The Visual Computer (2022) 38:3419–3433
https://doi.org/10.1007/s00371-022-02547-6

ORIGINAL ARTICLE

Fast harmonic tetrahedral mesh optimization

D. Ströter[1] · J. S. Mueller-Roemer[2] · D. Weber[2] · D. W. Fellner[3]

Accepted: 21 May 2022 / Published online: 20 June 2022
© The Author(s) 2022

Abstract
Mesh optimization is essential to enable sufficient element quality for numerical methods such as the finite element method (FEM). Depending on the required accuracy and geometric detail, a mesh with many elements is necessary to resolve small-scale details. Sequential optimization of large meshes often imposes long run times. This is especially an issue for Delaunay-based methods. Recently, the notion of harmonic triangulations [1] was evaluated for tetrahedral meshes, revealing significantly faster run times than competing Delaunay-based methods. A crucial aspect for efficiency and high element quality is boundary treatment. We investigate directional derivatives for boundary treatment and massively parallel GPUs for mesh optimization. Parallel flipping achieves compelling speedups by up to 318×. We accelerate harmonic mesh optimization by 119× for boundary preservation and 78× for moving every boundary vertex, while producing superior mesh quality.

Keywords Numerical optimization · GPGPU · Simplicial meshes · Simulation

1 Introduction

Meshing a domain Ω into a set of simplices T is a fundamental task in geometry processing. The resulting mesh can be used to solve differential equations, enabling a wide range of applications including physically based animation using the FEM [19,34] and spectral geometry processing [17,39].

Mesh generation for numerical computation is not only concerned with finding a triangulation T of Ω. Elements of small volume or area, i.e., ill-shape, must be avoided too. In fact, a single ill-shaped element may cause numerical methods to fail [27]. For this reason, current meshing tools [13,30] perform an optimization step after generating an initial mesh. However, it is an open issue for tetrahedral meshes that quality functions are not consistent with the Delaunay triangulation, leading to ill-shaped elements [18]. Recently, Alexa [1] introduced harmonic triangulations, defining an energy whose minimization significantly improves element quality of Delaunay meshes.

The accuracy of numerical methods improves with the mesh resolution. Thus, meshes with many elements are required for the analysis of complex geometric structures. Sequential mesh optimization on the CPU results in slow run times for large meshes, which is especially an issue in interactive settings [33]. Parallel algorithms that use modern parallel processors, specifically massively parallel GPUs, are necessary to optimize large meshes quickly. As harmonic triangulations outperform established Delaunay-based optimization methods, we use them as a basis to devise a parallel mesh optimization algorithm.

In this paper, we extend harmonic mesh optimization to faster run times, improved convergence and boundary treatment. Our contributions are:

- A novel mesh optimization scheme that efficiently improves high-resolution tetrahedral meshes.
- A robustly converging mesh optimization scheme.
- Novel massively parallel algorithms for mesh optimization.
- Gradient-based boundary vertex optimization , replacing reprojection.

✉ D. Ströter
daniel.stroeter@gris.tu-darmstadt.de

1 TU Darmstadt, 64277 Darmstadt, Germany

2 Fraunhofer IGD and TU Darmstadt, Darmstadt, Germany

3 Fraunhofer IGD, TU Darmstadt and TU Graz, Darmstadt, Germany

2 Preliminaries and notation

Although we focus on tetrahedral meshes, our notation covers arbitrary dimensions, because we extend the harmonic triangulations framework. We define a d-dimensional simplicial mesh $\mathscr{M} = (\mathrm{T}, \mathrm{V})$ as a tuple of a d-simplex sequence T and a sequence of vertices $\mathrm{V} \subset \mathbb{R}^d$. Boundary vertices are included in $\partial\mathrm{V}$, where $\partial\mathrm{V} \subseteq \mathrm{V}$. We denote a k-simplex τ as a $(k+1)$-tuple $(\mathbf{x}_0, \ldots, \mathbf{x}_k) \in \mathrm{V}^{k+1}$ of vertices, where $k \leq d$. Oriented volumes, face areas, and normals are represented by v_τ, a_τ, and $\mathbf{n}_\tau$, respectively. The ith vertex of τ is given by $\tau_i \in \mathrm{V}$. Likewise, the ith element in T or V is denoted as T_i or V_i, respectively. The matrix of a d-simplex' vertex set can be written as $\mathbf{X}_\tau = (\tau_0, \ldots, \tau_d) \in \mathbb{R}^{d\times(d+1)}$, where the ith column is the position of τ_i. We use the matrix $\mathbf{M}_d$ to express the vertices of a d-simplex in relation to its first vertex such that $v_\tau = \det(\tau_1 - \tau_0, \ldots, \tau_d - \tau_0)/d! = \det(\mathbf{X}_\tau \mathbf{M}_d)/d!$:

$$\mathbf{M}_d = \begin{pmatrix} -1 & -1 & \ldots & -1 \\ \mathbf{e}_1 & \mathbf{e}_2 & \ldots & \mathbf{e}_d \end{pmatrix} \in \mathbb{R}^{(d+1)\times d}, \quad (1)$$

where $\mathbf{e}_i$ denotes the ith canonical unit vector of $\mathbb{R}^d$.

As the subtriangulation forming the one-ring neighborhood of a specific vertex $\mathbf{x} \in \mathrm{V}$ is of interest during optimization, we introduce the following notation:

$$\mathtt{t}(\mathbf{x}) = \{\tau \in \mathrm{T} \mid \mathbf{x} \in \tau\}. \quad (2)$$

For any k-simplex τ we obtain the $(k-1)$-sub-simplex opposite to τ_i with the set difference $\tau \setminus \tau_i$. The goal of harmonic mesh optimization [1] is to minimize the trace of the Laplacian $\mathbf{L}_\mathrm{T}$ consisting of $\mathrm{tr}(\mathbf{L}_\tau)$, where $\tau \in \mathrm{T}$:

$$\mathrm{tr}(\mathbf{L}_\tau) = \frac{1}{d^2} \frac{\sum_{i=0}^{d} a^2_{\tau\setminus\tau_i}}{|v_\tau|} \quad (3)$$

$$\mathrm{tr}(\mathbf{L}_\mathrm{T}) = \sum_{\tau\in\mathrm{T}} \mathrm{tr}(\mathbf{L}_\tau). \quad (4)$$

Dropping the constant factor leads to the harmonic index η:

$$\eta(\tau) = \frac{\sum_{i=0}^{d} a^2_{\tau\setminus\tau_i}}{|v_\tau|}, \text{ where } \tau \in \mathrm{T}. \quad (5)$$

The optimized triangulation shall respect the input boundary $\mathscr{B}$ and be free of inversions to satisfy the requirements of numerical methods. Considering these conditions leads to the following nonlinear optimization problem:

$$\begin{aligned} &\underset{\mathscr{M}=(\mathrm{T},\mathrm{V})}{\text{minimize}}\ \mathrm{tr}(\mathbf{L}_\mathrm{T}) \\ &\text{subject to } \partial\mathrm{T} \cong \mathscr{B} \wedge \forall\tau \in \mathrm{T}, v_\tau > 0, \end{aligned} \quad (6)$$

where $\cong$ denotes an approximate congruence between the target boundary $\mathscr{B}$ and the discrete boundary $\partial\mathrm{T}$. For full boundary preservation, we can enforce $\partial\mathrm{T} = \mathscr{B}$. Additionally, we prohibit inversions, i.e., tetrahedra τ with $v_\tau < 0$. We denote boundary vertices surrounded by co-planar faces as $\mathrm{V}_\mathscr{F} \subseteq \partial\mathrm{V}$, boundary vertices on a geometrical edge as $\mathrm{V}_\mathscr{E} \subseteq \partial\mathrm{V}$, and boundary vertices representing a geometrical corner as $\mathrm{V}_\mathscr{C} \subseteq \partial\mathrm{V}$.

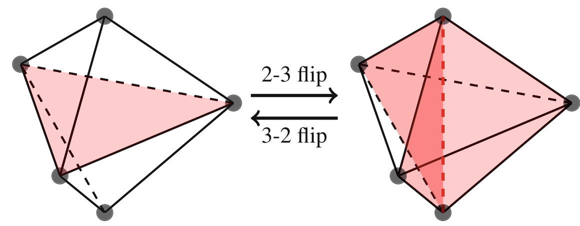

Fig. 1 By using a bistellar 2-3 or 3-2 flip, we retriangulate a convex region defined by five points (interior faces and edges are shaded red)

To perform harmonic mesh optimization, Alexa [1] combines a flipping algorithm and a gradient descent scheme. The flipping algorithm performs 2-3 and 3-2 bistellar flips (see Fig. 1). If a bistellar flip reduces $\mathrm{tr}(\mathbf{L}_\mathrm{T})$, it is a harmonic flip. Harmonic flips can be locally ordered by their reduction of the trace. Prioritizing harmonic flips with the largest reduction of the traces produces good element quality. Thus, harmonic flips may be arranged in an ordered queue favoring flips by their reduction of $\mathrm{tr}(\mathbf{L}_\mathrm{T})$. Additionally, a harmonic flip either coincides with a Delaunay flip or it produces a local triangulation of two tetrahedra, while the Delaunay flip would produce three tetrahedra.

In order to minimize $\mathrm{tr}(\mathbf{L}_\mathrm{T})$, a gradient descent scheme can be used to relocate vertices. The first step is to assemble a gradient for each vertex of the mesh by calculating a gradient for each tetrahedron $\tau \in \mathrm{T}$:

$$\frac{\partial \mathrm{tr}(\mathbf{L}_\tau)}{\partial \mathbf{X}_\tau} = \frac{1}{d!} (\mathbf{X}_\tau \mathbf{M}_d)^{-\top} \mathbf{M}_d^\top (\mathrm{tr}(\mathbf{L}_\tau)\mathbf{I} - 2\mathbf{L}_\tau) \quad (7)$$

To avoid inverted elements, Alexa uses binary search to find a single step size λ for one gradient descent step on the entire mesh. Instead of using λ for vertex relocation, Alexa uses Brent's method [23] to find a minimum along the steepest descent located at some $\alpha \in [0, \lambda]$. Boundary vertices are reprojected onto the surface after global gradient descent.

3 Related work

The literature comprises a long history in investigating tetrahedral mesh optimization. Freitag et al. [9] improve tetrahedral meshes by swapping common faces or edges and relocating vertices. For fast run times, Freitag et al. [8] relocate batches of non-adjacent vertices in parallel, while preventing element inversions. Many mesh optimization frameworks use computational efficient Laplacian smooth-

ing, relocating a vertex in the direction of the arithmetic average of the adjacent vertices [9,26,29,36]. However, Laplacian smoothing does not strictly guarantee to produce a high-quality or even inversion-free mesh. In addition, the gradient of the Laplacian does not vanish in general, which complicates finding appropriate termination criteria. Consequently, previous works devised different quality functions for a mesh element [27]. Knupp [16] confirms the use of the Jacobian as a building block for quality functions of a finite element. Today, many mesh optimization methods rely on distortion methods using the Jacobian. We provide a review of distortion-based methods in Sect. 3.1. As our work contributes massively parallel algorithms for mesh optimization, we discuss related work in this field in Sect. 3.2. We also discuss boundary treatment in our work and highlight previous work in Sect. 3.3.

3.1 Distortion energies for mesh optimization

Besides mesh improvement, energies minimizing global distortion are typically used for parameterization tasks such as surface fitting or remeshing. Hormann et al. [12] introduce the most isometric parameterizations (MIPS). Originally, MIPS is intended for mapping a triangulation of data points to a triangulation in the plane. Fu et al. [10] extend MIPS to the advanced MIPS (AMIPS) energy that effectively minimizes distortion in 2D and 3D. For vertex relocation, they perform nonlinear Gauss–Seidel iterations simultaneously on sets of non-adjacent vertices. However, nonlinear optimization methods typically impose slow run times and do not scale well to meshes with many elements. For this reason, Rabinovich et al. [24] present a local/global algorithm that scales to large data sets through replacing the nonlinear energy with a simple proxy energy. The local step calculates weights mapping gradients to the distortion of elements using the proxy energy. With the weighted gradients, a global system can be efficiently assembled and solved. For solving the global system, an initial inversion-free step size is found using the method of Smith et al. [31].

While distortion energies are effective in improving ill-shaped elements, harmonic triangulations provide a local order of bistellar flips [1]. As flips are locally ordered by energy reduction, we formulate a massively parallel algorithm performing locally most beneficial flips that quickly improves element quality. Additionally, our work focuses on Delaunay-based methods, as harmonic flips are related to Delaunay flips. We achieve scalability by neat parallelization.

3.2 Parallel tetrahedral mesh optimization

Lots of recent work address parallel tetrahedral mesh optimization. Benitez et al. [3] perform smoothing and untangling in a distributed environment using domain decomposition. Shontz et al. [28] relocate vertices by solving ordinary differential equations on a distributed system using domain decomposition. Zint et al. [40] describe a GPU-parallel method to search for an optimal vertex position on a coarse grid of candidate positions. While this enables optimization of non-differentiable functions, we focus on differentiable energies, as they enable first-order methods that converge more quickly than exhaustive search. In addition, we focus on fine-grained parallelism that leads to fast run times on a single machine and does not require a distributed system.

In contrast to parallel vertex relocation, parallel local reconnection of vertices imposes the additional challenge of preventing concurrent processing of overlapping regions. Nonetheless, vertex relocation and reconnection should be used in concert [15] to achieve an effective optimization. D'Amato et al. [5] designed a CPU-GPU framework that performs local remeshing and vertex relocation in parallel using a decomposition of the mesh into clusters. Shang et al. [26] present a multi-threaded algorithm for parallel local reconnection, which maps reconnection operations to feature points sorted along a space filling curve. They assume geometrical separation of remeshing operations so that regions rarely overlap. Ibanez et al. [14] schedule the application of cavity-based remeshing on shared memory systems. Their method finds independent sets of cavities for processing in batches of these independent sets. Drakopoulos et al. [7] describe a parallel speculative local remeshing approach for high-performance computing. They use atomic operations for synchronization in case of overlapping regions. In contrast to established parallel local reconnection methods, our parallel flipping algorithm does not require a precomputed decomposition of the mesh or atomic operations but relies on the local order of harmonic flips.

3.3 Boundary treatment in tetrahedral mesh optimization

Boundary treatment in tetrahedral mesh optimization is a sparsely discussed field. While some methods rely on curved boundaries [6], we only rely on the boundary of the discrete mesh. Many methods either subdivide ill-shaped boundary elements [15] or reproject boundary vertices back on the original surface [1]. Subdivision of boundary elements increases the element count, which is a drawback, as each element costs computationally. The drawbacks of boundary reprojection are that it requires to find the closest point on the boundary and the reprojection step does not respect energy minimization leading to reduced convergence.

Yin et al. [38] replace reprojection of boundary vertices with shape functions approximating the surface based on the discrete mesh. They incorporate the shape functions as a penalty term into the to-be-optimized function to enforce

boundary conformance. Contrary to our method, the penalization approach requires the choice of a suitable penalty number. Wicke et al. [35] address optimization of the mesh boundary for dynamic domain remeshing. They penalize relocation of boundary vertices by augmenting the optimization function with a quadric error term. Although this allows for efficient relocation of boundary vertices, element quality to surface distance is an apples-to-oranges comparison. Xu et al. [37] propose harmonic guided optimization to further improve the quality of boundary elements despite the usage of a quadric error term. They precompute a harmonic scalar field on a voxelized grid. As the field is maximal at the boundary and minimal for the medial axis of the mesh, it enables the computation of weights tweaking the importance of boundary preservation and element quality. Our method keeps boundary vertices on the surface without using a penalization term and thereby without the need of precomputing additional weights.

4 Optimization algorithms

In this section, we describe a harmonic mesh optimization algorithm suitable for parallelization on massively parallel GPUs. In order to compute subsimplex-to-simplex relationships or simplex-to-subsimplex relationships, we employ the mesh data structure developed by Mueller-Roemer et al. [20,21], because it provides memory-efficient organization of these relationships in a compactly encoded ternary sparse row format. The following algorithms assume an inversion-free mesh, i.e., every oriented volume v_τ must be positive.

4.1 Vertex relocation

We focus on gradient descent of interior vertices first and detail the treatment of boundary vertices in Sect. 4.2. We achieve conflict-free parallelization by coloring vertices into independent sets $\mathscr{S}_C$. Therefore, Algorithm 1, which outlines our vertex relocation scheme, can process vertices in Gauss–Seidel iteration order.

A drawback of Alexa's gradient descent scheme [1] is that a single line search is performed for all vertices of the mesh. Thus, vertices potentially affect each other leading to a small set of vertices preventing substantial optimization of the majority of vertices. Additionally, the gradient directions of vertices might lead to conflicting updates, reducing the convergence rate. Instead of performing a single line search for the entire mesh, we perform local gradient descent.

Since each pass over an independent set of vertices affects mesh quality, it is beneficial for the optimization to recalculate the gradients for each batch of independent sets. As the gradient of each vertex depends on multiple tetrahedra, parallel gradient assembly using Eq. (7) requires synchronization primitives, such as atomic operations, in order to handle write conflicts. Thus, we propose calculating the harmonic gradient for each vertex using the following equation instead of Eq. (7), which facilitates parallel processing in independent sets:

$$\frac{\partial \mathrm{tr}(\mathbf{L}_\tau)}{\partial \mathbf{x}} = \frac{a_{\tau\setminus\mathbf{x}}}{d!d^3 v_\tau}\left(\left(\frac{2a_{\tau\setminus\mathbf{x}}^2}{v_\tau} - \eta(\tau)\right)\mathbf{n}_{\tau\setminus\mathbf{x}} + 2\sum_{\mathbf{v}\in(\tau\setminus\mathbf{x})} a_{\tau\setminus\mathbf{v}}^2 \frac{\mathbf{n}_{\tau\setminus\mathbf{x}}^\top \mathbf{n}_{\tau\setminus\mathbf{v}}}{v_\tau}\mathbf{n}_{\tau\setminus\mathbf{v}}\right) \tag{8}$$

The proof of Eq. (8) can be found in the appendix. It is an interesting observation that the harmonic gradient is a linear combination of the face normals of τ. We leave the geometric interpretation of Eq. (8) for future work.

With one pass over $\mathtt{t}(\mathbf{x})$, the gradient of the incident tetrahedra can be calculated by application of the sum rule. For convenience, we introduce a notation for the gradient of tetrahedra incident to $\mathbf{x}$:

$$\nabla_{\mathbf{x}} \mathrm{tr}(\mathbf{L}_{\mathtt{t}(\mathbf{x})}) = \sum_{\tau\in\mathtt{t}(\mathbf{x})} \frac{\partial \mathrm{tr}(\mathbf{L}_\tau)}{\partial \mathbf{x}} \tag{9}$$

Unlike Alexa [1], we locally compute an inversion-free interval $[0, \lambda_{\mathbf{x}}]$ for vertex $\mathbf{x}$ given its local gradient. As an inversion occurs when vertex $\mathbf{x}$ passes the plane spanned by the opposing triangle $\tau \setminus \mathbf{x}$, the exact step size $\lambda_{\mathbf{x}}$ can be determined by performing a simple plane-ray intersection test. An illustration of this principle appears in Fig. 2. Starting from $\lambda_{\mathbf{x}}$, an inversion free step size can be found with binary search using the root finding method of Smith et al. [31]. To avoid unnecessary search iterations, we reduce $\lambda_{\mathbf{x}}$ by a factor $\mu \in [0, 1)$ beforehand. We choose $\mu = .95$ in our work. We set $\lambda_{\mathbf{x}}$ to the resulting step size. As a result, the local gradient descent update formula is as follows:

$$\mathbf{x} \leftarrow \mathbf{x} - \alpha\nabla_{\mathbf{x}} \mathrm{tr}(\mathbf{L}_{\mathtt{t}(\mathbf{x})}), \text{ where } \alpha \in [0, \lambda_{\mathbf{x}}],\ \mathbf{x} \in \mathrm{V}. \tag{10}$$

A bracketing scheme is used to determine α. Like Alexa, we use Brent's [23] method in our work; however, we use it locally.

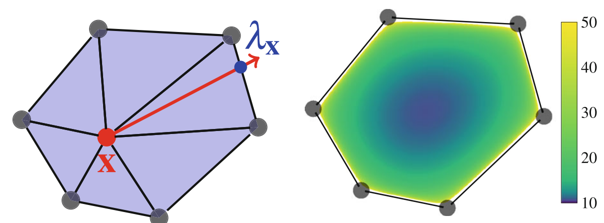

Fig. 2 Calculating the inversion-free interval $[0, \lambda_{\mathbf{x}}]$ for the red vertex $\mathbf{x}$ in the triangulation on the left can be achieved by finding the closest intersection point of the gradient ray with planes defined by opposing faces. The field of $\mathrm{tr}(\mathbf{L}_{\mathtt{t}(\mathbf{x})})$ is shown on the right

Algorithm 1 Parallel vertex relocation algorithm.

1: **procedure** RELOCATEVERTICES($\mathscr{M} = (\mathrm{T}, \mathrm{V}), \mathscr{S}_C$)
2:   **for all** $c \in \mathscr{S}_C$ **do** ▷ Go through independent sets
3:     **for all** $i \in c$ **do** ▷ In parallel
4:       $\mathbf{x} \leftarrow \mathrm{V}_i$; $\lambda \leftarrow \infty$
5:       **if** subject to $\partial\mathrm{T} = \mathscr{B} \wedge \mathbf{x} \in \mathrm{V}_{\mathscr{C}}$ **then**
6:         **continue**
7:       **end if**
8:       $\mathbf{g} \leftarrow \nabla_{\mathbf{x}} \mathrm{tr}(\mathbf{L}_{\mathrm{t}(\mathbf{x})})$ ▷ See Eq.(9)
9:       **if** subject to $\partial\mathrm{T} = \mathscr{B} \wedge (\mathbf{x} \in \mathrm{V}_{\mathscr{F}} \vee \mathbf{x} \in \mathrm{V}_{\mathscr{E}})$ **then**
10:         $\mathbf{g} \leftarrow$ TANGENTSUBSPACEDERIVATIVE($\mathbf{x}, \mathbf{g}$)
11:       **else if** subject to $\partial\mathrm{T} \cong \mathscr{B} \wedge \mathbf{x} \in \partial\mathrm{V}$ **then**
12:         $\mathbf{g} \leftarrow$ FINDTANGENTDERIVATIVE($\mathbf{x}, \lambda, \mathbf{g}$)
13:       **end if**
14:       **if** $\|\mathbf{g}\| < \varepsilon_g$ **then continue end if**
15:       $\mathbf{g} \leftarrow -\frac{\mathbf{g}}{\|\mathbf{g}\|_2}$
16:       $t_0 \leftarrow \min_{\tau \in \mathrm{t}(\mathbf{x})} \{t \mid t = \text{intersect-ray-plane}(\mathbf{g}, \mathbf{x}, \tau \setminus \mathbf{x})\}$
17:       $\lambda \leftarrow \min(t_0, \lambda)$
18:       $\lambda \leftarrow \lambda \cdot \mu$ ▷ Prevent division by zero at upper bound
19:       search $\leftarrow$ *true*
20:       **do**
21:         $\mathrm{V}_i \leftarrow \mathbf{x} + \lambda\mathbf{g}$
22:         search$\leftarrow$ **is** $v_\tau < \varepsilon_v$ **for any** $\tau \in \mathrm{t}(\mathbf{x})$
23:         **if** subject to $\partial\mathrm{T} \cong \mathscr{B}$ **then**
24:           search$\leftarrow$ **is** $\mathbf{x}$ still on boundary primitive?
25:         **end if**
26:         **if** search **then** $\lambda \leftarrow \lambda/2$ **endif**
27:       **while** search
28:       $\mathrm{V}_i \leftarrow \mathbf{x}$
29:       $\alpha \leftarrow$ *bracketing*($\mathbf{x}, \mathbf{g}, [0, \lambda], \mathrm{tr}(\mathbf{L}_{\mathrm{t}(\mathbf{x})})$) ▷ We use [23]
30:       $\mathrm{V}_i \leftarrow \mathbf{x} + \alpha\mathbf{g}$
31:       **if** subject to $\partial\mathrm{T} \cong \mathscr{B}$ **then**
32:         IDENTIFYBOUNDARYSTATE($\mathbf{x}$)
33:       **end if**
34:     **end for**
35:   **end for**
36: **end procedure**

4.2 Directional derivatives for boundary treatment

Unlike relocation of interior vertices, gradient descent of boundary vertices can deform the surface resulting in a significant loss of geometric detail. We intend to avoid reprojection of vertices onto the boundary in our work, while still keeping boundary vertices on the boundary. For this purpose, we investigate directional derivatives for mesh optimization. We first address full preservation of the mesh surface and detail an algorithm for relocating every boundary vertex in Sect. 4.3. If the primary concern is to fully preserve the input surface, we only allow gradients to be co-planar to the boundary surface. We classify boundary vertices depending on their adjacent surface triangles to obtain rules for full surface preservation, which we summarize in Algorithm 2. For full boundary preservation, the following rules apply:

- $\mathbf{x} \in \mathrm{V}_{\mathscr{F}}$: All incident boundary triangles are co-planar. Thus, there is a unique tangent plane.
- $\mathbf{x} \in \mathrm{V}_{\mathscr{E}}$: Two sets of incident boundary triangles are co-planar. Thus, there is a unique tangent line.
- $\mathbf{x} \in \mathrm{V}_{\mathscr{C}}$: There is no unique tangent plane or line. A corner vertex cannot be moved without altering the surface.

For full boundary preservation, we apply homogeneous Neumann boundary conditions [2], while alternative boundary conditions are an ongoing research topic [32]:

$$\mathbf{n}_{\mathbf{x}}^{\top} \nabla_{\mathbf{x}} \mathrm{tr}(\mathbf{L}_{\mathrm{t}(\mathbf{x})}) = 0, \;\text{ where } \mathbf{x} \in \partial\mathrm{V}.$$

Thus, a boundary vertex is only relocated along a tangent plane or line. Let p be a tangent plane on the surface with linearly independent unit vectors $\mathbf{u}_1$ and $\mathbf{u}_2$:

$$p(t,s) = \mathbf{x} + t\,\mathbf{u}_1 + s\,\mathbf{u}_2, \;\text{ where } \mathbf{n}_{\mathbf{x}}^{\top}\mathbf{u}_1 = 0, \; \mathbf{n}_{\mathbf{x}}^{\top}\mathbf{u}_2 = 0, \; t, s \in \mathbb{R}.$$

We now show, how we apply directional derivatives to the surface of a tetrahedral mesh. Let the function $g_{\mathbf{x}}$ replace a boundary vertex $\mathbf{x}$ with a given vertex $\mathbf{x}'$ and calculate the trace for all incident tetrahedra:

$$g_{\mathbf{x}}(\mathbf{x}') = \sum_{\tau \in \mathrm{t}(\mathbf{x})} \mathrm{tr}(\mathbf{L}_{(\tau \setminus \mathbf{x}) \cup \mathbf{x}'}). \tag{11}$$

With the use of $g_{\mathbf{x}}$ and p we can express the field of $\mathrm{tr}(\mathbf{L}_{\mathrm{t}(\mathbf{x})})$ on the tangent plane as:

$$g_{\mathbf{x}}(p(t,s)), \;\text{ where } t, s \in \mathbb{R}. \tag{12}$$

As our goal is to obtain a gradient for $\mathbf{x}$ on the tangent plane p, we evaluate the gradient at $t = 0$ and $s = 0$. The gradient follows by the chain rule:

$$\begin{aligned} \nabla g_{\mathbf{x}}(p(0,0)) &= \nabla_{p(0,0)} g_{\mathbf{x}}(p(0,0)) \nabla p(0,0) \\ &= \nabla_{\mathbf{x}} g_{\mathbf{x}}(\mathbf{x}) \nabla p(0,0) \end{aligned} \tag{13}$$

The gradient of the tangent plane p evaluates to $(\mathbf{u}_1, \mathbf{u}_2)^{\top}$. Because $g_{\mathbf{x}}(\mathbf{x})$ replaces $\mathbf{x}$ with itself, we can further simplify the gradient to:

$$\nabla g_{\mathbf{x}}(p(0,0)) = \nabla_{\mathbf{x}} \mathrm{tr}(\mathbf{L}_{\mathrm{t}(\mathbf{x})}) \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \begin{pmatrix} t_0 \\ s_0 \end{pmatrix} \in \mathbb{R}^2. \tag{14}$$

The gradient on the plane can be transformed to $\mathbb{R}^3$, resulting in the directional derivative:

$$\mathbb{R}^3 \ni \nabla_{\mathbf{x}}|_p \; \mathrm{tr}(\mathbf{L}_{\mathrm{t}(\mathbf{x})}) = t_0\,\mathbf{u}_1 + s_0\,\mathbf{u}_2 \tag{15}$$

In case of a tangent line for $\mathbf{x} \in \mathrm{V}_{\mathscr{E}}$, one can just drop $\mathbf{u}_2$ and perform the analog calculations. The use of directional derivatives provides several benefits for mesh optimization:

Algorithm 2 Derivative on a tangent sub-space.

```
1: procedure TANGENTSUBSPACEDERIVATIVE(x, g)
2:   if x ∈ V_F then
3:     τ_b ← get_boundary_triangle(x)
4:     (u_1, u_2) ← tangent_plane(τ_b)
5:     return (u_1^⊤ g)u_1 + (u_2^⊤ g)u_2
6:   else                                   ▷ Otherwise x ∈ V_E
7:     e_b ← get_boundary_edge(x)
8:     u_1 ← tangent_line(e_b)
9:     return (u_1^⊤ g)u_1
10:  end if
11: end procedure
```

1. Reprojection of vertices after gradient descent is not necessary. Thus, it becomes obsolete to find the closest surface triangle, which can be computationally expensive.
2. Line search on a tangent subspace converges against a local minimum. No special convergence criteria are necessary for the boundary.
3. Projection of relocated vertices to the closest surface triangle can produce inversions or projection of vertices onto opposing faces. We avoid these issues by inversion free intervals for line searches along the boundary.

4.3 Moving every boundary vertex

We present an algorithm that relies on directional derivatives to optimize boundary vertices while keeping them on the mesh surface. As a result, the approximation error due to boundary vertex relocation is controlled by the input mesh surface, which is assumed to be of high resolution such that the approximation error for curved surfaces is low enough. The main idea of our algorithm is to relax the homogeneous Neumann boundary condition such that the gradient has to lie only on a single tangent plane (or line) p. At the same time, we relocate boundary vertices only along the boundary:

$$\mathbf{n}_p^\top \nabla_{\mathbf{x}}|_p \operatorname{tr}(\mathbf{L}_{\mathrm{t}(\mathbf{x})}) = 0, \text{ where } \mathbf{x} \in \partial \mathrm{V}$$
$$\mathbf{x} - \alpha \nabla_{\mathbf{x}}|_p \operatorname{tr}(\mathbf{L}_{\mathrm{t}(\mathbf{x})}) \text{ is on } \mathscr{B}, \text{ where } \alpha \in [0, \lambda_{\mathbf{x}}]$$

This relaxation enables relocation of a vertex along a single boundary primitive granting deviations from the input surface for better mesh quality. In order to ensure that the updated vertex still lies on the boundary, we need to bound λ_x such that $\mathbf{x}$ does not leave the boundary. Algorithm 3 exhibits our method of finding a descent direction for a vertex on the boundary. During the optimization, our algorithm maintains the location of the vertex on the boundary, which leads to the following three states:

1. **On vertex:** The vertex overlaps with a boundary vertex. This is the initial state for each boundary vertex.
2. **On edge:** The vertex lies on a boundary edge but not on either of its vertices.
3. **On triangle:** The vertex lies within a boundary triangle but not on any of its edges.

Depending on the state, we calculate the directional derivative for each boundary primitive adjacent to the vertex. Our algorithm checks for each directional derivative, if its descent direction does not relocate the vertex away from $\mathscr{B}$, i.e., conforms to $\mathscr{B}$. Following the rule of steepest descent, we choose the directional derivative with the largest magnitude. Figure 3 shows how our algorithm relocates a vertex on the boundary maintaining the state of the vertex. In order to keep the state consistent after gradient descent, we limit the step size such that the vertex remains on its boundary primitive.

After relocation, we check if a vertex of state *on triangle* is now *on edge* or *on vertex*. Likewise, we check if a vertex of state *on edge* is now *on vertex*. This check is outlined by Algorithm 4 and uses the barycentric coordinates of the vertex regarding its current boundary triangle. If one or two barycentric coordinates are close to zero, the vertex is set to the corresponding edge or vertex, respectively.

4.4 Flips

Performing flips in parallel requires conflict detection, because otherwise flipping does not guarantee a valid mesh, as can be seen in Fig. 4. In harmonic triangulations, prioritizing flips by their reduction of the trace leads to good

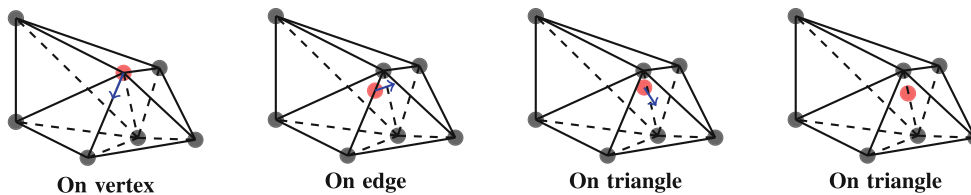

Fig. 3 We use directional derivatives, in order to relocate the vertex (red) along the boundary. Initially, the directional derivative on the boundary with the largest magnitude is along an edge. After relocating the vertex, the directional derivative of the triangle to the right of the edge is chosen. Finally, vertex relocation converges after relocating the vertex on the triangle

Algorithm 3 Algorithm to find directional derivative for boundary vertex

1: **procedure** FINDTANGENTDERIVATIVE($\mathbf{x}$, λ, $\mathbf{g}$)
2: $\mathbf{g}_p \leftarrow 0$ ▷ $\mathbf{g}_p \in \mathbb{R}^d$
3: **if** $\mathbf{x}$ is on boundary vertex $\mathbf{x}_b$ **then**
4: **for** boundary triangle τ_b containing $\mathbf{x}_b$ **do**
5: $(\mathbf{u}_1, \mathbf{u}_2) \leftarrow$ *tangent_plane*(τ_b)
6: $\mathbf{d}_p \leftarrow (\mathbf{u}_1^\top \mathbf{g})\mathbf{u}_1 + (\mathbf{u}_2^\top \mathbf{g})\mathbf{u}_2$
7: **if** ray $\mathbf{x} - \beta \mathbf{d}_p$ intersects $\tau_b \setminus \mathbf{x}_b \wedge \|\mathbf{g}_p\|_2 < \|\mathbf{d}_p\|_2$ **then**
8: $\mathbf{g}_p \leftarrow \mathbf{d}_p$
9: $\lambda \leftarrow \beta_0$ ▷ For β_0 point along ray is on $\tau_b \setminus \mathbf{x}_b$
10: Set $\mathbf{x}$ on τ_b
11: **end if**
12: **end for**
13: **for** boundary edge e_b containing $\mathbf{x}_b$ **do**
14: $\mathbf{u}_1 \leftarrow$ *tangent_line*(e_b)
15: $\mathbf{d}_p \leftarrow (\mathbf{u}_1^\top \mathbf{g})\mathbf{u}_1$
16: **if** ray $\mathbf{x} - \beta \mathbf{d}_p$ intersects $e_b \setminus \mathbf{x}_b \wedge \|\mathbf{g}_p\|_2 < \|\mathbf{d}_p\|_2$ **then**
17: Analogous to lines 8 and 9
18: Set $\mathbf{x}$ on e_b
19: **end if**
20: **end for**
21: **else if** $\mathbf{x}$ is on boundary edge e_b **then**
22: **for** boundary triangle τ_b adjacent to e_b **do**
23: Analogous to lines 5 and 6
24: $(\mathbf{x}_0, \mathbf{x}_1) \leftarrow e_b$
25: **if** ray $\mathbf{x} - \beta d_p$ intersects $\tau_b \setminus \mathbf{x}_0$ or $\tau_b \setminus \mathbf{x}_1$ **then**
26: **if** $\|\mathbf{g}_p\|_2 < \|\mathbf{d}_p\|_2$ **then**
27: Analogous to lines 8 - 10
28: **end if**
29: **end if**
30: **end for**
31: Check e_b analogous to lines 14 - 19
32: **else** $\mathbf{x}$ is on boundary triangle τ_b
33: **for** boundary edge e_b in τ_b **do**
34: Check e_b analogous to lines 14 - 19
35: **end for**
36: **end if**
37: **return** $\mathbf{g}_p$
38: **end procedure**

Algorithm 4 Algorithm to identify the boundary state

1: **procedure** IDENTIFYBOUNDARYSTATE($\mathbf{x}$)
2: **if** $\mathbf{x}$ is on a boundary triangle τ_b **then**
3: $\lambda_0, \lambda_1, \lambda_2 \leftarrow$ *barycentrics*$(\mathbf{x}, \tau_b)$
4: **if** $\lambda_i \approx 0 \wedge \lambda_j \approx 0 \wedge j \neq i$ **then**
5: Set $\mathbf{x}$ on corresponding edge
6: **else if** $\lambda_i \approx 0$ **then**
7: Set $\mathbf{x}$ on corresponding vertex
8: **end if**
9: **end if**
10: **if** $\mathbf{x}$ is on a boundary edge e_b **then**
11: $\lambda_0, \lambda_1 \leftarrow$ *barycentrics*$(\mathbf{x}, e_b)$
12: **if** $\lambda_i \approx 0$ **then**
13: Set $\mathbf{x}$ on corresponding vertex
14: **end if**
15: **end if**
16: **end procedure**

element quality [1]. We exploit this property and resolve the issue of conflict detection by finding the locally most beneficial harmonic flip, as shown in Fig. 5. As a result, we are able to perform harmonic flips massively parallel without significant differences to sequential computations. We present Algorithm 5 that identifies feasible and locally most beneficial flips. We encode flips by the index of the flipped mesh facet and an identifier for the type of the flip:

$$\mathit{flip} = (\mathtt{i}, \mathtt{id}) \in \mathbb{Z} \times \{\text{2-3-flip, 3-2-flip}, \emptyset\} \tag{16}$$

Our algorithm performs a parallel pass over all $\tau \in \mathrm{T}$ to identify the most beneficial flip for each τ. Each τ can be flipped at either one of its six edges or one of its four faces. Hence, we evaluate feasibility checks and quality improvements regarding η of the potential flips in a predetermined order. Face or edge flips are only feasible on interior faces or edges, respectively. Each flip requires its incident tetrahedra to form a convex subtriangulation. Whenever a flip is feasible, we compare its quality improvement to the currently most beneficial flip. As a result, we obtain the most beneficial harmonic flip for τ. If no harmonic flip has been found, our flipping algorithm terminates. Otherwise, we proceed with another parallel pass over all $\tau \in \mathrm{T}$.

In order to prepare for building a new Triangulation T', we allocate an array of markers indicating whether $\tau \in \mathrm{T}$ is part of T' or not and an array of integers for the number of newly added tetrahedra. For each $\tau \in \mathrm{T}$, we find locally most beneficial flips in parallel. Using flip type and facet index, we obtain the tetrahedra incident to the flip using the precomputed connectivity relationships. No conflict occurs, if the flip is the most beneficial flip for each incident tetrahedron in the convex region of the flip. In this case, the flip is locally the most beneficial and is selected to be performed. Consequently, the tetrahedron associated with the thread can be marked for removal. We elect a coordinator thread to perform the flip. If the index of τ is the lowest of the incident tetrahedra, the associated thread is declared as the coordinator. The

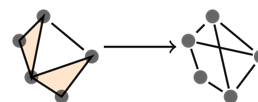

Fig. 4 Suppose one thread is assigned to each of the two orange triangles adjacent to the white triangle. Concurrently, both threads perform a flip, which leads to an invalid triangulation

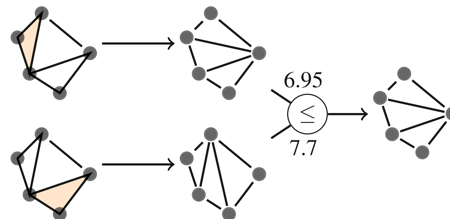

Fig. 5 In case of conflicting flips, we choose the locally most beneficial flip. As a result, we obtain a valid triangulation

coordinator thread sets its integer value to the number of tetrahedra added by performing the flip. Since only coordinator threads write to the array of integers, thread i is a coordinator thread if this array holds a non-zero entry at position i.

An exclusive prefix sum over the integers for new tetrahedra provides offset positions and the total number of tetrahedra to be added. The marker values of the tetrahedra in sum amount to the number of remaining tetrahedra. We allocate a new buffer for the resulting tetrahedra and copy the remaining tetrahedra through a stream compaction to a newly allocated buffer. In a final parallel pass over the tetrahedra, the coordinator threads perform the flips and append the resulting tetrahedra to the remaining tetrahedra using the offset positions.

4.5 Combined vertex relocation and flipping

We perform several alternating passes of vertex relocation and harmonic flipping. Our algorithm terminates if its effect on the mesh becomes insignificant. Gradient descent converges if the gradient approaches zero. Therefore, we terminate if $\nabla \mathrm{tr}(\mathbf{L}_\mathrm{T})$ is sufficiently small. Thus, when $\|\nabla \mathrm{tr}(\mathbf{L}_\mathrm{T})\|$ is smaller than some ϵ_c, gradient descent is not expected to cause significant improvements. In addition, update rates can become vanishingly small. To avoid this situation, we terminate if the difference of the current to the prior gradient is smaller than ϵ_c. As $\mathrm{tr}(\mathbf{L}_\mathrm{T})$ is scale dependent, we advise to choose a relative ϵ_c. We opt for choosing ϵ_c based on a constant ϵ governing the accuracy in finding a minimum:

$$\epsilon_c \leftarrow \max(\epsilon,\ \epsilon \|(\nabla \mathrm{tr}(\mathbf{L}_\mathrm{T}))\|) \tag{17}$$

As some vertices converge more quickly than others, we do not further optimize a vertex with a gradient norm smaller than ε_g. We choose $\epsilon = 10^{-5} = \varepsilon_g$ and $\|\cdot\|_2^2$ as the norm in our work.

Connectivity relationships and coloring need to be updated after flipping. Checking for flips is an unnecessary overhead if flips are unlikely to be found. Thus, we apply a heuristic reducing the number of checks. A counter k_f holds the number of iterations without flip checking and is initialized as $k_f = 1$. Whenever flip checking fails to find flips, we double k_f. Analogously, if flip checking finds flips, k_f is halved rounding up. If the counter has reached a predetermined number 2^N, we terminate, as additional flips are unlikely to be found. We opt for choosing $N = 3$. In summary, we terminate at iteration i, if one of the following conditions is met:

(C1) $\|(\nabla \mathrm{tr}(\mathbf{L}_\mathrm{T}))_i\| < \epsilon_c$
(C2) $\|(\nabla \mathrm{tr}(\mathbf{L}_\mathrm{T}))_i - (\nabla \mathrm{tr}(\mathbf{L}_\mathrm{T}))_{i-1}\| < \epsilon_c$
(C3) $k_f = 2^N$

Algorithm 5 Parallel flipping algorithm to optimize $\mathscr{M}$

```
1: procedure FLIPMESH(M = (T, V))
2:   flips ← ((-1, ∅), ..., (-1, ∅)) ∈ (ℤ×{2-3-flip, 3-2-flip, ∅})^|T|
3:   for i = 0, ..., |T| − 1 do                         ▷ Find flips in parallel
4:     η_impr ← 0; flip ← (-1, -1); τ ← T_i
5:     for all faces f in τ do                         ▷ In predetermined order
6:       if 2-3-flip(f, τ) is feasible ∧ η_impr < η_flip then
7:         flips_i ← (index_of(f), 2-3-flip)
8:       end if
9:     end for
10:     for all edges e in τ do                        ▷ In predetermined order
11:       if 3-2-flip(e, τ) is feasible ∧ η_impr < η_flip then
12:         flips_i ← (index_of(e), 3-2-flip)
13:       end if
14:     end for
15:   end for
16:   if (-1, ∅) = flips_i for i = 0, ..., |T| − 1 then
17:     return false
18:   end if
19:   new_tets ← (0, ..., 0) ∈ ℤ^|T|
20:   tets_marked ← (1, ..., 1) ∈ {0, 1}^|T|
21:   for i = 0, ..., |T| − 1 do                       ▷ Detect conflicts in parallel
22:     (j, type) ← flips_i; is_coordinator ← true; agree ← true
23:     if j = -1 then continue end if
24:     for all tetrahedron τ involved in flip(j, type) do
25:       k ← index_of(τ); (j_adj, type_adj) ← flips_k
26:       agree ← agree ∧ type = type_adj ∧ j = j_adj
27:       is_coordinator ← is_coordinator ∧ i ≤ k
28:     end for
29:     if agree then tets_marked_i ← 0 end if
30:     if agree ∧ is_coordinator then
31:       new_tets_i ← if type = 2-3-flip then 3 else 2 end if
32:     end if
33:   end for
34:   offsets ← (0, ..., 0) ∈ ℤ^(|T|+1)
35:   for i = 1, ..., |T|: offsets_i ← offsets_(i−1) + new_tets_(i−1)
36:   N_remaining ← Σ_(i=0)^(|T|−1) tets_marked_i
37:   T′ ← allocate(N_remaining + offsets_|T|)
38:   copy_if_marked(src = T, dst = T′, tets_marked)
39:   for i = 0, ..., |T| − 1 do                       ▷ Perform flips in parallel
40:     if new_tets_i ≠ 0 then
41:       (j, type) ← flips_i; offset ← N_remaining + offsets_i
42:       T′_offset ← flip(j, type)
43:     end if
44:   end for
45:   M ← (T′, V)
46:   return true
47: end procedure
```

5 Results

We present experiments to demonstrate the benefits of our algorithms from Sect. 4. To ensure a fair comparison, we implemented the algorithms from scratch using C++ and CUDA [22]. We compiled the code using Visual Studio 2019 and CUDA 11.2 on Windows 10. We ran the experiments on a machine equipped with an NVIDIA RTX 3090 GPU and an Intel i7 3930K CPU. In order to avoid outliers in run time

measurements, we have determined the median run time from 10 executions.

5.1 Parallel harmonic flips

We compare our GPU parallel harmonic flipping algorithm performing locally most beneficial flips to the sequential CPU algorithm performing flips in an ordered queue. We perform flips on the input mesh, until no further harmonic flips can be found. The results appear in Table 1. While Alexa [1] performed harmonic flips on Delaunay triangulations of point sets, we perform flips on meshes generated with Tetgen [30] a constrained Delaunay mesher, leading to a lesser reduction of the number of tetrahedra. We detail the exact numbers of tetrahedra in the resulting triangulations, in order to show that postponing locally not most beneficial flips to later flipping passes does not lead to significant differences in the resulting triangulation. Our experiments reveal substantial speedups of 106×–318×. As harmonic bistellar flips either coincide with the Delaunay triangulation or reduce a triangulation of three tetrahedra to two tetrahedra, our parallel flipping algorithm is a useful tool for mesh optimization and generation, quickly reducing the tetrahedron count while somewhat preserving the Delaunay criterion. Our experiments confirm that harmonic flipping well preserves the percentage of locally Delaunay tetrahedra.

5.2 Robustness

In order to validate the practicability of our parallel algorithms, we have applied our optimization algorithm in Sect. 4.5 to the 10k tetrahedral meshes generated by Hu et al. [13]. Our algorithm did not produce any inversion due to the choice of the inversion free interval for each vertex $\mathbf{x}$. After termination, each triangular face was connected to one or two tetrahedra. In addition, we consistently observed alternating face orientations for triangular faces adjacent to two tetrahedra. The Manhattan distance of distinct vertices was larger than 10^{-10} for all except for two meshes meaning that our optimization method does not produce geometrically duplicated vertices. For the two meshes with geometrically close vertices, we observed smaller vertex distances already before optimization.

We calculate the one-sided Hausdorff distance of the boundary vertices of the optimized mesh to the input mesh surface, in order to validate that our vertex relocation algorithm on the boundary (c.f. Sect. 4.3) keeps vertices on the boundary. We divide the resulting distances by the average boundary edge length to put them in relation to the dimensions of the model. For 99.95% of the meshes, the one-sided Hausdorff distance was below 10^{-3}, which shows that boundary vertices remain on the input surface considering round off errors. In four out of the five remaining cases, roundoff errors on directional derivative calculation accumulate to a degree that the resulting deviation is roughly 10^{-2}. In only one case, a significant deviation of .19 can be observed. As the meshes generated by Hu et al. [13] generally are of high quality and already optimized, the experiments regarding runtime, convergence and mesh quality use unoptimized meshes.

5.3 Element quality and convergence

We investigate resulting element quality and convergence of both Alexa's method [1] and our method. Our work covers two methods of using directional derivatives at the boundary. Using directional derivatives only for vertices in $V_{\mathscr{F}}$ and $V_{\mathscr{E}}$ provides surface preservation ($\partial T = \mathscr{B}$). In addition, directional derivatives can be used to move vertices along the input surface ($\partial T \cong \mathscr{B}$) to optimize all vertices at the cost of altering the model shape. We compare Alexa's reprojection-based method [1] to both variants of boundary treatment.

Our boundary preserving method is most useful for input meshes with few corner vertices. Thus, we investigate the

Table 1 We compare our harmonic flipping algorithm to the original harmonic flipping algorithm. The table includes the exact numbers for |T| to show that our algorithm does not produce significantly different results. We observe substantial speedups of 106×–318×

Mesh		Sequential flipping		Ours (GPU)	
Name	|T|	Time (sec)	|T|	Time (sec)	|T|
Block	78690	.425	76037	0.004	76037
Ghost	160799	1.139	154263	0.005	154263
Die	232767	1.751	225689	0.006	225689
Snowman	227567	1.704	217837	0.006	217837
Barrel	463797	4.649	443838	0.014	443838
Falcon	1072784	8.178	1034101	0.027	1034103
Part	1099271	8.583	1057930	0.030	1057930
Cube	1538635	15.282	1471163	0.048	1471162
World	1786620	16.504	1725729	0.055	1725730
Pot	4034608	34.847	3886310	0.137	3886309

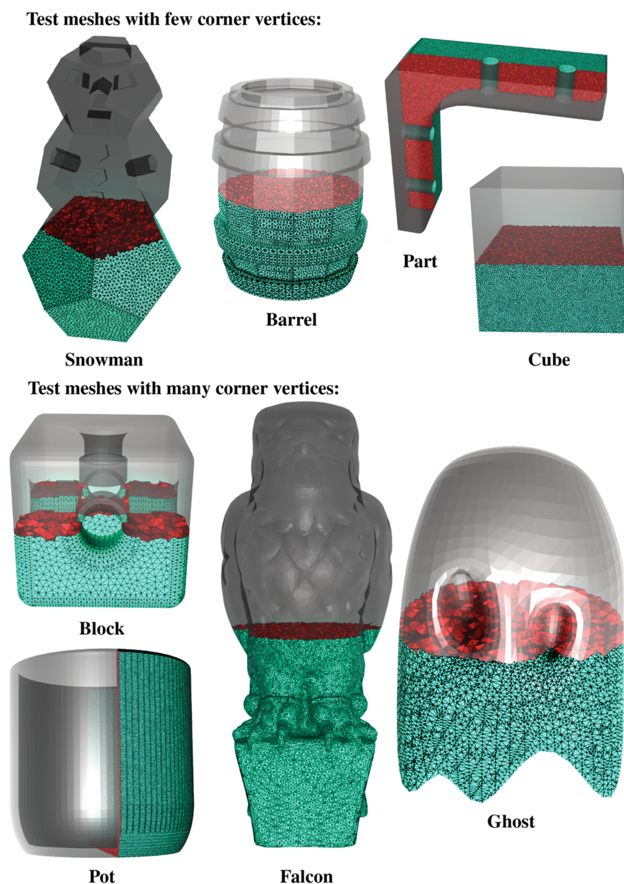

Fig. 6 Test meshes used in the evaluation

boundary preserving method on the top four meshes shown in Fig. 6 and provide the results in Table 2. Although all of the input meshes include critical minimal dihedral angles, both methods achieve to improve the minimal angle, while our method achieves significantly larger minimal angles with the exception of similar minimal angles for the Part. Likewise, the lower 5% of dihedral angles is significantly larger with the exception of the Part. If we relocate every boundary vertex of the Part model, we achieve a minimal dihedral angle of 8.12° and a lower 5-percentile $\phi_{5\%}$ of 38.63°, which is a better result than the reprojection-based method. Relocating every boundary vertex does not result in significant differences for the other three meshes. Our method consistently results in lower energy states for η regarding the maximum, 95-percentile and the sum over T.

For meshes with many corner vertices forming curved surfaces, optimizing all boundary vertices is important, as boundary preserving optimization typically results in lower minimal angles oftentimes not even half as large. We evaluate our method on the bottom four meshes shown in Fig. 6 and provide the results in Table 3. While our method improves the minimal angles of all inputs, reprojection-based optimization impairs the initial minimal angles on the Block and Pot meshes. As the reprojection step does not respect energy minimization, a degradation of mesh quality may occur. Using directional derivatives along the boundary respects energy minimization leading to lower energy states for η with the exception of the 95-percentile of the Block.

We have investigated the impact on the mesh surfaces and the convergence of the optimization methods on different meshes. We present typical results in Fig. 7. While reprojection of boundary vertices distorts sharp detail, directional derivatives along boundary faces and edges can be used to preserve the mesh surface. Moreover, the reprojection step mitigates convergence, because it does not respect energy minimization. On the contrary, gradient descent of directional derivatives converges to a local minimum on the boundary, as can be seen in the monotonously decreasing curve of the gradient norm for the Barrel. We observe convergence for relocating every vertex along the boundary as well. The reprojection based optimization oftentimes terminates in a premature state. Since Alexa's [1] algorithm potentially chooses small step sizes, reprojection to the closest point on the mesh surface oftentimes does not significantly change vertex positions from the initial state leaving a lot of optimization potential. Directional derivatives along the boundary respect energy minimization even when migrating to different boundary primitives. However, the gradient norm does not reduce as monotonous as for choosing a constant boundary primitive, as gradient norms change, when a vertex is associated with another boundary primitive. Convergence is achieved though, while the input shape is approximately preserved. This is notable, as our use of directional derivatives enables robust improvement in high-resolution meshes and keeps boundary vertices on the boundary while converging.

5.4 Run time

We compare run times of Alexa's [1] and our massively parallel algorithm for full and approximate boundary preservation. Figure 8 shows the run time comparisons for full and approximate boundary preservation. For full boundary preservation, we achieve notable speedups of 9.17×–119×. Although the boundary reprojection prevents convergence on these meshes, the competing algorithm still performs a considerable number of iterations until no harmonic flips can be found. This is not the case for the more complex meshes we used for comparison with our vertex relocation along the boundary (c.f. Sect. 4.3). The reduced convergence of reprojecting vertices on the boundary leads to lower iteration numbers of the competing optimization algorithm. Additionally, our method for optimizing vertices along the boundary imposes more branching than the full boundary preservation reducing the impact of massively parallel processing

Table 2 We provide dihedral angles ϕ and energy states using Alexa's [1] and our method ($\partial T = \mathscr{B}$) on meshes with few corner vertices

Name	Input					Alexa's [1]					Ours $\partial T = \mathscr{B}$				
	ϕ_{min}	$\phi_{5\%}$	η_{max}	$\eta_{95\%}$	$\eta(T)$	ϕ_{min}	$\phi_{5\%}$	η_{max}	$\eta_{95\%}$	$\eta(T)$	ϕ_{min}	$\phi_{5\%}$	η_{max}	$\eta_{95\%}$	$\eta(T)$
Snowman	0.1	27.46	1849.08	11.4	1.33e+06	3.7	33.4	44.68	7.69	1.14e+06	**8.67**	**36.32**	**18.55**	**6.72**	**1.05e+06**
Barrel	0.12	27.71	530.69	2.92	828788	9.26	36.62	4.94	2.03	671173	**10.84**	**36.66**	**4.46**	**2**	**659666**
Part	0.26	30.38	322.04	6.76	4.53e+06	**7.3**	**38.57**	11.8	4.95	3.88e+06	7.07	38.54	**11**	**4.9**	**3.83e+06**
Cube	0.08	29.12	134.4	.52	495820	6.8	37.48	1.29	0.37	410171	**10.72**	**38.48**	**0.8**	**0.35**	**400212**

Table 3 We provide dihedral angles ϕ and energy states using Alexa's [1] and our method ($\partial T \cong \mathscr{B}$) on meshes with many corner vertices

Name	Input					Alexa's [1]					Ours $\partial T \cong \mathscr{B}$				
	ϕ_{min}	$\phi_{5\%}$	η_{max}	$\eta_{95\%}$	$\eta(T)$	ϕ_{min}	$\phi_{5\%}$	η_{max}	$\eta_{95\%}$	$\eta(T)$	ϕ_{min}	$\phi_{5\%}$	η_{max}	$\eta_{95\%}$	$\eta(T)$
Block	4.65	27.22	82.37	20.31	960808	3.98	**34.32**	47.5	**15.21**	796930	**9.84**	34.2	**32.77**	15.42	**770264**
Ghost	0.03	25.36	9622.21	10.42	982795	1.72	29.93	83.42	8.36	840025	**2.79**	**33.5**	**22.34**	**6.93**	**721325**
Falcon	0.65	26.82	217.02	18.62	1.15e+07	2.53	32.87	76.35	14.45	1.01e+07	**3.84**	**34.00**	**32.77**	**13.08**	**9.27e+06**
Pot	4.14	28.79	40.06	10.12	2.51e+07	.04	33.67	2252.44	8.36	2.29e+07	**9.21**	**37.01**	**17.05**	**7.31**	**2.09e+07**

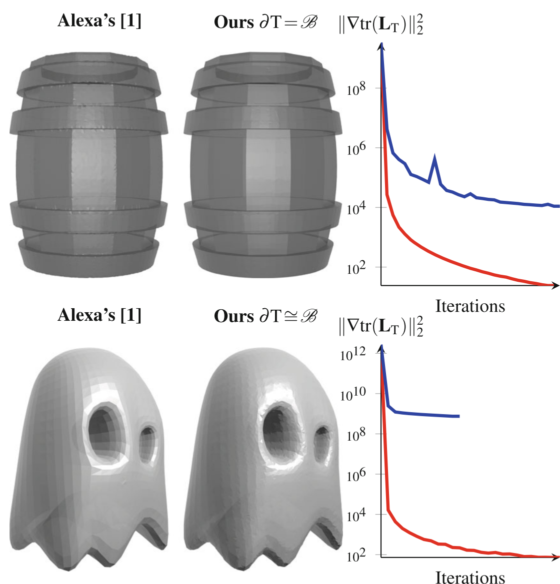

Fig. 7 We compare our method (red) to Alexa's [1] reprojection based method (blue). We visualize resulting boundaries and plot the gradient norm throughout optimization

and leading to up to 40% slower run times. Thus, we obtain lower but still notable speedups of 3.51×–78×.

6 Conclusions

In summary, we have devised an efficient and robustly converging mesh optimization method processing millions of tetrahedra in a few seconds. We have introduced massively parallel algorithms and parallelization strategies for optimizing meshes while preventing inverted elements. Our parallel flipping algorithm achieves speedups of 106×–318× without producing significantly different results from sequential flipping. We have evaluated the use of directional derivatives for boundary treatment in mesh optimization. Our method supports both, full preservation of the surface and optimization of boundary vertices along the surface. The results for using directional derivatives are compelling, as we achieve significantly better mesh quality compared to reprojection, while keeping vertices on the boundary without adding any error terms to the optimization function. Our method tends to smooth sharp surface details, which is a limitation. A natural extension to our method is to calculate the directional derivative for curved surfaces such as CAD bodies to improve precision. As a result of improved convergence and neat parallelization, we accelerate harmonic mesh optimization by up to 119× in the boundary preserving case and by up to 78× for relocating all boundary vertices.

As the convergence is governed by gradient descent, the use of a nonlinear conjugate gradient method [11] or a momentum method [25] might improve convergence. A fast adaptive mesh optimization could be obtained through the use of parallel refinement, which potentially leads to better mesh quality and improved surface approximation. An interesting idea is to incorporate parallel harmonic flipping into GPU-accelerated Delaunay meshers such as [4] to diminish element count and improve mesh quality.

Acknowledgements Open Access funding enabled and organized by Projekt DEAL. The second and the third author were supported by the EC project DIGITbrain, No. 952071, H2020. We thank Marc Alexa for providing sources of the original harmonic mesh optimization implementation.

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

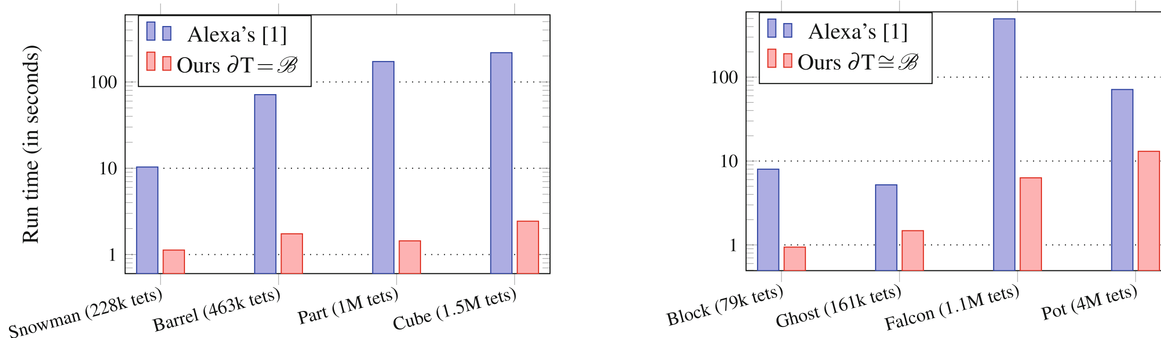

Fig. 8 Run times of Alexa' algorithm [1] compared to ours for full boundary preservation (left) and approximate boundary preservation (right)

Appendix

Proof of Eq. (8): We know from [1] that:

$$(\mathbf{X}_\tau \mathbf{M}_d)^{-\top}\mathbf{M}_d^\top = \frac{-1}{d v_\tau}(a_{\tau\setminus\tau_0}\mathbf{n}_{\tau\setminus\tau_0}, \ldots, a_{\tau\setminus\tau_d}\mathbf{n}_{\tau\setminus\tau_d}) \quad (18)$$

$$(\mathbf{L}_\tau)_{ij} = \frac{1}{d^2 v_\tau} a_{\tau\setminus\tau_i} a_{\tau\setminus\tau_j} \mathbf{n}_{\tau\setminus\tau_i}^\top \mathbf{n}_{\tau\setminus\tau_j} \quad (19)$$

Inserting Eq. (18) and Eq. (3) in Eq. (7) results in:

$$\frac{\partial \mathrm{tr}(\mathbf{L}_\tau)}{\partial \mathbf{X}_\tau} = \frac{-1}{d! d v_\tau}(a_{\tau\setminus\tau_0}\mathbf{n}_{\tau\setminus\tau_0}, \ldots, a_{\tau\setminus\tau_d}\mathbf{n}_{\tau\setminus\tau_d})\left(\frac{\sum_{\mathbf{v}\in\tau} a_{\tau\setminus\mathbf{v}}^2}{d^2|v_\tau|}\cdot\mathbf{I} - 2\mathbf{L}_\tau\right) \quad (20)$$

We develop the matrix on the right side of the product. While the non-diagonal entries are directly given by Eq. (19), the diagonal entries evaluate to:

$$\left(\frac{\sum_{\mathbf{v}\in\tau} a_{\tau\setminus\mathbf{v}}^2}{d^2|v_\tau|}\cdot\mathbf{I} - 2\mathbf{L}_\tau\right)_{ii} = \frac{1}{d^2}\left(\frac{\sum_{\mathbf{v}\in\tau} a_{\tau\setminus\mathbf{v}}^2}{|v_\tau|} - \frac{2}{v_\tau}a_{\tau\setminus\tau_i}^2\right) \quad (21)$$

Without loss of generality, we assume that $\mathbf{x} = \tau_i \in \tau$. As the goal is to obtain a formula for the partial derivative regarding $\mathbf{x} \in \tau$, we evaluate the product of the left row vector with the ith column vector of the matrix on the right side of Eq. (20) to obtain the harmonic gradient for a single vertex:

$$\frac{\partial \mathrm{tr}(\mathbf{L}_\tau)}{\partial \tau_i} = \frac{-1}{d d v_\tau}\left(\frac{a_{\tau\setminus\tau_i}}{d^2}\mathbf{n}_{\tau\setminus\tau_i}\left(\frac{\sum_{\mathbf{v}\in\tau} a_{\tau\setminus\mathbf{v}}^2}{|v_\tau|} - \frac{2}{v_\tau}a_{\tau\setminus\tau_i}^2\right) - 2\sum_{\substack{j=0\\ j\neq i}}^{d} a_{\tau\setminus\tau_j}\mathbf{n}_{\tau\setminus\tau_j}(\mathbf{L}_\tau)_{ji}\right)$$

We insert Eq. (19) for $(\mathbf{L}_\tau)_{ji}$ and simplify to:

$$\frac{\partial \mathrm{tr}(\mathbf{L}_\tau)}{\partial \tau_i} = \frac{a_{\tau\setminus\tau_i}}{d! d^3 v_\tau}\left(\left(\frac{2a_{\tau\setminus\tau_i}^2}{v_\tau} - \eta(\tau)\right)\mathbf{n}_{\tau\setminus\tau_i} + 2\sum_{\mathbf{v}\in(\tau\setminus\tau_i)} a_{\tau\setminus\mathbf{v}}^2 \frac{\mathbf{n}_{\tau\setminus\tau_i}^\top \mathbf{n}_{\tau\setminus\mathbf{v}}}{v_\tau}\mathbf{n}_{\tau\setminus\mathbf{v}}\right)$$

If we let $\tau_i = \mathbf{x}$, we obtain Eq. (8) □

References

1. Alexa, M.: Harmonic triangulations. ACM Trans. Gr. **38**(4), 1–14 (2019)
2. Alexa, M., Herholz, P., Kohlbrenner, M., Sorkine-Hornung, O.: Properties of Laplace operators for tetrahedral meshes. Computer Gr. Forum **39**(5), 55–68 (2020)
3. Benítez, D., Rodríguez, E., Escobar, J.M., Montenegro Armas, R.: Parallel optimization of tetrahedral meshes. In: Proceedings of the 6th European Conference on Computational Mechanics: Solids, Structures and Coupled Problems, ECCM 2018 and 7th European Conference on Computational Fluid Dynamics, pp. 4403–4412 (2018)
4. Cao, T.T., Nanjappa, A., Gao, M., Tan, T.S.: A GPU accelerated algorithm for 3d delaunay triangulation. In: Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D 14, pp. 47–54. ACM Press (2014)
5. D'Amato, J., Vénere, M.: A CPU–GPU framework for optimizing the quality of large meshes. J. Parallel Distrib. Comput. **73**(8), 1127–1134 (2013)
6. Dassi, F., Kamenski, L., Farrell, P., Si, H.: Tetrahedral mesh improvement using moving mesh smoothing, lazy searching flips, and rbf surface reconstruction. Computer-Aided Des. **103**, 2–13 (2018)
7. Drakopoulos, F., Tsolakis, C., Chrisochoides, N.P.: Fine-grained speculative topological transformation scheme for local reconnection methods. AIAA J. **57**(9), 4007–4018 (2019)
8. Freitag, L., Jones, M., Plassmann, P.: A parallel algorithm for mesh smoothing. SIAM J. Scientif. Comput. **20**(6), 2023–2040 (1999)
9. Freitag, L.A., Ollivier-Gooch, C.: Tetrahedral mesh improvement using swapping and smoothing. Int. J. Numer. Methods Eng. **40**(21), 3979–4002 (1997)
10. Fu, X.M., Liu, Y., Guo, B.: Computing locally injective mappings by advanced MIPS. ACM Trans. Gr. **34**(4), 1–12 (2015)
11. Hager, W.W., Zhang, H.: A survey of nonlinear conjugate gradient methods. Pacif. J. Optim. **2**(1), 35–58 (2006)
12. Hormann, K., Greiner, G.: Mips: An efficient global parametrization method. Erlangen-nuernberg univ (germany) computer graphics group, Tech. rep. (2000)
13. Hu, Y., Schneider, T., Wang, B., Zorin, D., Panozzo, D.: Fast tetrahedral meshing in the wild. ACM Trans. Gr. **39**(4), 1–117 (2020)
14. Ibanez, D., Shephard, M.: Mesh adaptation for moving objects on shared memory hardware. techreport 2016-24, Rensselaer Polytechnic Institute (2016). https://scorec.rpi.edu/REPORTS/2016-24.pdf
15. Klingner, B.M., Shewchuk, J.R.: Aggressive tetrahedral mesh improvement. In: Proceedings of the 16th International Meshing Roundtable, pp. 3–23 (2007)
16. Knupp, P.M.: Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. part II?a framework for volume mesh optimization and the condition number of the jacobian matrix. Int. J. Numer. Methods Eng. **48**(8), 1165–1185 (2000)
17. Liu, H.T.D., Jacobson, A., Ovsjanikov, M.: Spectral coarsening of geometric operators (2019)
18. Lo, D.S.H.: Finite element mesh generation. CRC Press, Boston (2014)
19. Manteaux, P.L., Wojtan, C., Narain, R., Redon, S., Faure, F., Cani, M.P.: Adaptive physically based models in computer graphics. Computer Gr. Forum **36**(6), 312–337 (2017)
20. Mueller-Roemer, J.S., Altenhofen, C., Stork, A.: Ternary sparse matrix representation for volumetric mesh subdivision and processing on GPUs. Computer Gr. Forum **36**(5), 59–69 (2017)
21. Mueller-Roemer, J.S., Stork, A.: GPU-based polynomial finite element matrix assembly for simplex meshes. Computer Gr. Forum **37**(7), 443–454 (2018)
22. Nvidia: Cuda 11.2. [Online; accessed May-2022] (2022). https://developer.nvidia.com/cuda-downloads
23. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C, 2 edn. Cambridge University Pr. (2002)
24. Rabinovich, M., Poranne, R., Panozzo, D., Sorkine-Hornung, O.: Scalable locally injective mappings. ACM Trans. Gr. **36**(2), 1–16 (2017)
25. Ruder, S.: An overview of gradient descent optimization algorithms (2016)

26. Shang, M., Zhu, C., Chen, J., Xiao, Z., Zheng, Y.: A parallel local reconnection approach for tetrahedral mesh improvement. Proc. Eng. **163**, 289–301 (2016)
27. Shewchuk, J.R.: What is a good linear finite element? Interpolation, conditioning, anisotropy, and quality measures. Preprint, University of California at Berkeley (2002). https://people.eecs.berkeley.edu/~jrs/papers/elemj.pdf
28. Shontz, S.M., Varilla, M.A.L., Huang, W.: A parallel variational mesh quality improvement for tetrahedral meshes. Proceedings of the 28th International Meshing Roundtable (2020)
29. Shontz, S.M., Vavasis, S.A.: A mesh warping algorithm based on weighted laplacian smoothing. In: IMR, pp. 147–158. Citeseer (2003)
30. Si, H.: TetGen, a Delaunay-based quality tetrahedral mesh generator. ACM Trans. Math. Softw. **41**(2), 1–36 (2015)
31. Smith, J., Schaefer, S.: Bijective parameterization with free boundaries. ACM Trans. Gr. **34**(4), 1–9 (2015)
32. Stein, O., Grinspun, E., Wardetzky, M., Jacobson, A.: Natural boundary conditions for smoothing in geometry processing. ACM Trans. Gr. **37**(2), 1–13 (2018)
33. Ströter, D., Krispel, U., Mueller-Roemer, J., Fellner, D.: TEdit: A distributed tetrahedral mesh editor with immediate simulation feedback. In: Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (2021)
34. Weber, D., Mueller-Roemer, J., Altenhofen, C., Stork, A., Fellner, D.: Deformation simulation using cubic finite elements and efficient *p*-multigrid methods. Computers Gr. **53**, 185–195 (2015)
35. Wicke, M., Ritchie, D., Klingner, B.M., Burke, S., Shewchuk, J.R., Obrien, J.F.: Dynamic local remeshing for elastoplastic simulation. ACM Trans. Gr. **29**(4), 1–11 (2010)
36. Xi, N., Sun, Y., Xiao, L., Mei, G.: Designing parallel adaptive laplacian smoothing for improving tetrahedral mesh quality on the GPU. Appl. Sci. **11**(12), 5543 (2021)
37. Xu, K., Cheng, Z.Q., Wang, Y., Xiong, Y., Zhang, H.: Quality encoding for tetrahedral mesh optimization. Computers Gr. **33**(3), 250–261 (2009)
38. Yin, J., Teodosiu, C.: Constrained mesh optimization on boundary. Eng. Computers **24**(3), 231–240 (2008)
39. Zhang, H., Kaick, O.v., Dyer, R.: Spectral methods for mesh processing and analysis. In: Eurographics 2007 - State of the Art Reports (2007)
40. Zint, D., Grosso, R.: Discrete mesh optimization on GPU. In: Lecture Notes in Computational Science and Engineering, pp. 445–460 (2019)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Daniel Ströter received his M.Sc. in Computer Science at TU Darmstadt in 2019. He received the best thesis award of Fraunhofer Institute for Computer Graphics Research (IGD) for his master thesis in GPGPU accelerated tetrahedral mesh processing. His research focuses on massively parallel algorithms for geometry processing, modeling, and rendering, in particular for volumetric meshes. As a PhD candidate at the Interactive Graphics Systems Group of TU Darmstadt, he continues his research efforts.

Johannes S. Mueller-Roemer joined the Interactive Engineering Technologies department of the Fraunhofer Institute for Computer Graphics Research (IGD) in 2011 after receiving his M.Sc. in Information and Media Technology (with honors and best thesis award) from BTU Cottbus. He received the doctorate in computer science (summa cum laude) from TU Darmstadt in 2019. His research interests include massively parallel GPU-accelerated geometry processing, visualization, and simulation.

Daniel Weber studied computer science and received his PhD from the Technische Universität Darmstadt in 2015. Since 2008 Daniel Weber works as a researcher in the department for Interactive Engineering Technologies at the Fraunhofer institute for computer graphics. His research interests are physically based simulation in the domains of structural mechanics and fluid dynamics and parallelization on multi-core and many-core architectures. Since 2017 he is a lecturer at the Technische Universität Darmstadt, where he teaches physically based simulation and animation.

Dieter W. Fellner is professor of computer science at TU Darmstadt, Germany and Director of the Fraunhofer Institute of Computer Graphics (IGD) at the same location. He is also professor at TU Graz, Austria, and he currently serves as Chairman of the Fraunhofer Information and Communication Technology Group. His research activities over the last years covered efficient rendering and visualization algorithms, generative and reconstructive modeling, virtual and augmented reality, graphical aspects of internet-based multimedia information systems, cultural heritage and digital libraries as well as visual healthcare technologies. He is a member of the Academia Europaea and a Fellow of the EUROGRAPHICS Association.