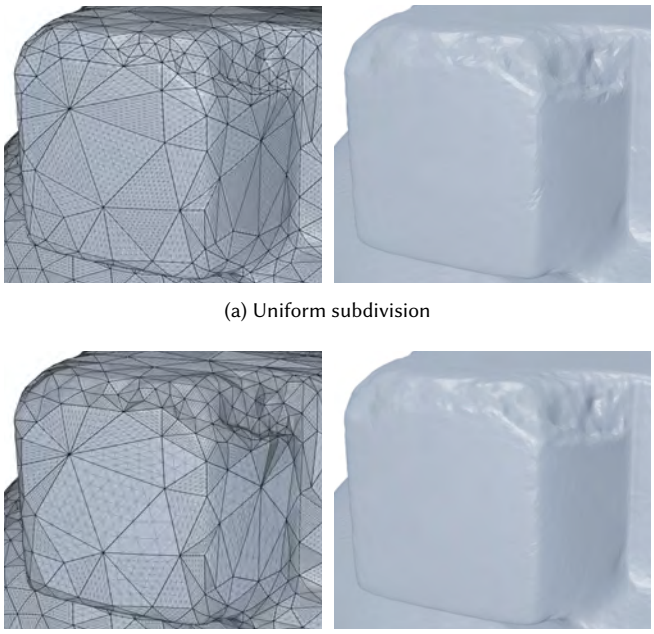
The *approximated strategy* is a heuristic where only n_{op} randomly selected potential operations are evaluated for cost, and the least costly one is either performed, or discarded if its cost exceeds an *adaptive cost threshold* C_max. The advantage of this strategy is that it bypasses the need to upkeep the queue and to pre-evaluate and re-evaluate operation costs, resulting in a much faster process. The parameters n_{op} and C_max balance between adherence to the ideal order and computational time, and we determined them by repeated testing. We use $n_{op} = 3$ and increase C_max by 30% every 20 consecutive rejected operations; to find an initial guess for C_max we pick the lowest cost from a set of 50 randomly sampled collapses.

5.4 Strict isotropy enforcement

In our context, face isotropy is an important goal (Sec. 1.2); specifically, faces in M_B with an aspect ratio much lower than 0.5 can be considered extremely undesirable, especially for larger base-mesh triangles that will be tessellated more densely. Ideally, it would be desirable to avoid, almost completely, any such faces in M_B, something that we failed to achieve by solely manipulating the cost function.

A natural strategy would be to just disallow any edge collapse producing a face with an aspect ratio smaller than a given threshold value a_{max}, irrespective of its cost. We found that this strategy causes the simplification process to run out of viable operations too early, in several areas of M (Fig. 7, middle). Instead, we keep track, for each face, of the best aspect ratio experienced by that face during the simplification, and only disallow an operation if it produces a face worse than a_{max} *and* also worse than its best aspect ratio minus a fixed delta (we used 0.1). Combined with the

Fig. 7. Decimating a 1.5M faces input mesh M (left) to 15K faces while enforcing a minimal aspect ratio threshold of 0.4. Middle: direct enforcement of this constraint locks all operations early during the simplification in certain areas, resulting in clusters of small faces in M_B. Right: our strategy (see text) counters this effect.



(a) Uniform subdivision

(b) Adaptive subdivision

Fig. 8. Comparison between uniform and adaptive subdivision strategies.

smoothing strategy (Sec. 5.1) this is effective at enforcing the desired constraint on aspect ratio without hindering the coarsening (Fig. 7, right).

5.5 Stopping criteria

We run the coarsening until all residual operations are disallowed for either having an infinite cost (with equation 7), exceeding the geometric error $C_{\mathrm{g}}^{\max}$, or violating the isotropy criteria. This means that our system determines on its own the coarsest base mesh and therefore the most efficient representation of the input surface as a μ-mesh. Optionally, a maximal number of base-mesh faces can also be set, and the coarsening stops as soon as that number is reached.

Our simplification algorithm preserves positive visibility in all produced vertices, while safeguarding other properties of common remeshing and simplification algorithms, to prevent inversion of normals or loss of manifoldness [Dey et al. 1999].

Even if we never introduce vertices failing to admit a positive-visibility direction, such configurations may be present in the input. In such cases, we allow the simplification to proceed by setting the visibility score of a collapse $C_{\mathrm{v}}(e)$ to a very small positive value: in our experiments, this solves the problem because the input mesh undergoes a multitude of coarsening operations that tend to fix such issues without ever introducing new ones (in any case, any surviving problems are strictly local).

6 DETERMINATION OF REFINEMENT LEVELS

After computing the base mesh, the tessellation factor 2^k must be determined for each of its faces, taking advantage of the μ-mesh ability to vary the tessellation levels across base-mesh faces.

We roughly target a given number of μ-triangle count $|F_\mu|$. By default, we set $|F_\mu|$ at the number of faces of the original mesh.

We devise two different strategies, targeting either uniform μ-triangle sizes or sizes of μ-triangle that are adaptive to the local shape complexity. Depending on the context, either can make sense. For example, if textures are forfeited in favor of per-μ-triangle data (see Sec. 9.2), uniform μ-triangle can be desirable. A comparison is shown in Fig. 8.

Uniform μ-triangle area. A regular subdivision of a triangle at level l produces 2^{2l} μ-triangles; therefore, an estimation of the constant subdivision level that would produce the desired number of micro-triangles can be derived as

$$l = \frac{1}{2}\log_2\left(\frac{|F_\mu|}{|F_B|}\right). \tag{8}$$

This gives an estimation for the expected average μ-triangle area a_m as

$$a_m = \frac{\mathcal{A}_B}{|F_\mu|} = \frac{\mathcal{A}_B}{2^{2l}\cdot|F_B|}, \tag{9}$$

where $\mathcal{A}_B$ is the total area of the base mesh. Then, for a given base triangle b, its estimated number of μ-triangles is simply the ratio of its area a_b to the average micro-triangle area:

$$|F_\mu^b| = \frac{a_b}{a_m}. \tag{10}$$

and we set the subdivision level of a base triangle b as the rounding to the nearest integer of

$$s_b = \frac{1}{2}\log_2|F_\mu^b| = l + \frac{1}{2}\log_2\left(\frac{a_b}{a_B}\right), \tag{11}$$

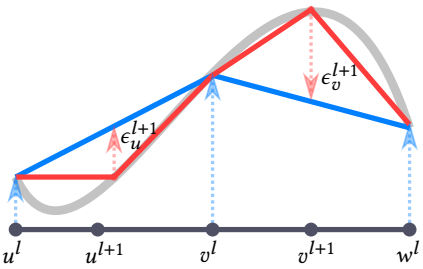


Fig. 9. Error estimation for adaptive subdivision. The current subdivision level l and the displaced micro-geometry are shown in blue. The displacements of the micro-vertices added at level $l+1$ are shown in red.

This has a straightforward interpretation as the global subdivision level l plus a correction that depends on the base triangle area a_b relative to the average base triangle area $a_B = \mathcal{A}_B/F_B$.

Adaptive μ-triangle area. In this strategy, we use an iterative algorithm to determine subdivision levels of base triangles, driven by the residual displacement error incurred by the actual displacement of μ-vertices.

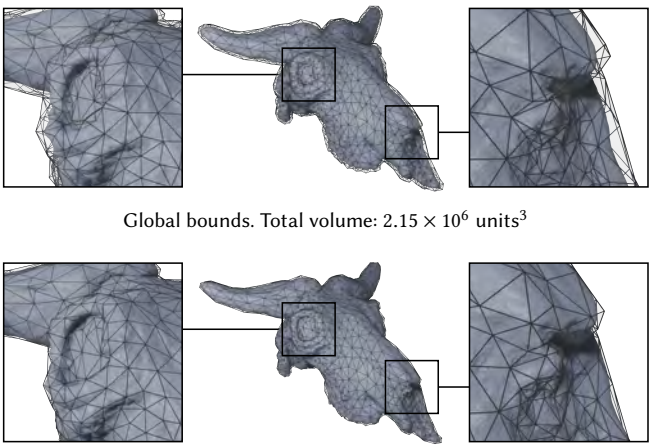
Start with $l = 0$ at all faces (corresponding to no refinement), and for each face, we predict the effect on the geometric error of a potential increase of the subdivision level by one. To do so, we simulate the subdivision level increase, project all the newly introduced μ-vertex on the input mesh, and record the maximum distance from any of them to the surface represented by the current subdivision level (Fig. 9).

We store in a priority queue all potential resolution increase operations (always exactly one per base-mesh face), sorted according to the predicted effect on the geometric error; we extract and perform the most impactful refinement operation (reinserting in the queue a new operation representing a further increase of resolution), until the targeted number of micro-faces is reached.

Enforcing water-tightness. Both strategies operate on base triangles in isolation, potentially infringing the μ-mesh constraint about neighboring faces differing by at most one tessellation level (under the penalty of introducing gaps between base faces). The constraint is enforced by a correction phase, where all pairs of adjacent faces with a difference larger than one are corrected by (conservatively) rising the lower tessellation level to the higher level minus one, until no such pair remains. Finally, we set the edge decimation flags at each pair of faces with a mismatching tessellation level, ensuring a watertight μ-mesh surface.

7 DETERMINATION OF SCALAR DISPLACEMENT VALUES BY RAY-CASTING

Once the base mesh geometry is defined, we tessellate each base face according to its subdivision level, and determine the scalar displacement values; this is done by ray-casting each μ-vertex from its position on the base mesh, along the interpolated directions, against the original mesh. For each μ-vertex, we record its *unnomralized* displacement values as the parametric position, on the ray, of the intersection with the input mesh. These values can be negative or larger than 1, because the displacement vectors are unitary, and the input mesh surface will in general run both above and below the base mesh.



Global bounds. Total volume: 2.15×10^6 units^3

Local bounds. Total volume: 0.63×10^6 units^3

Fig. 10. Displacement vector lengths determined with global vs. local bounds.

To accelerate the ray-casting procedure, we use a BVH of axis-aligned bounding boxes for the input mesh. We ensure robustness to self-intersections and other geometric artifacts in the input mesh, by adopting standard mitigation techniques, such as only intersecting the high-resolution surface at faces that are coherently oriented with respect to the ray.

We also exploit the smoothness of the scalar displacement field to improve robustness with respect to outlier rays and missed intersections. If we detect intersection outliers (e.g. rays that travel too far before hitting the input surface), we clear the corresponding scalar displacements and replace them by interpolating the displacement distances from valid neighbors.

8 OPTIMIZATION OF BASE DISPLACEMENT LENGTHS

At this point, we determine the length of the displacement vectors, striving to keep these lengths short so as to reduce the volume of the displacement prismoids, as per our goal (Sec 1.2).

We shift the base mesh and assign a length to all its displacement vectors to ensure that the displacement volume encapsulates the entire displaced surface.

We identify, for each base-mesh vertex v, the global minimum and maximum unnormalized displacements values $\bar{\delta}_{min}$ and $\bar{\delta}_{max}$, among all μ-vertex in the star of faces around v. Then, the position $\mathbf{p}_v$ and displacement vector $\mathbf{d}_v$ of v is updated to

$$\mathbf{p}_v \leftarrow \mathbf{p}_v + \bar{\delta}_{min}\mathbf{d}_v, \tag{12}$$

$$\mathbf{d}'_v \leftarrow (\bar{\delta}_{max} - \bar{\delta}_{min})\mathbf{d}_v. \tag{13}$$

While not necessarily optimal, this simple strategy results in tight displacement volumes.

Scaling each displacement direction differently causes the orientation of the interpolated directions to change slightly, which requires

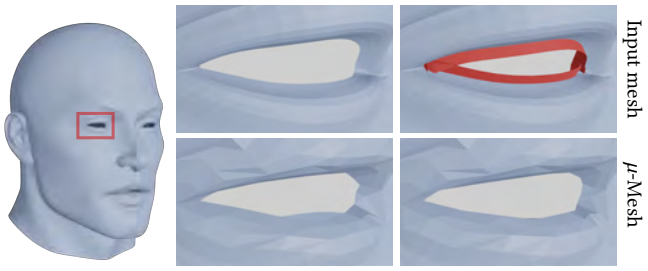


Fig. 11. In order to more accurately reproduce the boundary curves of the input mesh, we can use displacement directions that are **tangential** to the surface (right), rather than orthogonal, at μ-mesh boundaries. In these cases, we also add, in the input mesh, a vertical "wall" (in red) made of small polygons orthogonal to the surface boundary, as a target to be hit by the ray-casting procedure.

repeating the ray-casting phase to update them; additionally, due to curvature, even the updated displacements are not guaranteed to be limited to the 0 to 1 interval.

A simple way to bypass this costly update would be to define $\bar{\delta}_{min}$ and $\bar{\delta}_{max}$ to be the global bounds of the un-normalized displacement values across the entire surface; the scalar values can then be normalized by a linear mapping. However, this strategy leads to much larger displacement volumes, by more than a factor of 3 in our experiments (see Fig. 5).

As a time-saving approximation, at the expense of introducing only a negligible increase in the surface approximation error, we resort to a quick update of displacement values after changing the displacement vectors at base-mesh vertices: for each μ-vertex, we simply re-project the displaced position on the old ray into the newly defined ray.

9 EXTENSIONS

We extend our general framework for μ-meshes construction to tackle two cases of practical importance: μ-meshes enriched with traditional texture data, and open μ-meshes featuring semantically important boundaries.

9.1 μ-Meshes with detailed boundaries

When the input surface is open, its boundary curves are subject to being coarsened, losing their geometric details. In many scenarios, this can be acceptable, but there are cases where the shape of the boundary is semantically important; consider for example the eye-shaped holes depicted in Fig. 11.

A straightforward solution is to flag border edges and disallow coarsening them, so as to keep them finely tessellated in the base mesh; this is a trade-off with an obvious cost in terms of increased base-mesh complexity and triangulation quality. We propose a cheaper alternative where we let boundary edges be coarsened in the base mesh, and we reuse the displacement map mechanism to recover the original boundary shape in the μ-meshes.

To this end, we redefine displacement directions at boundary vertices of the base mesh to be *tangent* to the surface, rather than approximately orthogonal to it. We define such directions so that they are both tangential to the surface *and* orthogonal to the open boundary edges of the base mesh. The boundary μ-vertices will thus be displaced to form the boundary curve. This trades, in the proximity of the boundary, the ability to express height fields on the surface with the ability to reproduce a detailed boundary curve.

To define appropriate scalar displacement values, prior to the ray-casting phase, we enrich the input mesh with a band of "flap" triangles departing from the boundaries, orthogonally to the surface. These triangles are designed to be hit by the rays stemming from the μ-vertices at the boundary of the μ-mesh (see Fig. 11).

9.2 Texture-mapped μ-meshes

In many cases, the input mesh can come with its own set of original texture maps (storing, for example, colors), and our μ-mesh construction procedure may optionally be required to preserve the textured signal information in the final μ-mesh. There are several routes to accomplish this result.

The increased triangle count afforded by μ-meshes opens the possibility to avoid using traditional texture maps and simply re-bake the input textures into attributes stored at μ-vertex, using the μ-mesh schema. This resulting solution closely resembles in concept the Mesh-Color approach to texture mapping [Yuksel et al. 2010].

In other cases, however, it can still be desirable to store the original signal in traditional textures. As we discussed in Sec. 1.1, the μ-meshes schema accommodates texture coordinates, just like any standard mesh: simply, UV-coordinates are stored at base mesh vertices and interpolated linearly, during rendering, at each μ-vertex.

This route requires generating a parametrization, or UV-map, for the base mesh (including texture cuts). Many automatic or semiautomatic tools can be used for this purpose, treating the base mesh as a traditional triangle mesh. A vast literature exists covering this task [Sheffer et al. 2006], and it is beyond the scope of this work to discuss it; we observe that the task is eased by the relative coarseness of the base mesh. Considering the intrinsic multi-resolution nature of μ-mesh, this route constitutes an excellent answer to the open problem of how to share textures or UV-maps across different levels of detail [Yuksel et al. 2019].

A residual problem, however, remains. The parametrization is unaware of the displacements, which negatively affect it by aggravating **texture distortions**. We introduce a way to re-optimize any existing parametrization defined on the base mesh of a μ-mesh to account for its actual shape. This strongly mitigates the problem and warrants the use of any existing tool to UV-map μ-meshes.

Traditionally, UV-maps are optimized by defining a function measuring the distortion of the map (e.g. angle or area distortions), and minimizing this function over the space of all per-vertex UV-assignment.

We reformulate the distortion function so that it sums the contributions across all *displaced* micro-triangles instead, whose vertices are expressed as convex combinations of base mesh texture coordinates, and minimize this function over all base vertex UV-assignments.

This concept can be applied to any distortion energy function; we used the ARAP energy [Liu et al. 2008], which has the benefit of penalizing both angle and area distortions, resulting in a maximization

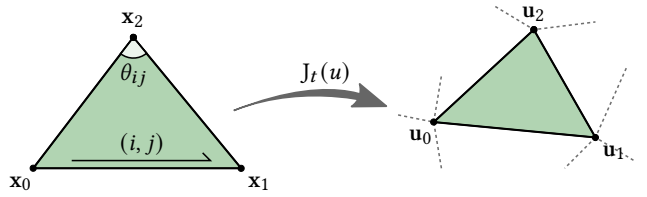


Fig. 12. Notation used for the ARAP energy formulation.

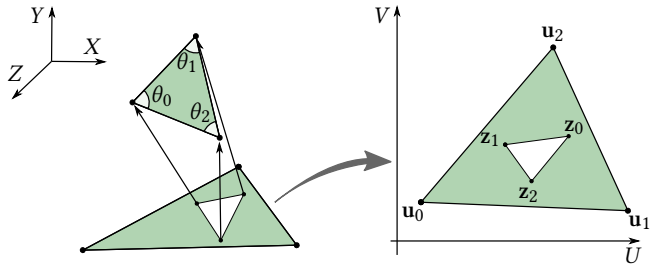


Fig. 13. Notation for the barycentric ARAP energy formulation.

of isometry, a crucial property for texture mapping. On standard triangle meshes, ARAP energy can be efficiently minimized using a local/global minimization scheme, which we adapt to μ-meshes, as follows.

Recap: ARAP optimization (for standard meshes). Let t be the index of a mesh triangle $\mathbf{x}_t = (\mathbf{x}_0^t, \mathbf{x}_1^t, \mathbf{x}_2^t)$ and, and let $\mathbf{u}_t = (\mathbf{u}_0^t, \mathbf{u}_1^t, \mathbf{u}_2^t)$ be the 2D texture coordinates associated to t. Then, let $\mathrm{J}_t(u)$ be the 2×2 Jacobian of the map from $\mathbf{x}_t$ to $\mathbf{u}_t$, and $\mathrm{R}_t \in \mathcal{R}$ an auxiliary matrix encoding a 2D rotation.

The ARAP energy [Liu et al. 2008] is defined as

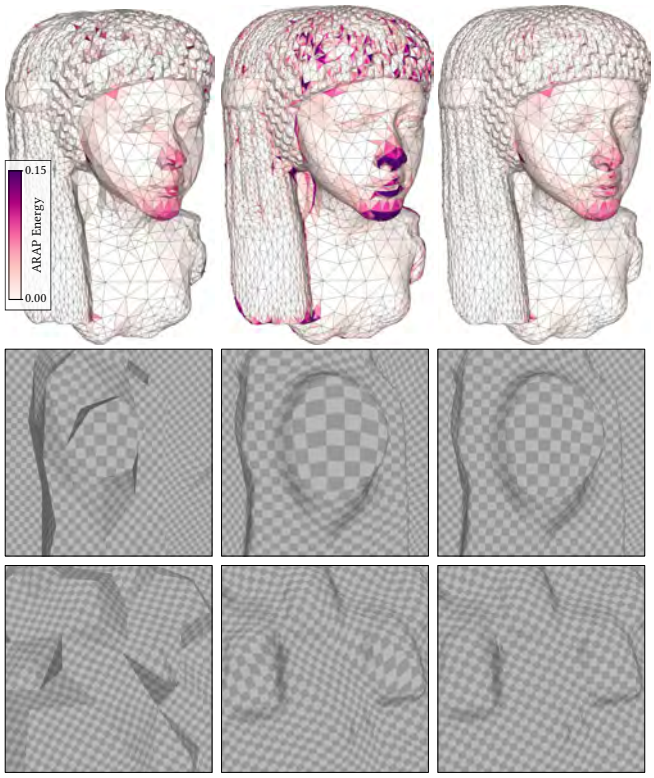
$$E(\mathbf{u}, \mathcal{R}) = \frac{1}{2} \sum_{(i,j) \in he} \cot \theta_{ij} \|(\mathbf{u}_i - \mathbf{u}_j) - \mathrm{R}_{t(i,j)}(\mathbf{x}_i - \mathbf{x}_j)\|^2 \quad (14)$$

where $t(i,j)$ is the triangle to which half-edge (i,j) belongs and θ_{ij} is the opposite angle within $t(i,j)$ (Figure 12).

This energy is minimized by alternating local and global steps: in the *local* steps, the parametric coordinates $\mathbf{u}$ are kept fixed, and the best fitting rotation matrices L are determined per triangle by computing the SVD decomposition of $\mathrm{J}_t(\mathbf{u}) = \mathrm{U}\Sigma\mathrm{V}^\top$ and setting $\mathrm{R}_t = \mathrm{U}\mathrm{V}^\top$; in the *global* steps, the rotation matrices are kept fixed, yielding a quadratic energy in $\mathbf{u}$ (equation 14) that can be minimized by setting the gradient to zero and solving the resulting sparse linear system.

Extending ARAP to μ-meshes. Given a displaced micro-triangle t, its UV coordinates $\mathbf{z}_i$ in parameter space are expressed as a linear combination of the base UV coordinates $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2$ (Figure 13):

$$\mathbf{z}_i = \mathrm{U}\mathbf{w}^i = \begin{pmatrix} | & | & | \\ \mathbf{u}_0 & \mathbf{u}_1 & \mathbf{u}_2 \\ | & | & | \end{pmatrix} \begin{pmatrix} w_0^i \\ w_1^i \\ w_2^i \end{pmatrix}, \; i = 0, 1, 2 \quad (15)$$



UV-mapped base mesh | after μ-mesh displacement | μ-mesh after UV re-optimization

Fig. 14. Our minimization of texture distortions for μ-meshes.

where the base UV coordinates have been arranged as columns of a 2×3 matrix U multiplying the vector of the barycentric weights $\mathbf{w}^i$.

In the *local* step, we compute the best-fitting rotation R_t associated with each displaced micro-triangle t. This is easily computed as the linear map between the isometric 2D projection of the *displaced* micro-triangle $\mathbf{x}_t$ and its interpolated parameter-space position $\mathbf{z}_t$.

When solving the *global* step, the energy term associated to a generic "micro" half-edge $(i, i+1)$ which can be written in quadratic form:

$$E(i, i+1) = \cot \theta_{i+2} \|(\mathrm{U}(\mathbf{w}^i - \mathbf{w}^{i+1}) - \mathrm{R}_t(\mathbf{x}_i - \mathbf{x}_{i+1})\|^2, \quad (16)$$

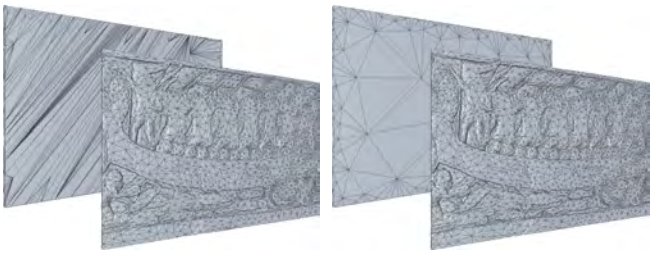
with R_t encoding the rigid alignment of the isometric 2D projection $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$ of the *displaced* micro-triangle to its UV counterpart.

The gradient of the energy term (16) with respect to $\mathbf{u}_j$ is then

$$\nabla_{\mathbf{u}_j} E(i, i+1) =$$
$$= 2 \cot \theta_{i+2} \left(\sum_{k=0}^{2} \mathbf{u}_k^\top (w_k^i - w_k^{i+1}) - \mathrm{R}_t(\mathbf{x}_i - \mathbf{x}_{i+1}) \right) (w_j^i - w_j^{i+1}).$$
$$j = 0, 1, 2 \quad (17)$$

We compute the global gradient by accumulating the micro half-edge terms, and set it to zero by solving the global ARAP step with fixed L matrices.

We show the results of this strategy in Figure 14 and Sec. 10.7.



Directly from input | With preliminary mesh optimization

Fig. 15. An example of base mesh generation for a bas-relief scan without and with a preliminary input optimization step, improving isotropy of the base triangulation at the back of the model.

10 RESULTS AND COMPARISONS

We perform an extensive quantitative and qualitative evaluation of our μ-mesh generation framework. Our reference implementation is publicly released as open-source software at the project page.

Input preparation. Before starting the iterative coarsening, the input mesh triangulation is optimized with local topology transformations [Hoppe et al. 1993] to remove large triangles and slivers, as these can affect the quality of the final base mesh (see Fig. 15). We iterate operations until no edge that is longer than 5 times the average input edge length exists, alternating between splits, flips, and collapses.

10.1 Batch-conversion of high-resolution models

To demonstrate the robustness and feasibility of our method for general conversion of 3D models, we run our algorithms over the ThreeDScans repository [Three D Scans 2022], on commodity hardware, reporting aggregate data about runtime and compression efficiency, as well as average geometric error and isotropy distribution across the entire collection comprising of 121 high-resolution meshes (mean input triangle count is 1.7M). We provide all resulting μ-meshes for inspection, in the additional material, as well as a μ-mesh preview tool.

The results of this experiment are shown in Fig. 19. On average, we achieve a 15:1 compression ratio and the processing time per model is less than five minutes. At the same time, our method consistently produces highly isotropic displaced μ-meshes while keeping the geometric error at reasonable levels.

10.2 μ-Mesh ray-tracing performance

We measure the rendering performance of the batch-converted μ-meshes from Sec. 10.1 using an NVIDIA GeForce RTX 4090, which supports μ-mesh ray-tracing. Compared to an equivalent standard triangle mesh, the μ-meshes produced by our construction method, improve BVH size by a median factor of 6 and BVH build time by a factor of 4, while the tracing of rays is a median of 1.3 times slower. The relative performance may be attributed to the dynamic generation of watertight μ-triangles, which are then subject to the standard ray-triangle intersection computation. Crucially, prismoid optimization produces a 3 to 4-fold speedup in rendering performance by better fitting the bounding prismoids to the displaced μ-mesh surface. We provide a table with aggregated and individual measurements in the additional materials.

10.3 Memory size and accuracy

The experiment shown in Figure 16 evaluates the impact of the resolution of the base mesh on both the GPU occupancy and accuracy of the produced μ-mesh. In this experiment, we force the coarsening of the base mesh to stop only at a $1/n$ fraction of the input mesh triangle count, and in each case, we set the subdivision levels to match the resolution of the input mesh.

GPU size. The μ-mesh is more compact than the input mesh (considering it as an indexed mesh with 32-bit indices and 32-bit floats for coordinates, with no attribute) already for a reduction of $1/4$. The GPU occupancy is reduced for coarser base meshes, but for fewer faces than $1/64$, the improvements become negligible, as the majority of memory is used to store the scalar displacements. At that point, the μ-mesh uses only about 0.14 of the memory required by the explicit indexed mesh representation.

Reproduction accuracy. The reproduction accuracy is always very high. We estimate the geometric error as the area-weighted average of the projection distance from the original surface to the μ-mesh surface, expressed as a percentage of the input AABB diagonal. For higher resolution base meshes, the error is closer to zero, while for a wide range of reduction factor, from $1/8$ to $1/128$ it is measured at roughly 2×10^{-5} the bounding box diagonal, due to the effect of re-sampling the geometry.

10.4 Effect of scalar quantization

A crucial part of the μ-mesh data structure is that scalar displacements are quantized to a small number of bits (see Sec. 1.1). In our outputs, this quantization can be made more aggressive thanks to our displacement minimization (Sec. 8), which also reduces the effect of the aliasing. The impact can be significant, because, on our outputs, the scalar displacement values account for 85% to 90% of the final storage, so reducing the number of bits has almost a linear effect on the total GPU storage size.

The experiment in Figure 17 visually shows the impact of quantization on a μ-mesh obtained with our method. This visual comparison suggests that as low as 7 bits are already sufficient to avoid noticeable quantization artifacts. In all other experiments, however, we conservatively used 11 bits, which is standard for the μ-mesh format and roughly matches the precision that would be afforded by standard 32-bit floating point numbers around the value +1.

10.5 Level of Details

One of the expected advantages of the μ-mesh representation is its multi-resolution capabilities, i.e. the ability to be rendered at a range of different Levels of Detail (without introducing any crack or artifact), simply by dynamically capping the subdivision levels. Figure 18 showcases this on one example produced by our method, confirming the expectations.

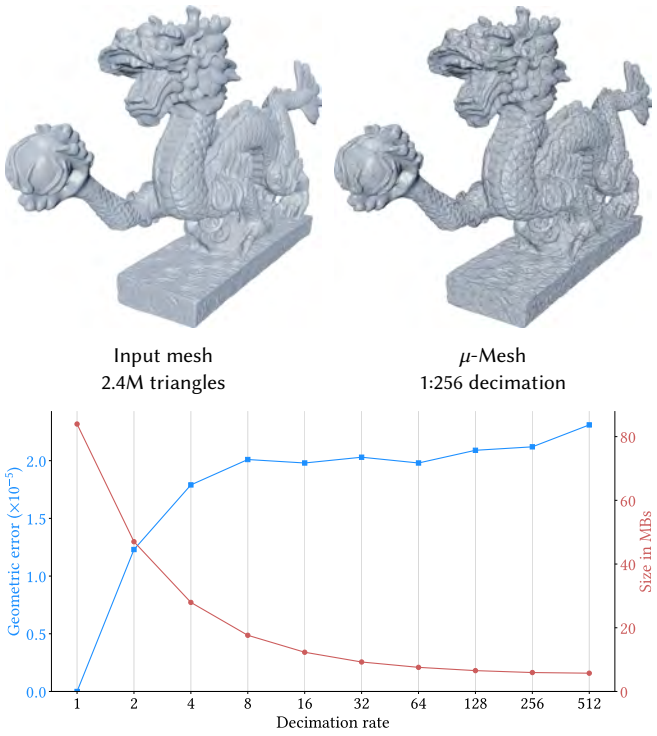


Fig. 16. Used GPU RAM and introduced geometric error for various decimation rates of the base mesh.

Table 1. Conversion of high-resolution 3D models to μ-Meshes. Visual results are shown in Figure 24.

Model	$\|F\|$	$\|F_B\|$	$\|F_\mu\|$	Input MBs	μ-Mesh MBs	Isotropy	Error $\times 10^{-6}$	Time (sec)
Telegraph (A)	8 M	35 K	8.5 M	136.3	7.5	0.69	6.79	436
GW Bust (B)	26 M	30 K	27 M	447.2	20.2	0.73	3.28	1660
Murex (C)	3.5 M	30 K	4 M	60.4	3.9	0.81	10.3	317
Lucy (D)	28 M	35 K	29. 5M	475.3	22.7	0.75	3.38	1661
Statuette (E)	10 M	40 K	10.5 M	171.5	22.7	0.75	13.1	563
Dragon (F)	7 M	30 K	7.5 M	123.9	6.7	0.81	9.16	399
Fangyi (G)	21.5 M	100 K	23 M	370.3	20.5	0.74	6.77	1215
Ewer (H)	8 M	100 K	8 M	136.2	9.8	0.78	13.5	462

10.6 Scalability

We test our method on a collection of high-resolution meshes with face counts in the order of 10^7. Results are shown in Figure 24 reported in Table 1. Our method fares well in terms of the compression ratios, processing times, resulting geometric errors (as a percentage of bounding box diagonal), and triangle shape isotropy, measured as the area-weighted average of the micro triangle aspect ratios.

10.7 Textures and UV-maps

The experiments in Figures 14 and 20 assess the efficacy of our strategy to mitigate the texture distortion induced by the displacement (Sec. 9.2).

For these models, we construct an initial parametrization over the base mesh using Blender automatic parametrization tools [Blender Development Team 2022], followed by an automatic tool to reduce texture cuts [Maggiordomo et al. 2021], and followed by a pass of ARAP texture coordinate optimization [Liu et al. 2008].

Table 2. Displacement-aware parametrization of μ-Meshes. Textured results are shown in Figure 20

Model	Fig.	$\|F_B\|$	$\|F_\mu\|$	ARAP base	μ–ARAP before	μ–ARAP after	Gain	Time
Wooden Dragon	20a	20 K	3.3 M	0.007	0.035	0.021	39.9 %	20 s
Grotesque02	20b	30 K	6.5 M	0.012	0.033	0.025	39.3 %	28 s
Wooden Lion	20c	10 K	1.5 M	0.021	0.039	0.029	25.6 %	8 s
Tokay Gecko	20d	10 K	2.3M	0.035	0.089	0.069	22.4 %	10 s

In Table 2 we report energy measures before and after optimizing base-texture coordinates with our re-weighted formulation, as well as the optimized standard ARAP energy on the base mesh, which is used as initialization for the displacement-aware minimization. The energy decrease is quite noticeable despite significantly restricting the degrees of freedom of the system. The resolution of the base mesh ultimately determines the system variables, and coarser base meshes result in smaller gains in terms of distortion energy (rows 3 and 4 of Table 2). In terms of performance, our method is extremely fast as it operates on a reduced set of variables. In fact, when minimizing the re-weighted ARAP energy with the local-global method the bottleneck is the *local* optimization step, and not from the sparse linear solve required in the global step. The local step requires computing the SVD of a possibly huge number of small 2×2 matrices (one for each micro-triangle), but this process is trivial to parallelize to gain a significant speed-up. As can be seen from the table, the time required to minimize the re-weighted energy is extremely small.

10.8 Comparison with standard mesh simplification

As our μ-mesh generation procedure has no perfect predecessor, we compare it against the use of existing tools to achieve the same effect. The base mesh can be constructed with simplification tools, or with tools designed for general displacement-map generation.

For this comparison, we choose Meshlab [Cignoni et al. 2008] because of its popularity and Simplygon [Microsoft 2022] because it natively supports the generation of displacement-mapped surfaces with its simplification and resampling workflow.

In Figure 21 we show a qualitative comparison of the results obtained using as starting point the base meshes obtained with Meshlab, Simplygon, and our decimation strategies. In all cases, an input mesh, 2.1M faces, is decimated to 25K faces to produce the base mesh. Displacing base meshes obtained with Meshlab and Simplygon produce geometric artifacts and self-intersections, while our base mesh does not suffer from such issues because it ensures that consistently oriented displacement directions can always be defined. Our method outperforms competing methods also in terms of geometric error, μ-triangles isotropy, and processing times (see Figures 22 and 22).

10.9 Comparison with Bijective shells [Jiang et al. 2020]

Base mesh construction. In Figure 23, we compare against Bijective shell [Jiang et al. 2020] as a way to construct the base mesh; this work is akin to our framework in that care is taken to ensure re-projectability of a coarsened mesh into the input mesh. In [Jiang

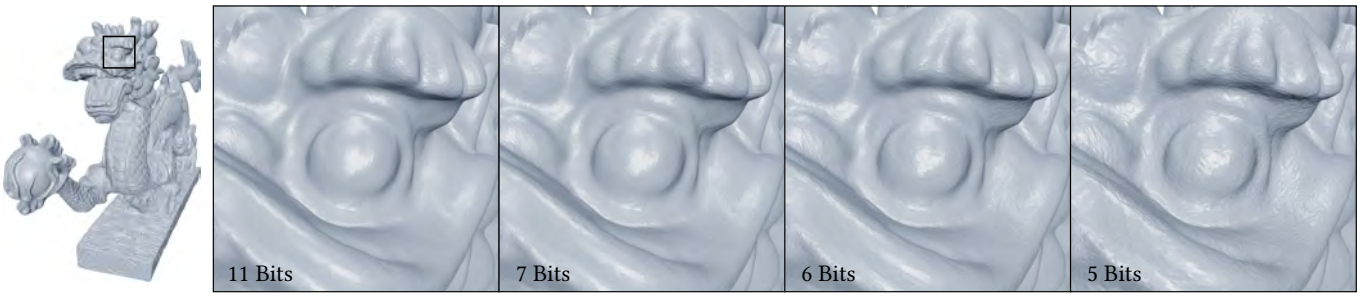


Fig. 17. Effect of quantization of scalar displacements.

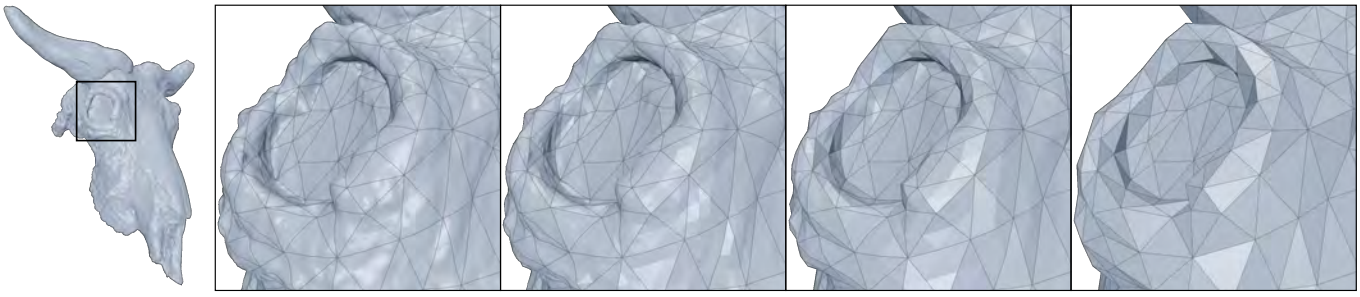


Fig. 18. Dynamically controlling the level of detail of a μ-Mesh by progressively decreasing the subdivision level of base faces.

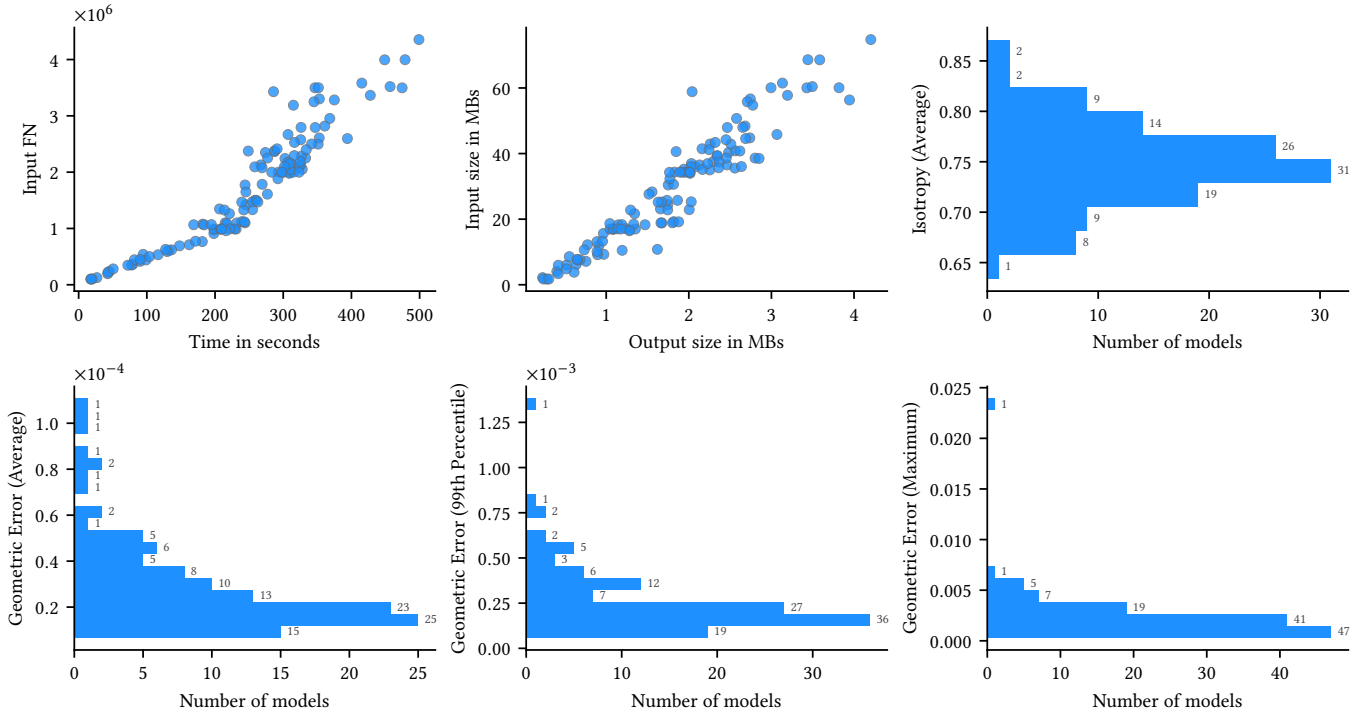


Fig. 19. Batch-processing the ThreeDScans [Three D Scans 2022] mesh repository. For each dataset, we report the end-to-end processing time, data compression, and quality metrics for the displaced μ-mesh: average isotropy, and average, 99th percentile, and maximum geometric error (expressed as a fraction of the bounding box diagonal).

(a) RWT180 – Wooden Dragon

(b) RWT205 – Grotesque02

(c) RWT236 – Wooden Lion

(d) RWT295 – Tokay Gecko

Fig. 20. Examples of produced textured μ-Meshes, from models taken photogrammetry reconstructions dataset [Maggiordomo et al. 2020]. The color maps show the ARAP energy before and after the optimization with our re-weighted scheme (see legend in Fig. 14). Also see Table 2.

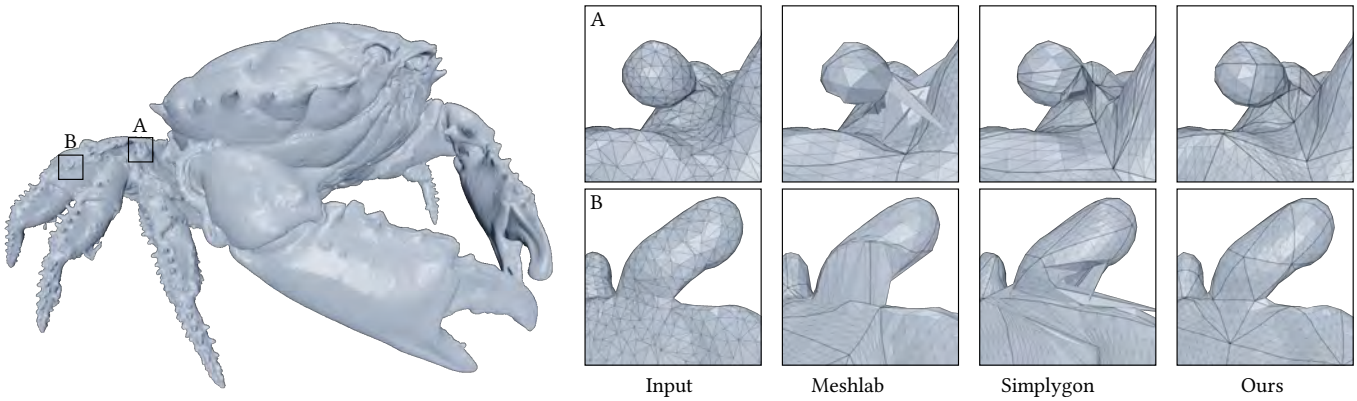


Fig. 21. Comparison with other mesh simplification approaches. The two close-ups show failures due to the presence of base mesh vertices with negative visibility, resulting in displacement directions that are not coherently oriented.

et al. 2020], this is guaranteed by redefining the displacement to occur along a more complex path than a straight line. However, as the case in Figure 23 shows, the existence of this bijective map is not sufficient to ensure correct reproduction when the paths are straight, as with μ-meshes.

Visibility direction computation. We tested our new algorithm to compute optimal visibility (Sec. 4.1) against the QP formulation of (1) proposed in [Jiang et al. 2020], using a state-of-the-art QP solver [Stellato et al. 2020]: our method is more than 26× faster (0.64 against 17.24 seconds to solve 1M instances).

11 CONCLUSIONS

In this paper, we introduced a robust method to automatically convert a high-resolution triangle mesh into a μ-mesh, a recently disclosed representation, designed for direct hardware rendering using commercially available GPUs. Our remeshing algorithms are

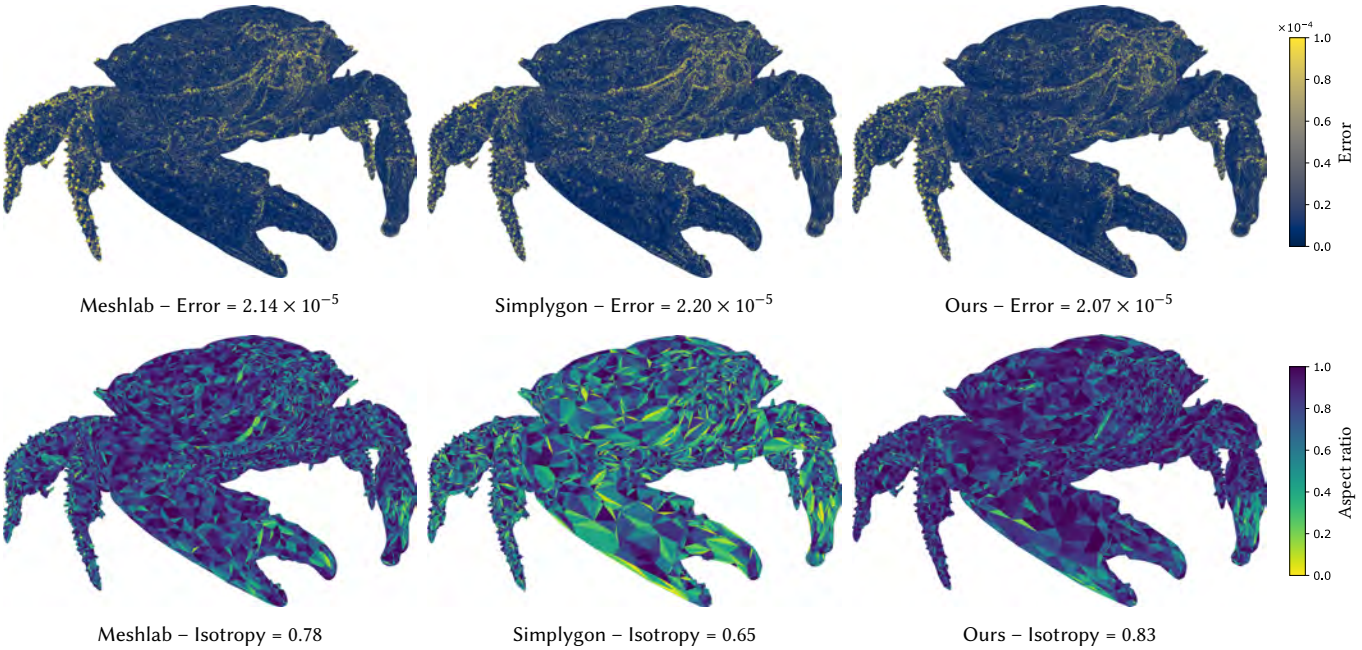


Fig. 22. μ-Mesh geometric error (above) and isotropy (below) for different base mesh generation approaches.

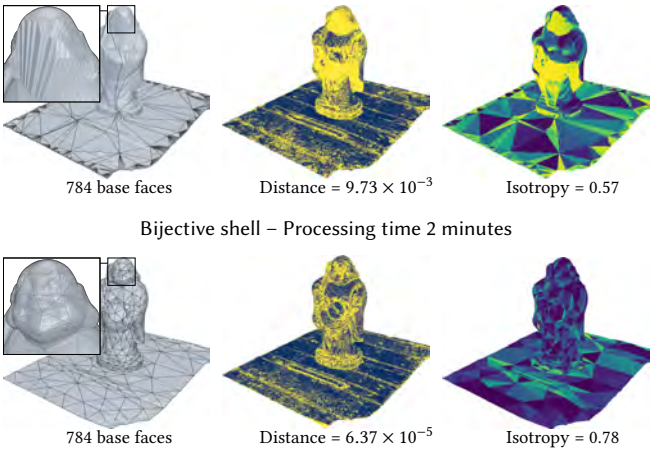


Fig. 23. RWT1 (The Thinker Sculpture by OpenMVS) – Comparison with Bijective shell [Jiang et al. 2020]. The input mesh has 220K faces. This is a failure case for Bijective shells, resulting in a base mesh that is unable to reproduce the original surface by means of scalar displacements.

specifically designed to produce high-quality μ-meshes, delivering accurate, compact models directly consumable by GPUs today. μ-meshes offer significant BVH savings and good tracing performance with nothing blocking increased performance should usage warrant the investment. With this combination, we significantly lower the barrier to rendering extremely detailed geometries and could bring previously unattainable geometric fidelity to the full spectrum of systems.

This work is intended as a stepping stone towards enabling the adoption of extremely high-resolution geometries in interactive and real-time graphics applications, allowing a wide variety of high-quality 3D assets to be readily available by means of a robust and reliable conversion process.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful comments. We are also grateful to Yury Uralsky for extensive discussions and resulting insights, to Christoph Kubisch for his help with performance measurements and to Manuel Kraemer for his feedback and valuable pointers. We acknowledge a gift of support from NVIDIA.

REFERENCES

Adobe. 2021. Bake Mesh Maps. Substance 3D Documentation, https://helpx.adobe.com/substance-3d-painter/using/baking.html.

Pierre Alliez, Éric Colin de Verdière, Olivier Devillers, and Martin Isenburg. 2005. Centroidal Voronoi diagrams for isotropic surface remeshing. *Graphical Models* 67, 3 (2005), 204–231. https://doi.org/10.1016/j.gmod.2004.06.007 Special Issue on SMI 2003.

Romain Aubry and Rainald Löhner. 2008. On the 'most normal' normal. *Communications in Numerical Methods in Engineering* 24, 12 (2008), 1641–1652. https://doi.org/10.1002/cnm.1056

Blender Development Team. 2022. *Blender*. Blender Project. https://blender.org

Mario Botsch and Leif Kobbelt. 2004. A Remeshing Approach to Multiresolution Modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (Nice, France) *(SGP '04)*. Association for Computing Machinery, 185–192. https://doi.org/10.1145/1057432.1057457

Tamy Boubekeur and Christophe Schlick. 2008. A Flexible Kernel for Adaptive Mesh Refinement on GPU. *Computer Graphics Forum* (2008). https://doi.org/10.1111/j.1467-8659.2007.01040.x

Fig. 24. A selection of μ-Meshes generated from very high-resolution inputs. For quantitative measures, see Table 1.

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. https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136

Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. 2003. BDAM: Batched Dynamic Adaptive Meshes for High Performance Terrain Visualization. *Computer Graphics Forum* 22, 3 (sept 2003), 505–514.

Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. 2004. Adaptive Tetrapuzzles: Efficient out-of-Core Construction and Visualization of Gigantic Multiresolution Polygonal Models. *ACM Trans. Graph.* 23, 3 (aug 2004), 796–803. https://doi.org/10.1145/1015706.1015802

Paolo Cignoni, Claudio Montani, Claudio Rocchini, Roberto Scopigno, and Marco Tarini. 1999. Preserving attribute values on simplified meshes by resampling detail textures. *The Visual Computer* 15, 10 (1999), 519–539. https://doi.org/10.1007/s003710050197

Jonathan Cohen, Dinesh Manocha, and Marc Olano. 1997. Simplifying Polygonal Models Using Successive Mappings. In *Proceedings of the 8th Conference on Visualization '97* (Phoenix, Arizona, USA) *(VIS '97)*. IEEE Computer Society Press, 395–ff.

Jonathan Cohen, Marc Olano, and Dinesh Manocha. 1998. Appearance-Preserving Simplification. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, 115–122. https://doi.org/10.1145/280814.280832

Gordon Collins and Adrian Hilton. 2002. Mesh Decimation for Displacement Mapping. In *Eurographics 2002 - Short Presentations*. Eurographics Association. https://doi.org/10.2312/egs.20021033

Robert L. Cook. 1984. Shade Trees. *SIGGRAPH Comput. Graph.* 18, 3 (jan 1984), 223–231. https://doi.org/10.1145/964965.808602

Leila De Floriani, Paola Magillo, and Enrico Puppo. 1998. Efficient implementation of multi-triangulations. In *Proceedings Visualization '98 (Cat. No.98CB36276)*. 43–50. https://doi.org/10.1109/VISUAL.1998.745283

Tamal K Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V Nekhayev. 1999. Topology Preserving Edge Contraction. *Publications de l'Institut Mathématique* 86 (1999), 23–45.

David A. Field. 2000. Qualitative measures for initial meshes. *Internat. J. Numer. Methods Engrg.* 47, 4 (2000), 887–906. https://doi.org/10.1002/(SICI)1097-0207(20000210)47:4<887::AID-NME804>3.0.CO;2-H

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 '97)*. ACM Press/Addison-Wesley Publishing Co., 209–216. https://doi.org/10.1145/258734.258849

Michael Garland and Paul S. Heckbert. 1998. Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings Visualization '98 (Cat. No.98CB36276)*. 263–269. https://doi.org/10.1109/VISUAL.1998.745312

Michael Garland and Yuan Zhou. 2005. Quadric-Based Simplification in Any Dimension. *ACM Trans. Graph.* 24, 2 (apr 2005), 209–239. https://doi.org/10.1145/1061347.1061350

Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. 2000. Normal Meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., 95–102. https://doi.org/10.1145/344779.344831

Autodesk Help. 2021. Transfer Maps. Autodesk Knowledge Network, https://knowledge.autodesk.com/support/maya/learn?s=Transfer+Maps.

Hugues Hoppe. 1996. Progressive Meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, 99–108. https://doi.org/10.1145/237170.237216

Hugues Hoppe. 1999. New quadric metric for simplifying meshes with appearance attributes. In *Proceedings Visualization '99 (Cat. No.99CB37067)*. 59–510. https://doi.org/10.1109/VISUAL.1999.809869

Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1993. Mesh Optimization. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (Anaheim, CA) *(SIGGRAPH '93)*. Association for Computing Machinery, 19–26. https://doi.org/10.1145/166117.166119

Kaimo Hu, Dong-Ming Yan, David Bommes, Pierre Alliez, and Bedrich Benes. 2017. Error-Bounded and Feature Preserving Surface Remeshing with Minimal Angle Improvement. *IEEE Transactions on Visualization and Computer Graphics* 23, 12 (2017), 2560–2573. https://doi.org/10.1109/TVCG.2016.2632720

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant Field-Aligned Meshes. *ACM Trans. Graph.* 34, 6, Article 189 (nov 2015), 15 pages. https://doi.org/10.1145/2816795.2818078

Zhongshi Jiang, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2020. Bijective Projection in a Shell. *ACM Trans. Graph.* 39, 6, Article 247 (nov 2020), 18 pages. https://doi.org/10.1145/3414685.3417769

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, Article 157 (jul 2021), 16 pages. https://doi.org/10.1145/3450626.3459840

Brian Karis, Rune Stubbe, and Graham Wihlidal. 2021. Nanite – A Deep Dive. In *Advances in Real-Time Rendering in Games. ACM SIGGRAPH 2021 Courses)* (Virtual) *(SIGGRAPH '21)*.

Dawar Khan, Alexander Plopski, Yuichiro Fujimoto, Masayuki Kanbara, Gul Jabeen, Yongjie Jessica Zhang, Xiaopeng Zhang, and Hirokazu Kato. 2022. Surface Remeshing: A Systematic Literature Review of Methods and Research Directions. *IEEE Transactions on Visualization and Computer Graphics* 28, 3 (2022), 1680–1713. https://doi.org/10.1109/TVCG.2020.3016645

Deok-Soo Kim, Panos Y. Papalambros, and Tony C. Woo. 1995. Tangent, normal, and visibility cones on Bézier surfaces. *Computer Aided Geometric Design* 12, 3 (1995), 305–320. https://doi.org/10.1016/0167-8396(94)00016-L

Aaron Lee, Henry Moreton, and Hugues Hoppe. 2000. Displaced Subdivision Surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., 85–94. https://doi.org/10.1145/344779.344829

Peter Lindstrom and Greg Turk. 2000. Image-Driven Simplification. *ACM Trans. Graph.* 19, 3 (jul 2000), 204–241. https://doi.org/10.1145/353981.353995

Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J. Gortler. 2008. A Local/Global Approach to Mesh Parameterization. In *Proceedings of the Symposium on Geometry Processing* (Copenhagen, Denmark) *(SGP '08)*. Eurographics Association, 1495–1504.

Yang Liu, Wenping Wang, Bruno Lévy, Feng Sun, Dong-Ming Yan, Lin Lu, and Chenglei Yang. 2009. On Centroidal Voronoi Tessellation—Energy Smoothness and Fast Computation. *ACM Trans. Graph.* 28, 4, Article 101 (sep 2009), 17 pages. https://doi.org/10.1145/1559755.1559758

Haik Lorenz and Jürgen Döllner. 2008. Dynamic mesh refinement on GPU using geometry shaders. (2008).

David P. Luebke. 2001. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications* 21, 3 (2001), 24–35. https://doi.org/10.1109/38.920624

Andrea Maggiordomo, Paolo Cignoni, and Marco Tarini. 2021. Texture Defragmentation for Photo-Reconstructed 3D Models. *Computer Graphics Forum* 40, 2 (2021), 65–78. https://doi.org/10.1111/cgf.142615

Andrea Maggiordomo, Federico Ponchio, Paolo Cignoni, and Marco Tarini. 2020. Real-World Textured Things: A repository of textured models generated with modern photo-reconstruction tools. *Computer Aided Geometric Design* 83 (2020), 101943. https://doi.org/10.1016/j.cagd.2020.101943

Marmoset. 2022. Baking in Toolbag. https://marmoset.co/toolbag/baking/.

Morgan McGuire and Kyle Whitson. 2008. Indirection Mapping for Quasi-Conformal Relief Texturing. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games* (Redwood City, California) *(I3D '08)*. Association for Computing Machinery, 191–198. https://doi.org/10.1145/1342250.1342280

Microsoft. 2022. Simplygon: The Standard in 3D Games Content Optimization. https://www.simplygon.com/.

NVIDIA. 2022. NVIDIA Ada GPU Architecture. https://www.nvidia.com/it-it/geforce/ada-lovelace-architecture. https://images.nvidia.com/aem-dam/Solutions/geforce/ada/ada-lovelace-architecture/nvidia-ada-gpu-architecture-whitepaper-1.03.pdf

Daniele Panozzo, Enrico Puppo, Marco Tarini, Nico Pietroni, and Paolo Cignoni. 2011. Automatic Construction of Quad-Based Subdivision Surfaces Using Fitmaps. *IEEE Transactions on Visualization and Computer Graphics* 17, 10 (2011), 1510–1520. https://doi.org/10.1109/TVCG.2011.28

Enrico Puppo. 1998. Variable resolution triangulations. *Computational Geometry* 11, 3 (1998), 219–238. https://doi.org/10.1016/S0925-7721(98)00029-7

Pedro V. Sander and Jason L. Mitchell. 2005. Progressive Buffers: View-dependent Geometry and Texture for LOD Rendering. In *Eurographics Symposium on Geometry Processing 2005*, Mathieu Desbrun and Helmut Pottmann (Eds.). The Eurographics Association. https://doi.org/10.2312/SGP/SGP05/129-138

Michael Schwarz and Marc Stamminger. 2009. Fast GPU-based Adaptive Tessellation with CUDA. *Computer Graphics Forum* 28, 2 (2009), 365–374. https://doi.org/10.1111/j.1467-8659.2009.01376.x

Alla Sheffer, Emil Praun, and Kenneth Rose. 2006. Mesh Parameterization Methods and Their Applications. *Found. Trends. Comput. Graph. Vis.* 2, 2 (jan 2006), 105–171. https://doi.org/10.1561/0600000011

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. 2020. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation* 12, 4 (2020), 637–672. https://doi.org/10.1007/s12532-020-00179-2

Vitaly Surazhsky, Pierre Alliez, and Craig Gotsman. 2003. *Isotropic Remeshing of Surfaces: a Local Parameterization Approach.* Research Report RR-4967. INRIA.

Marco Tarini, Paolo Cignoni, and Roberto Scopigno. 2003. Visibility based methods and assessment for detail-recovery. In *IEEE Visualization, 2003. VIS 2003.* 457–464. https://doi.org/10.1109/VISUAL.2003.1250407

Three D Scans. 2022. Three D Scans - Free 3D scan archive. https://threedscans.com/

Emo Welzl. 1991. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, Hermann Maurer (Ed.). Springer Berlin Heidelberg, 359–370.

Dong-Ming Yan, Bruno Lévy, Yang Liu, Feng Sun, and Wenping Wang. 2009. Isotropic Remeshing with Fast and Exact Computation of Restricted Voronoi Diagram. *Computer Graphics Forum* 28, 5 (2009), 1445–1454. https://doi.org/10.1111/j.1467-8659.2009.01521.x

Sung-Eui Yoon, B. Salomon, R. Gayle, and D. Manocha. 2004. Quick-VDR: interactive view-dependent rendering of massive models. In *IEEE Visualization 2004.* 131–138. https://doi.org/10.1109/VISUAL.2004.86

Cem Yuksel, John Keyser, and Donald H. House. 2010. Mesh colors. *ACM Transactions on Graphics* 29, 2, Article 15 (2010), 11 pages. https://doi.org/10.1145/1731047.1731053

Cem Yuksel, Sylvain Lefebvre, and Marco Tarini. 2019. Rethinking Texture Mapping. *Computer Graphics Forum* 38, 2 (2019), 535–551. https://doi.org/10.1111/cgf.13656 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13656

Tobias Zirr and Tobias Ritschel. 2019. Distortion-Free Displacement Mapping. *Computer Graphics Forum* 38, 8 (2019), 53–62. https://doi.org/10.1111/cgf.13760 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13760