High-Performance Graphics (2023)
J. Bikker and C. Gribble (Editors)

Massively Parallel Adaptive Collapsing of Edges for Unstructured Tetrahedral Meshes

D. Ströter[1], A. Stork[1,2] and D. W. Fellner[1,2,3]

[1]Technical University of Darmstadt, Germany [2]Fraunhofer IGD, Germany
[3]Graz University of Technology, Institute of Computer Graphics and Knowledge Visualization, Austria

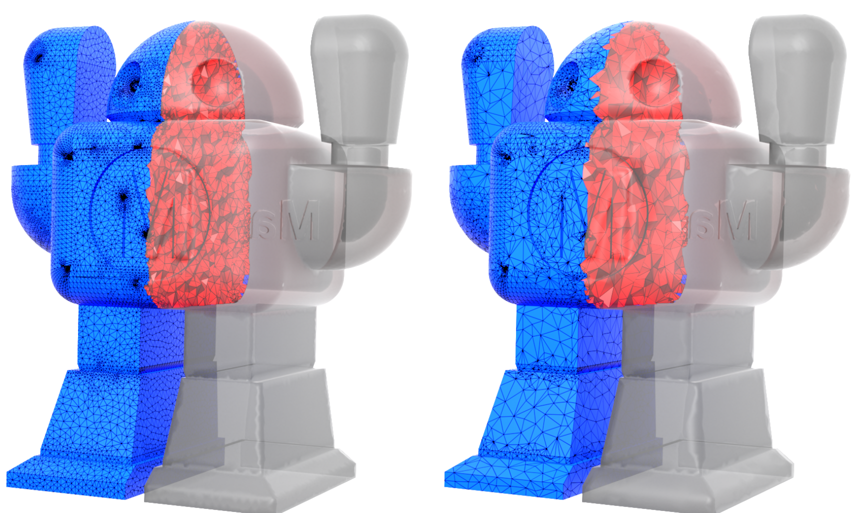

Figure 1: *Using our fast edge collapsing method, we are able to coarsen the high-resolution Robot mesh with 503*k *tetrahedra (left) to a coarser version with 190*k *tetrahedra (right) within 981 milliseconds. The Robot is part of the Thingi10*k *data set [ZJ16] (file ID 255172).*

Abstract

Many tasks in computer graphics and engineering involve unstructured tetrahedral meshes. Numerical methods such as the finite element method (FEM) oftentimes use tetrahedral meshes to compute a solution for complex problems such as physically-based simulation or shape deformation. As each tetrahedron costs computationally, coarsening tetrahedral meshes typically reduces the overhead of numerical methods, which is attractive for interactive applications. In order to enable reduction of the tetrahedron count, we present a quick adaptive coarsening method for unstructured tetrahedral meshes. Our method collapses edges using the massively parallel processing power of present day graphics processing units (GPU)s to achieve run times of up to one order of magnitude faster than sequential collapsing. For efficient exploitation of parallel processing power, we contribute a quick method for finding a compact set of conflict-free sub-meshes, which results in up to 59% fewer parallel collapsing iterations compared to the state of the art massively parallel conflict detection.

CCS Concepts

• ***Theory of computation*** → ***Massively parallel algorithms;*** • ***Computing methodologies*** → ***Volumetric models;*** *Simulation tools; Physical simulation;* • ***Mathematics of computing*** → *Mesh generation;*

1. Introduction

Many applications in computer graphics and engineering compute a solution for a partial differential equation (PDE) over a volumetric domain Ω by discretizing Ω with a mesh. Due to its robustness, unstructured tetrahedral meshing is well-established for approximating volumetric domains [Lo14]. For this reason, many numer-

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.
This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

DOI: 10.2312/hpg.20231139

ical methods for solving PDEs such as the FEM use tetrahedral meshes. Hence, the use of unstructured tetrahedral meshes is part of a wide range of applications including physically-based simulation [WBS*12] and 3D shape deformation [WJBK15].

The generation of tetrahedral meshes is not only concerned with discretizing Ω, because the number and quality of elements affect the accuracy of numerical methods. Element quality refers to the shape of elements [She02]. Typically, increasing the resolution of meshes improves the accuracy but reduces run time performance, as each element costs computationally. Mesh adaptation is a method to adaptively control the mesh resolution. Mesh adaptation algorithms involve iterative cycles of adaptive refinement and coarsening. Many iterations of refining and coarsening are usual and impose high run times. Additionally, the shape quality of elements needs to be sufficient for the application. In fact, one single ill-shaped element, i.e. low-quality element, can lead numerical methods to fail [She02]. Element quality can be improved by optimization algorithms, which oftentimes impose a dominant overhead in mesh generation. Mesh adaptation and optimization typically involve edge collapse operations. Furthermore, collapsing edges in meshes is used to reduce the size of large data sets for visualization [CCM*00; NE04]. GPU-memory efficient rendering of large unstructured meshes is an ongoing research topic [ZWMW23].

In order to improve run times, the parallelization of re-meshing operations and numerical methods is important. Performing numerical methods on complex meshes is one of the driving factors of high performance computing [RGD22], where many machines are orchestrated as nodes of a cluster. Partitioning of meshes with domain decomposition enables parallelization among the machines of a cluster, while each machine should fully exploit its parallel processing power for fast run times. Due to the impressive aggregated processing power of GPUs, current mesh adaptation tools, e.g., NASA's Refine [Par22] or Sandia National Laboratories' Omega_h [Iba22], adopt CUDA to boost on-node parallelism. The literature provides parallelization strategies for many re-meshing operations (see section 2.3), whereas fine-grained parallelization of edge collapsing on GPUs remains a challenge. Thus, our work extends the state of the art by a massively parallel edge collapsing method to quickly coarsen unstructured tetrahedral meshes (see fig. 1). This paper provides the following **contributions**:

- Massively parallel adaptive algorithm for collapsing edges
- Fast conflict detection for fine-grained parallel re-meshing
- Massively parallel boundary preserving robust coarsening of unstructured tetrahedral meshes

2. Background and Related Work

The literature on coarsening of tetrahedral meshes is vast. Section 2.1 describes the foundational concepts. An overview over important applications of tetrahedral meshes appears in section 2.2. We review prior work on parallel re-meshing in section 2.3.

2.1. Edge Collapse in Tetrahedral Meshes

Pioneering works about coarsening triangle meshes such as the progressive meshes by Hoppe [Hop96] inspired many publications about the coarsening of tetrahedral meshes. Staadt et al. [SG98] explore the use of edge collapsing for progressive tetrahedralizations and present various geometric checks necessary for preserving consistency. They check the oriented volumes of simplices to prevent degeneracies and inverted elements. In addition, they check for intersections when collapsing boundary edges to prevent self intersections. Dey et al. [DEGN99] detail the preconditions for collapsing edges under which they preserve the topological type of simplicial complexes up to the third dimension. They show that only collapse operations that satisfy the link condition (see fig. 2) are admissible. As an alternative to collapsing edges, Chopra et al. [CM02] replace one tetrahedron with one vertex to rapidly reduce the count of tetrahedra. This approach is only efficient for interior tetrahedra, because avoiding intersections at the boundary is computationally expensive. Kraus et al. [KE03] present a solution for collapsing boundary edges in a non-convex tetrahedral mesh. To prevent self-intersections they first convexify the mesh computing the convex hull and subsequently check for inversions, whenever a boundary edge of the non-convex mesh is collapsed within the convex hull. As tetrahedral re-meshing involves a variety of operators, Loseille and Menier [LM14] propose a cavity-based re-meshing operator that embeds collapsing, refinement and face/edge swaps. While our massively-parallel conflict detection also finds a set of conflict-free sub-meshes, we focus only on collapsing edges.

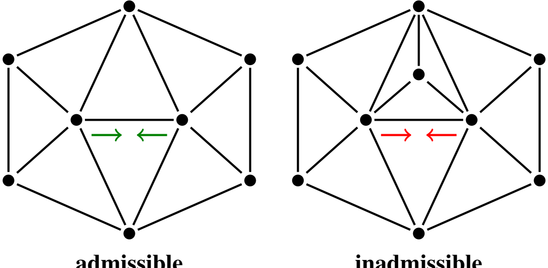

Figure 2: *Collapsing is inadmissible if the set intersection of the two one rings of edge vertices includes simplices that do not contain the to-be-collapsed edge.*

2.2. Applications of Tetrahedral Mesh Coarsening

In order to reduce workloads for scientific visualization, Cignoni et al. [CCM*00] simplify volume data collapsing edges. They involve an error evaluation of scalar data attached to the tetrahedra to obtain accurate visualizations. Similarly, Natarajan and Edelsbrunner [NE04] reduce workloads for visualizing large datasets that represent density functions. They use a modification of the quadric error measure of Garland and Heckbert [GH97] to prioritize collapse operations so that the density function is preserved while improving element quality. Many optimization methods for tetrahedral meshes collapse edges, because it removes low-quality elements [MBAE09]. Tetrahedral mesh generators such as Tetgen [Si20] supply mesh coarsening to adhere to a specified sizing function. A sizing function enables users to govern the size of elements. As each tetrahedral element imposes computational cost for simulations, Cutler et al. [CDM04] collapse edges in a mesh to re-

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.

duce the number of elements, while removing the most low-quality elements. In order to control the accuracy of numerical methods, the adaptation of unstructured grids frequently performs coarsening besides refinement [ALSS06]. Over the years, many tools for mesh adaptation emerged [CRJH09; Par22; Iba22]. The unstructured grid adaptation working group presents a concise overview and qualitative benchmark of many of these tools [IBK*17].

2.3. Parallel Re-Meshing

As the parallelization of re-meshing improves run times of many applications, the literature on parallel re-meshing is vast. Freitag et al. [FJP99] parallelize the relocation of vertices extracting sets of non-adjacent vertices with a graph coloring strategy. Parallel strategies such as Deveci et al. [DBDR16] quickly obtain a high-quality graph coloring. De Cougny et al. [DS99] present parallel refinement and coarsening for distributed environments. The mesh is partitioned in domains and each processor performs adaptation on its assigned domain. Refinement of edges is based on subdivision patterns. Faces of adjacent tetrahedra are triangulated equally to ensure a valid mesh. Collapsing of edges is parallelized among mesh domains. Synchronization is only necessary at the domain boundaries.

As distributed systems orchestrate many machines, sophisticated methods for parallel re-meshing on a single machine emerged. De Coro et al. [DT07] decimate polygonal meshes on the GPU using the geometry shader. They perform prior vertex-clustering and use the rendering pipeline to cull the triangles that become degenerate due to repositioning vertices. Instead of decimating polygonal meshes using the rendering pipeline, we focus on the coarsening of tetrahedral meshes. D' Amato et al. [DV13] present a CPU-GPU framework for parallel element shape optimization determining independent clusters around the worst quality elements. Our method determines more compact sub-meshes for efficient parallelization. Papageorgiou et al. [PP14] present a parallel algorithm for collapsing edges on triangular meshes using the GPU. While their work is optimized for manifold surface meshes, we focus on unstructured tetrahedral meshes. Additionally, their conflict detection requires determination and sorting of super-independent vertices, whereas our conflict detection prioritizes edges by a cost function. Loseille et al. [LMA15] propose a parallel re-meshing algorithm using the cavity-based operator by Loseille and Menier [LM14]. They achieve coarse-grained parallelism using domain decomposition to obtain a set of conflict-free cavities. Chen et al. [CTO20] detail the design of GPU-parallel Delaunay refinement. They show how to concurrently insert Steiner points with high occupancy among GPU threads. Jiang et al. [JDH*22] present a high-level abstraction approach to specify mesh editing algorithms. Their approach schedules operations with a shared memory locking mechanism. Conflicts are avoided by domain decomposition and locking mutexes on the two-ring neighborhood of edges. Our algorithm does not require locking mechanisms or domain decomposition, resulting in more compact sub-meshes.

Recently, Gautron et al. [GK23] present an algorithm for GPU-parallel edge collapsing for triangular surface meshes. Their algorithm packs the cost and the index of an edge in an edge descriptor. They propagate the minimal edge descriptor over the one ring of simplices for each of the two edge vertices. With the use of atomic operations, they prevent conflicts that arise due to massively parallel execution. Their algorithm is part of NVIDIA's Micro-Mesh toolkit [NVI23]. While their algorithm is designed for triangular meshes, it extends to tetrahedral meshes.

Algorithm 1 Edge collapsing iteration

```
 1: procedure COARSENTETMESH(M, C, P, Q, ε_c, edgeMarking)
 2:     edgesMarked ← allocate(M.numEdges())
 3:     edgesMarked.fill(0)
 4:     for i ← 0,...,M.numEdges() do                     ▷ In parallel
 5:         if edgeMarking[i] then
 6:             edgesMarked[i] ← 1
 7:         end if
 8:     end for
 9:     B ← EXTRACTTETMESHBOUNDARY(M)
10:     costs ← allocate(M.numEdges())
11:     for i ← 0,...,M.numEdges() do                     ▷ In parallel
12:         costs[i] ← C(M.edges[i])
13:     end for
14:     COLLAPSEEDGES(M, B, costs, P, Q, ε_c, edgesMarked)
15: end procedure
```

Lots of work addresses parallel re-meshing of triangular meshes, whereas massively parallel collapsing of edges for tetrahedral meshes is a rather unexplored topic. Most of prior work focuses on multi-core architectures, while many-core parallelism has not received as much attention despite its great performance potential. Many-core architectures provide fine-grained parallelism. Collapsing edges with fine-grained parallelism requires determination of compact and conflict-free sub-meshes. Our method uses the mesh connectivity for conflict detection, whereas many previous methods use synchronization primitives, e.g. atomics or mutexes to prevent conflicts, limiting their efficiency to specific systems or hardware.

3. Our Method

This section describes our method to quickly collapse edges of a tetrahedral mesh exploiting massively parallel processing. Figure 4 outlines how our method works.

3.1. Design and Data Structures

The design of the parallel edge collapsing method provides generic functions that can be specified to support specific use cases. The collapsing method depends on an array of vertices (3 real numbers per vertex) and an array of oriented tetrahedra (4 integers per tetrahedron). An outline of one collapsing iteration appears in algorithm 1. A placement strategy $\mathcal{P}$ specifies the point to which the edge is collapsed. As many applications enforce specific constraints for collapsing edges, a predicate $\mathcal{Q}$ protects edges from collapsing.

$\mathcal{Q}$ is typically specified such that the boundary of the mesh is protected. For boundary protection, only boundary edges along geometrical faces and ridges are collapsed, while other features are protected. This relies on classifying vertices into face, ridge and corner (see fig. 3) using surface normals of surrounding boundary triangles. We extract boundary vertices and triangles from the input mesh using the massively parallel boundary extraction of Stroeter

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.

et al. [SHK*23], which uses the orientation of the tetrahedra to obtain outwards pointing normals. Our implementation classifies the boundary vertices based on the scalar products of the normalized normals of surrounding boundary triangles. If two normals are colinear, their scalar product evaluates to 1. To keep the boundary preservation adaptive, feature classification relies on the thresholds ε_F and ε_R. For a face vertex, all the normals of surrounding boundary triangles do not differ by more than ε_F. The face vertex can be collapsed along the geometrical face. For a ridge vertex, only two adjacent edges are included by boundary triangles with normals differing by more than ε_R. The ridge vertex can be collapsed along these two ridge edges. A boundary vertex that is neither face nor ridge is a corner vertex. Corner vertices cannot be collapsed without altering the boundary respecting ε_F and ε_R. We consistently use $\varepsilon_F = 0.01$ and $\varepsilon_R = 0.1$ in our work. In addition, collapsing an interior edge connecting two distinct boundaries can cause surface artifacts. For this reason, we advice to detect boundary edges performing a parallel pass over triangles. If a triangle lies on the boundary, its three edges can be marked as boundary edges. If an edge contains boundary vertices but is not a boundary edge, we do not collapse this edge.

Each edge is associated with a cost. The cost of an edge is specified by the cost function $\mathcal{C}$ that is implemented in the system and can be parameterized to enable user specified inputs such as target edge length. With the specification of the cost function, one can control how collapse operations are prioritized. For each collapsing iteration, our method evaluates the cost function in parallel over edges, saving the cost values in an allocated buffer. Our collapsing algorithm performs iterations of parallel edge collapsing until no more collapse operations can be found or the number of collapse iterations is lower than a user-specified threshold ε_c. The threshold ε_c enables to prevent the overhead of collapsing iterations, which change the mesh insignificantly due to performing a negligible number of edge collapses.

As collapsing edges depends on the mesh connectivity, our method relies on data structures to lookup mesh connectivity. While our method abstracts from the mesh data structure, we advice to use a data structure designed for tetrahedral meshes such as OpenVolumeMesh [KBK13] or TCSR mesh [MAS17; MS18]. We use TCSR mesh in our work, because it is optimized for GPUs. Furthermore, the collapsing method marks mesh elements throughout the procedure. The marking of an element is represented by altering an entry in an array of marker values. Our method supports adaptive coarsening of tetrahedral meshes, because it collapses only edges that are marked for potential removal. A marker value is either 0 or 1. Specifically, our method uses three arrays of marker values:

1. `edgesMarked`: Indicates whether an edge is marked for collapsing "1" or unmarked "0".
2. `verticesMarked`: Indicates whether a vertex remains in the mesh "1" or is removed "0".
3. `tetrahedraMarked`: Indicates whether a tetrahedron remains in the mesh "1" or is removed "0".

In addition, the `vertAffectedByEdge` array indicates whether a vertex belongs to a collapsed edge "edge index" or not "-1". In the following, we describe the steps of one collapsing iteration.

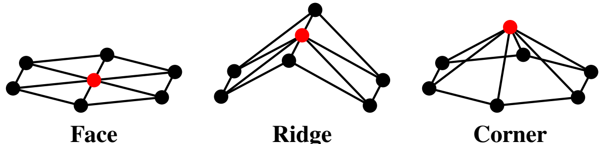

Figure 3: *We classify vertices into types to preserve the boundary.*

3.2. Find Admissible Edges for Collapsing

In order to preserve the consistency of the input tetrahedral mesh, it is mandatory to check if a collapse operation produces an invalid mesh (c.f. section 2.1). A parallel pass over edges filters for admissible edge collapse operations. First of all, filtering checks if the edge is marked for potential collapsing. To preserve the topological type of the mesh an edge collapse is only admissible, if it satisfies the link condition of Dey et al. [DEGN99]. If the link condition is satisfied, the algorithm tentatively performs the edge collapse using the specified placement $\mathcal{P}$. As the edge collapse operation should not produce inverted elements, we evaluate the signed volume *vol* of the resulting tetrahedra using the two one rings of tetrahedra of the edge vertices. For evaluating the volume of the resulting tetrahedra, our algorithm replaces the positions of the vertices belonging to the collapsed edge with the position of the new vertex. If a tetrahedron contains both edge vertices it can be skipped, because the collapse operation removes the tetrahedron. If the volume of any of the resulting tetrahedra is lower than a threshold ε_v the edge collapse operation is not admissible, because it creates inverted or degenerate tetrahedra. We use $\varepsilon_v = 2.e-12$ in our implementation. After the topological and geometrical checks, the algorithm evaluates $\mathcal{Q}$ for the edge and refuses the edge collapse operation in case $\mathcal{Q}$ is not satisfied. In our implementation, $\mathcal{Q}$ protects the boundary of the tetrahedral mesh (c.f. section 3.1). If all of the checks succeed, the edge remains to be marked for collapsing (see fig. 4 (b)). Otherwise, our algorithm unmarks the edge in the `edgesMarked` array.

3.3. Detecting and Handling Conflicts

Our method detects and resolves conflicts in two parallel passes. In a parallel pass over edges, the method first checks, if an edge is marked for potential collapsing, since prior checks potentially unmark edges. For an admissible collapse operation, the method searches for conflicts. The first step of conflict detection is checking adjacent edges. The conflict detection method uses the one ring of adjacent edges for each of the two edge vertices. If any of the adjacent edges is also marked for collapsing and incurs a lower cost, conflict detection prioritizes the edge with the lower cost. In case of equal cost values, we prioritize the edge with the lower index. Due to parallel processing over edges, any thread that unmarks an edge does not need to further check the adjacent edges, because this work is handled by other threads. If none of the adjacent edges incurs a lower cost, the edge remains to be marked for collapsing. As a collapse operation replaces an edge with a vertex, only one of the two edge vertices is removed. Tentatively, we set the entry in `verticesMarked` of the edge vertex with the larger index to 0. The other edge vertex remains and its position is updated later, if the

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.

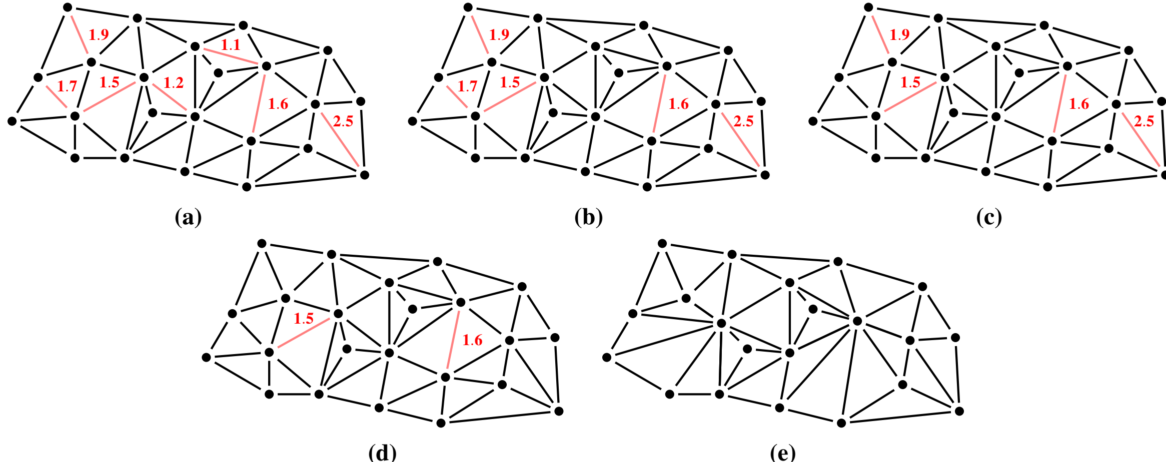

Figure 4: *In (a), red edges with cost values are marked for potential collapsing. The geometrical and topological checks unmark some edges in (b). In (c), conflict detection compares the cost of adjacent edges and prioritizes edges with lower cost. As some simplices are still associated with several collapsed edges, our method ensures that each simplex is affected by only one edge collapse operation in (d). As a result, the two edges can be collapsed simultaneously (see (e)).*

collapse operation is not rejected by the subsequent parallel pass. In addition, conflict detection records for both vertices the index of the collapsed edge by writing the edge index to `vertAffectedByEdge` at the position of the vertex indices. The recorded edge index entries cannot be overwritten by any other thread, because each thread checks the adjacent edges before writing.

As only checking adjacent edges for conflicts is not enough to prevent invalid collapses (see fig. 4 (c)), conflict detection performs a parallel pass over tetrahedra to detect the remaining conflicts. This parallel pass uses the recorded collapse operations in `vertAffectedByEdge` from the previous parallel pass. For each tetrahedron, conflict detection counts how many of the four tetrahedron vertices are marked for removal. Whenever a vertex is marked, we obtain the index of the to-be-collapsed edge from `vertAffectedByEdge` and write it to a local stack. The local stack requires at most four entries. If only one of the four vertices is associated with an edge collapse operation no further checks are required. Otherwise, conflict detection potentially requires to resolve conflicts. In order to resolve conflicts, the method iterates over the edge indices recorded in the local stack. If two edge indices in the stack are different, a conflict is found. In this case, we achieve conflict resolution by evaluating the cost of both edges and prioritizing the edge with the lower cost. If the two conflicting edges share the same cost value, we prioritize the edge with the lower index. The marking for the edge with the larger cost in the `edgesMarked` array is set to 0 and the entries in the `verticesMarked` for the two edge vertices are set to 1, because these vertices remain in the mesh. After the second parallel pass all marked edges can be safely collapsed without producing an invalid mesh (see fig. 4 (d) and (e)).

3.4. Collapse Edges and Construct New Mesh

After conflict detection determined a set of non-conflicting collapse operations, our method builds a new mesh with collapsed edges. As conflict detection already established a valid marking for the `verticesMarked` array, a parallel exclusive prefix scan provides offset positions for vertices and the total number of remaining vertices. Subtracting the number of remaining vertices from the number of input vertices results in the total number of collapse operations. If this number is zero, i.e., no edge is collapsed, or lower than a user-specified threshold ε_c, our algorithm terminates returning the input mesh. Otherwise, we proceed with building the resulting mesh with collapsed edges. Using the offset positions for vertices, the remaining vertices are copied to a newly allocated buffer. The next step is to determine a valid marking for the `tetrahedraMarked` array and collapse the marked edges. In a parallel pass over edges, each thread with a to-be-collapsed edge iterates over the tetrahedra containing the edge and sets their entries in the `tetrahedraMarked` array to 0. Subsequently, the thread compares the indices of the two edge vertices, in order to determine the removed vertex with the lower index and the remaining vertex with the larger index. The thread evaluates the placement strategy $\mathcal{P}$ to obtain the new coordinates for the remaining vertex and writes the coordinates to the newly allocated buffer for the remaining vertices using the offset positions. The offset position of the removed vertex is updated to the offset position of the remaining vertex, in order to prepare for building a valid triangulation.

After collapsing the edges in parallel, we perform an exclusive prefix scan over `tetrahedraMarked` to obtain offset positions and the number of remaining tetrahedra. We allocate an array for the tetrahedra of the resulting mesh. A parallel pass over tetrahedra updates the vertex indices of each marked tetrahedron using the offset positions for vertices. Each updated tetrahedron is copied to the array of remaining tetrahedra using the offset positions for tetrahedra. Finally, the remaining tetrahedra and the array of vertices represent the resulting mesh. Adaptive applications can then use the resulting mesh to select new edges for adaptive mesh coarsening and re-evaluate the cost function.

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.

4. Evaluation

We show that our algorithm for massively parallel collapsing is robust and efficiently exploits parallel processing power. First we apply our algorithm to a multitude of meshes in section 4.1 to validate its robustness and correctness. Subsequently, the evaluation focuses on efficiency. In section 4.2 and section 4.3 we investigate the performance of collapsing edges using the Die [ZJ16] (file ID 128640), Corner Bracket [Uga22] and Atlas Crank [Has20] meshes (see fig. 5). For performance investigations, we select all the edges with a length lower than a predetermined threshold for collapsing. The collapsing of edges terminates, when every edge exhibits a length larger than the threshold or is inadmissible for collapsing. The cost function calculates edge lengths prioritizing smaller edges. We briefly describe three competing algorithms, where sequential collapsing and Gautron et al.'s [GK23] method are implemented using the TCSR mesh data structure as well:

Sequential collapsing: Whenever an edge is admissible for collapsing, it is pushed to a priority queue that prioritizes by edge length. After sequential checking of all the edges, the algorithm pops the topmost edge from the queue until the queue is empty. For every edge, the sequential algorithm first checks if the collapse has been invalidated by a prior collapse. If the collapse has not been invalidated, it is performed. After collapsing the procedure sequentially builds the two arrays of tetrahedra and vertices that represent the collapsed mesh.

Jiang et al.'s [JDH*22] framework: We used the CPU-parallel framework of Jiang et al. [JDH*22] to coarsen tetrahedral meshes accelerated by Intel's Threading Building Blocks. This implementation performs the checks from section 3.2 to filter for admissible collapses preserving the boundary. Their scheduler is set up to prioritize smaller over larger edges and terminates, when no more edge can be collapsed. As Jiang et al.'s [JDH*22] framework ships its own data structure, this implementation does not use TCSR mesh.

Gautron et al.'s [GK23] method: We perform the steps described in section 3 except the conflict detection in section 3.3. Two parallel passes over the edges determine a set of edges that can be collapsed in parallel. The first parallel pass initially checks if an edge was marked as admissible for collapsing. For admissible edges, the parallel pass creates the descriptors of each edge and propagates the descriptors using CUDA's `atomicMin`. The propagation involves the one ring of tetrahedra for each of the two edge vertices. After edge descriptor propagation, another parallel pass over edges once again checks the one ring neighborhood of both edge vertices and marks the edge for collapsing if each adjacent tetrahedron is associated with the edge descriptor of the edge.

Section 4.4 evaluates the performance impact of skipping collapse operations using ε_c. We show that our method can be applied to important use cases. Section 4.5 presents how our method can be used to compress data for direct volume rendering. Section 4.6 shows that our method can be applied for mesh improvement. The evaluation machine is equipped with an Intel i9-11900K CPU and an NVIDIA RTX 3090 GPU. The implementations of the collapsing variants were compiled using Visual Studio 2022 and CUDA.

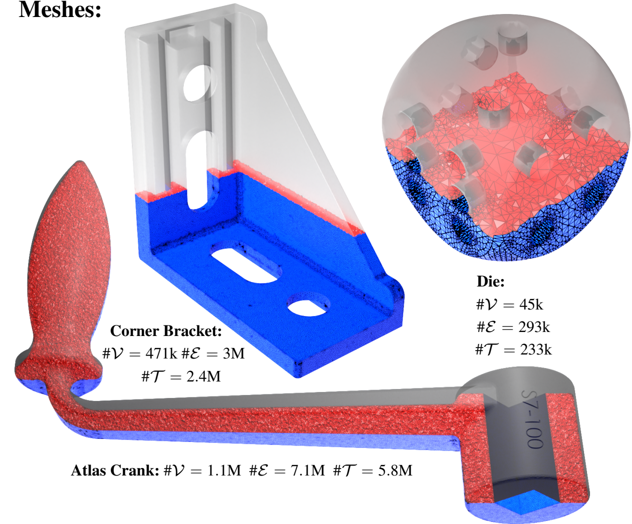

Figure 5: *We show cross sections of the meshes. We provide the numbers of vertices ($\#\mathcal{V}$), edges ($\#\mathcal{E}$) and tetrahedra ($\#\mathcal{T}$).*

Table 1: *Average run times for coarsening the 10*k *meshes of Hu et al. [HSW*20] until convergence ($\varepsilon_c = 0$).*

Method	Run time (s)
Ours	0.180
Gautron et al. [GK23]	0.202
Jiang et al. [JDH*22] (16 threads)	0.368
CPU-sequential	1.500

4.1. Robustness

In order to show that our GPU-parallel collapsing method produces valid meshes, we applied our method to all the 10k meshes generated by Hu et al. [HSW*20]. As a large number of edges should be collapsed to evaluate the robustness of the algorithm, the evaluation procedure uses the median edge length of each mesh as the threshold for collapsing.

Our evaluation procedure performs several consistency checks on the resulting meshes. It includes topological checks. Each triangular face should be part of either one or two tetrahedra. In addition, if a triangular face is shared by two tetrahedra, these tetrahedra include this face in alternating orientations. None of the resulting meshes violates the topological checks. Besides the topological checks, the evaluation procedure includes geometrical checks. The procedure checks for inverted tetrahedra. As duplicated vertices pose a problem to many applications, the procedure also checks for duplicated vertices. In the evaluation procedure, two vertices are duplicates, if their pairwise coordinates differ by less than $1e-13$ on every axis. Our method did not produce duplicated vertices or inverted elements on any of the input meshes. Table 1 shows the average run times for coarsening the 10k meshes with the competing methods. However, the majority of meshes generated by Hu et al. [HSW*20] only includes few elements. Thus, we investigate run time performance on larger meshes.

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.

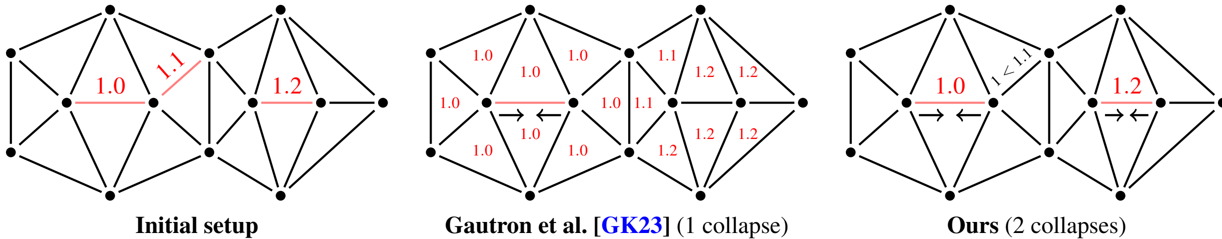

Figure 6: *In the initial setup three edges are marked (shaded red) for collapsing with cost values (red). Gautron et al.'s [GK23] approach does not achieve unique cost value distribution for one of the two independent sub-meshes finding only one collapse operation. Our approach unmarks the edge with cost 1.1 (1 < 1.1) when checking adjacent edges and finds 2 collapse operations.*

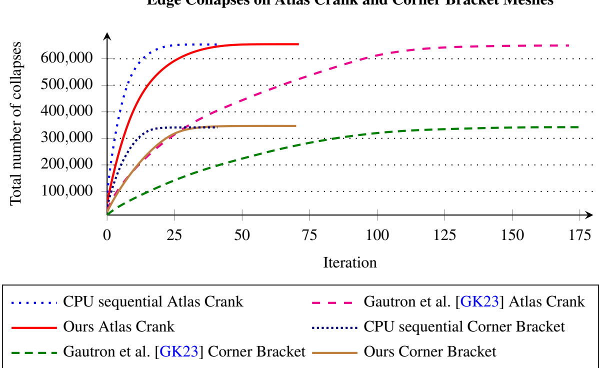

Figure 7: *We plot the total number of collapses per iteration throughout collapsing edges smaller than the median edge length.*

4.2. Conflict Detection

As conflict detection is an essential component of parallel re-meshing, we evaluate our conflict detection method. See fig. 6 for a schematic comparison with Gautron et al.'s [GK23] method. Our conflict detection method benefits from the intermediate step unmarking adjacent edges with larger cost values. This intermediate unmarking significantly reduces the potential conflicts for the second conflict detection step. Thus, the second conflict detection step on the basis of tetrahedra finds a compact set of sub-meshes for re-meshing. The resulting sub-meshes can be locally adjacent, because we find non-overlapping sub-meshes.

Especially for tetrahedral meshes, the ability of our conflict detection to find a more compact set of sub-meshes results in more parallelism, as conflicts frequently occur in the inner structures of the mesh. In order to show that our conflict detection leads to more parallelism, fig. 7 plots the total number of collapsed edges throughout collapsing edges smaller than the median edge length ($\varepsilon_c = 0$) on the Atlas Crank and Corner Bracket meshes for all the three competing collapsing variants. As can be seen in fig. 7, our conflict detection results in significantly fewer collapsing iterations compared to the atomic operation-based propagation of Gautron et al. [GK23]. Due the more compact set of conflict-free sub-meshes, more collapses can be performed in one single iteration. In fact, we observe convergence after up to 59% fewer iterations than using the conflict detection of Gautron et al. [GK23]. Thus, our conflict detection enables efficient exploitation of parallel processing power. As expected, the CPU sequential collapsing variant requires the fewest number of iterations for convergence, because parallel conflict detection tends to reject too many edges for collapsing. Nonetheless, our massively parallel conflict detection achieves high-quality sets of conflict-free sub-meshes. Compared to the sequential variant, our conflict detection results in up to 1.7× more iterations, whereas the method of Gautron et al. [GK23] results in up to 4.2× more iterations. A commonality of all the three collapsing methods is that they perform the bulk of the collapse operations in the initial iterations. After a certain number of iterations the mesh does not change much as only few edges are collapsed. Thus, all of the methods tend to spend a significant workload on performing collapsing iterations without a significant effect on the mesh.

4.3. Run Time Performance

As the speedup depends on the number of collapsed edges, the evaluation performs measurements for different edge length thresholds, interpolating between the minimal edge length and the median edge length. For each measurement, we set $\varepsilon_c = 0$ to collapse edges until no more admissible collapses can be found. As the TCSR mesh data structure requires rebuilding the connectivity relationships for every collapsing iteration, we present overall run time measurements including the rebuilding of connectivity relationships and collapsing run time measurements that only involve the steps of collapsing edges (c.f. section 3.2 to section 3.4) abstracting from the data structure of use. Each of the evaluated methods performs the checks described in section 3.2 to filter for edges admissible for collapsing. Additionally, measurements for massively parallel collapsing perform 20 repetitions and compute the median run time, because run times of parallel computations may exhibit a multimodal distribution [HB15].

We measure run times on the meshes shown in fig. 5 representing different mesh sizes. Figure 8 plots measured run times for edge length thresholds in between the minimal and median edge lengths. Our massively parallel method outperforms the CPU sequential variant by at least one order of magnitude on each of the three meshes. As the Die mesh is the smallest mesh, we achieve the lowest speedups for this mesh. Our method outperforms the framework of Jiang et al. [JDH*22] by up to 18× for the smaller thresholds and by 2.5× for the median edge length. The method

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.

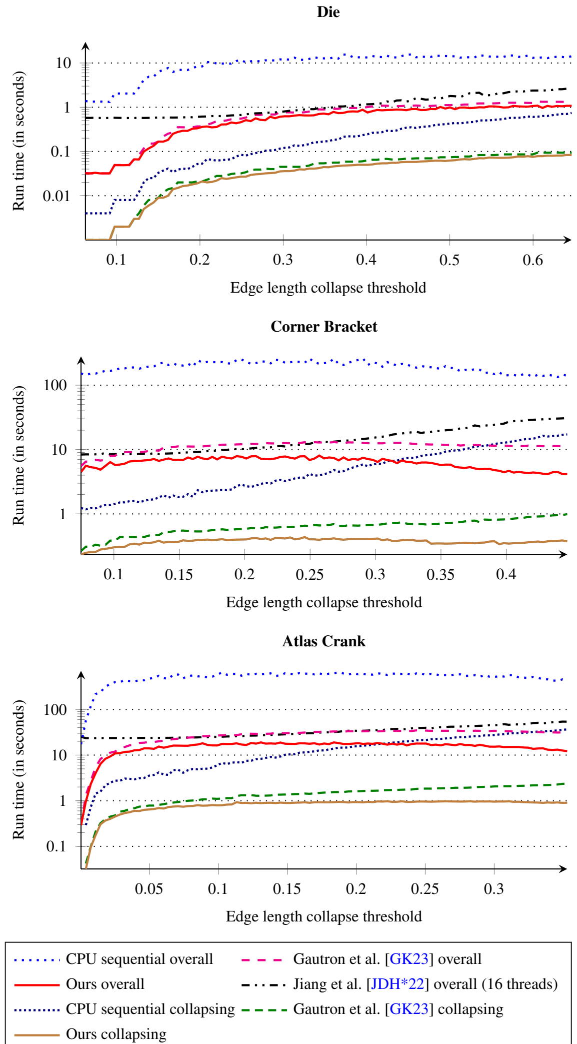

Figure 8: *Run times for parallel edge collapsing. The Y-axis is scaled logarithmically.*

of Gautron et al. [GK23] exhibits only slightly slower run times than our method, because the Die mesh is coarser in the interior and most conflicts occur near the mesh boundary. More compelling speedups can be found on the Corner Bracket and Atlas Crank meshes. For the Corner Bracket mesh, our method outperforms the CPU-sequential method by 33×, the framework of Jiang et al. [JDH*22] by 7.4× and the method of Gautron et al. [GK23] by 2.7×. In addition, our method exhibits better scaling behavior due to improved conflict detection. On the Atlas Crank mesh, our method outperforms CPU-sequential collapsing by 34×, the framework of Jiang et al. [JDH*22] by 4.4× and the GPU-parallel method by Gautron et al. [GK23] by 2.5×. The speedups for the Atlas Crank mesh are slightly lower than for the Corner Bracket mesh, because the Atlas Crank mesh exhibits more thin and curved structures than the Corner Bracket mesh. As a result, our method is significantly faster than the state of the art. An important commonality

Table 2: *Run times and speedups (of our method) for coarsening the Robot mesh [ZJ16] (file ID 255172) until convergence ($\varepsilon_c = 0$).*

Method	Run time (s)	Speedup
Ours	2.201	1.00×
Gautron et al. [GK23]	6.072	2.76×
Jiang et al. [JDH*22] (16 threads)	11.964	5.44×
Jiang et al. [JDH*22] (8 threads)	14.296	6.50×
CPU-sequential	29.488	13.40×

of the implementations of CPU-sequential collapsing, the method of Gautron et al. [GK23] and our method is that the major bottleneck is the rebuilding of connectivity relationships. Thus, a data structure that can be dynamically updated on the GPU would result in significantly improved performance enabling run times close to the run times for collapsing only in fig. 8.

4.4. Skipping Iterations that Collapse only Few Edges

Since many iterations collapse only a small number of edges (c.f. section 4.2), we evaluate the run time of our collapsing algorithm for different choices of the threshold ε_c. In order to aggressively coarsen the mesh and impose a considerable workload, we choose the doubled median edge length as the threshold for collapsing. We use the Robot mesh [ZJ16] (file ID 255172) for this evaluation, because it includes thin as well as large inner structures and flat as well as curved boundaries. For $\varepsilon_c = 0$, Table 2 shows run times and speedups of our method compared to the competing methods. An overview on how the Robot mesh and the run time of our method develops while increasing ε_c appears in fig. 9. As can be seen, the runtime significantly increases for choosing a low number for ε_c. For the jump from $\varepsilon_c = 700$ to $\varepsilon_c = 100$, the run time of our method almost doubles, meaning that many iterations only collapse hundreds of edges. Taking a close look on the boundaries for $\varepsilon_c = 700$ and $\varepsilon_c = 100$, it can be seen that these iterations primarily coarsen mesh regions with many short edges connected to each other. Conflicts frequently occur in these regions limiting the impact of parallel processing. Nonetheless, choosing $\varepsilon_c = 100$ eliminates only 75k more tetrahedra than choosing $\varepsilon_c = 700$. Thus, the impact on coarsening these local regions is not as substantial as coarsening the remainder of the mesh. We achieve a fast run time of 981 milliseconds (below one second) for choosing $\varepsilon_c = 700$, which provides means for immediate response times [New94]. The experiment validates that skipping collapsing iterations performing few collapses comes at low cost, because the bulk of the decimation happens in the initial iterations. For choosing $\varepsilon_c = 1000$, we achieve to halve the number of tetrahedra. For $\varepsilon_c = 700$, we can substantially reduce the number of tetrahedra.

4.5. Coarsening Meshes for Direct Volume Rendering

As coarsening meshes reduces memory requirements and workloads, we show that our method can be used for direct volume rendering (DVR) with little loss of rendering quality. For preserving the rendering quality, the coarsening method loads a scalar field Φ in addition to the tetrahedral mesh. The scalar field Φ includes one value for each vertex. In order to prevent loss of important details,

© 2023 The Authors.

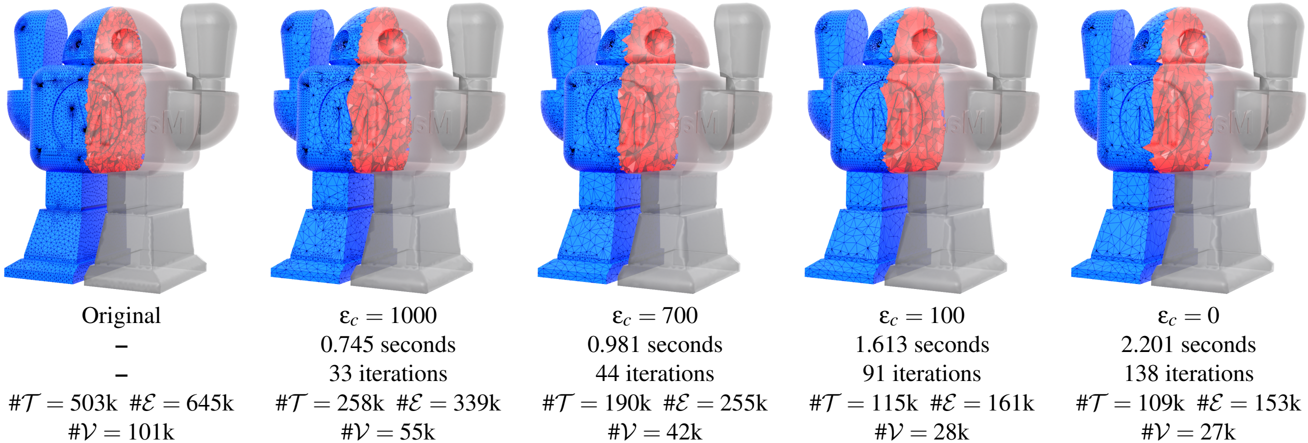

Figure 9: *Coarsening the Robot mesh [ZJ16] (file ID 255172) with different values for ε_c results in different meshes and run times. We provide run times in seconds and numbers of tetrahedra (#$\mathcal{T}$), edges (#$\mathcal{E}$) and vertices (#$\mathcal{V}$) for the resulting meshes.*

the used cost function measures the scalar field error incurred by an edge collapse operation:

$$\mathcal{C}(v_{idx_0}, v_{idx_1}, \Phi) = |\Phi(v_{idx_0}) - \Phi(v_{idx_1})|,$$

where v_{idx_0} and v_{idx_1} are the indices of the two edge vertices.

Collapsing edges produces new vertices with other spatial positions than the prior vertices. Thus, the scalar field needs to be updated. Like Cignoni et al. [CCM*00], we maintain the removed vertices to approximate the scalar values of new vertices. We use the spatial data structure applied for the DVR to interpolate scalar values of newly added vertices from the old scalar field. Our implementation uses the OLBVH data structure [SMSF20] due to its sparse use of memory. In addition, the implementation consistently used the middle point placement strategy to show that the interpolation of the scalar field works well. In order to preserve features, the collapsing of edges should be limited to edges incurring only a low cost. Thus, collapsing is limited to edges with a cost value lower than $0.02(\Phi_{max} - \Phi_{min})$. We demonstrate the quality of the resulting DVR images on the Fusion mesh in fig. 10. The collapsed version of the Fusion model was obtained within 8.6 seconds specifying $\varepsilon_c = 700$.

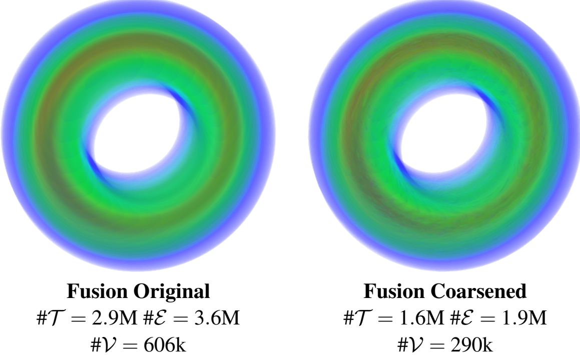

Figure 10: *Though the coarsened Fusion model includes half the vertices of the original only little noise occurs.*

4.6. Collapsing for Mesh Improvement

We specify the cost function $\mathcal{C}$ and the placement strategy $\mathcal{P}$ so that collapsing improves element quality. As the harmonic triangulations by Alexa [Ale19] efficiently improve dihedral angles, we use this concept for an example for mesh improvement. In harmonic triangulations, the goal is to optimize the harmonic index η:

$$\eta(\tau) = \frac{\sum_{a_i \in \tau} a_i^2}{vol_\tau},$$

where τ is a tetrahedron, $a_i, i = 0, \ldots, 3$ its four face areas and vol_τ its volume.

We specify $\mathcal{P}$ to perform a line search finding the optimal position for the vertex replacing the collapsed edge. The line search optimizes the sum of harmonic indices interpolating between the two edge vertices. The cost function $\mathcal{C}$ specifies the improvement in terms of η gained by the collapse operation, prioritizing larger improvement over smaller. If a collapse operation does not lead to an improvement of η or produces inverted elements, it is inadmissible. Like Cutler et al. [CDM04], we couple edge collapse operations with other operations. Our implementation combines GPU-efficient vertex relocation and face/edge swapping [SMWF22] with collapsing edges. One improvement iteration relocates the vertices, finds beneficial face/edge swaps and collapses edges of tetrahedra with a dihedral angle lower than a predetermined threshold. We present an example for our mesh optimization algorithm improving the Die mesh, providing element quality in terms of the minimal dihedral angle $\phi_{\min}$ and conformal AMIPS 3D energy $\mathcal{D}$ [HSW*20]:

$$\mathcal{D}(\tau) = \frac{\mathrm{tr}(\mathbf{J}_\tau^\top \mathbf{J}_\tau)}{\det(\mathbf{J}_\tau)^{2/3}},$$

where τ is a tetrahedron and $\mathbf{J}_\tau$ its Jacobian matrix.

Our algorithm collapsed tetrahedra with a dihedral angle lower

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.

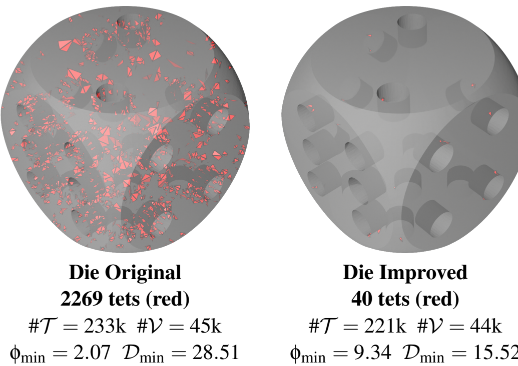

Figure 11: *Elements (red) with a dihedral angle (ϕ) lower than 13° before and after improvement.*

than 13° until convergence ($\varepsilon_c = 0$). Within 273 milliseconds, our optimization procedure performs 6 iterations and returns an improved mesh. As can be seen in fig. 11, the resulting mesh exhibits significantly better element quality and the bulk of low-quality elements is removed. Our method cannot guarantee to remove all tetrahedra with dihedral angles lower than the specified threshold, because not every edge can be collapsed.

5. Conclusion

We have presented a massively parallel algorithm for collapsing edges in an unstructured tetrahedral mesh using efficient conflict detection. This has opened the door for fast re-meshing of unstructured tetrahedral meshes. Our conflict detection produces large sets of conflict-free sub-meshes while prioritizing by edge cost. As a result, our massively parallel method for coarsening tetrahedral meshes achieves a high degree of parallelism. Our method for massively parallel collapsing is robust producing valid meshes on a large test set. Its design enables adaption to specific use cases such as DVR or mesh improvement. Our method does not depend on any specific synchronization primitives such as atomic operations or locking mechanisms. Thus, it can be implemented on any parallel computing architecture, provided a data structure for tetrahedral meshes that allows for the computation of connectivity relationships. The requirement of such data structure is a limitation of our method. Our evaluation showed that the largest performance bottleneck is the massively-parallel rebuilding of connectivity relationships after each collapsing iteration. With the use of a dynamic data structure for GPU-efficient rebuilding of connectivity relationships, our method is expected to achieve even better performance. An inherent limitation of parallel edge collapsing is that the performance-gain through parallelism reduces in regions with many adjacent to-be-collapsed edges, because many iterations are necessary to coarsen these regions. For the sake of good run time performance, coarsening these regions can be skipped by setting a threshold ε_c for the number of collapses. There are many interesting avenues for future work, including:

- Explore our conflict detection for other types of simplicial meshes such as triangular surface meshes.
- Using our massively parallel determination of conflict-free sub-meshes to speedup edge-based re-meshing using the cavity-based operator of Loseille et al. [LM14].
- Parallelization of other re-meshing tasks such as Delaunay boundary recovery of constrained edges [LLGZ14].
- Incorporate our massively parallel collapsing method in mesh adaptation tools to improve performance

Source Code

Relevant code of our conflict detection can be found here.

Acknowledgements

The second author has been supported by the EC project DIGITbrain, No. 952071, H2020. The Fusion mesh is courtesy of the university of Utah. We thank the anonymous reviewers for helping us in improving the quality of our paper.

References

[Ale19] ALEXA, MARC. "Harmonic Triangulations". *ACM Transactions on Graphics* 38.4 (2019), 1–14 9.

[ALSS06] ALAUZET, FRÉDÉRIC, LI, XIANGRONG, SEOL, E. SEEGYOUNG, and SHEPHARD, MARK S. "Parallel Anisotropic 3D Mesh Adaptation by Mesh Modification". *Engineering with Computers* 21.3 (Jan. 2006), 247–258 3.

[CCM*00] CIGNONI, P., COSTANZA, D., MONTANI, C., ROCCHINI, C., and SCOPIGNO, R. "Simplification of Tetrahedral Meshes with Accurate Error Evaluation". *Proceedings Visualization 2000*. IEEE, 2000 2, 9.

[CDM04] CUTLER, B., DORSEY, J., and MCMILLAN, L. "Simplification and Improvement of Tetrahedral Models for Simulation". *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. Association for Computing Machinery (ACM), July 2004 2, 9.

[CM02] CHOPRA, P. and MEYER, J. "TetFusion: An Algorithm for Rapid Tetrahedral Mesh Simplification". *IEEE Visualization, 2002. VIS 2002.* IEEE, 2002 2.

[CRJH09] COMPÈRE, GAËTAN, REMACLE, JEAN-FRANÇOIS, JANSSON, JOHAN, and HOFFMAN, JOHAN. "A Mesh Adaptation Framework for Dealing with Large Deforming Meshes". *International Journal for Numerical Methods in Engineering* 82.7 (Nov. 2009), 843–867 3.

[CTO20] CHEN, ZHENGHAI, TAN, TIOW-SENG, and ONG, HONG-YANG. *On Designing GPU Algorithms with Applications to Mesh Refinement*. 2020. arXiv: `2007.00324 [cs.GR]` 3.

[DBDR16] DEVECI, MEHMET, BOMAN, ERIK G, DEVINE, KAREN D., and RAJAMANICKAM, SIVASANKARAN. "Parallel Graph Coloring for Manycore Architectures". *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, May 2016 3.

[DEGN99] DEY, TAMAL, EDELSBRUNNER, HERBERT, GUHA, SUMANTA, and NEKHAYEV, DMITRY. "Topology Preserving Edge Contraction". *Publications de l'Institut Mathématique* 66 (1999) 2, 4.

[DS99] DE COUGNY, HL and SHEPHARD, MARK S. "Parallel Refinement and Coarsening of Tetrahedral Meshes". *International Journal for Numerical Methods in Engineering* 46.7 (1999), 1101–1125 3.

[DT07] DECORO, CHRISTOPHER and TATARCHUK, NATALYA. "Real-time Mesh Simplification using the GPU". *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. Association for Computing Machinery (ACM), Apr. 2007 3.

[DV13] D'AMATO, J.P. and VÉNERE, M. "A CPU–GPU Framework for Optimizing the Quality of Large Meshes". *Journal of Parallel and Distributed Computing* 73.8 (Aug. 2013), 1127–1134 3.

© 2023 The Authors.

[FJP99] FREITAG, LORI, JONES, MARK, and PLASSMANN, PAUL. "A Parallel Algorithm for Mesh Smoothing". *SIAM Journal on Scientific Computing* 20.6 (1999), 2023–2040 3.

[GH97] GARLAND, MICHAEL and HECKBERT, PAUL S. "Surface Simplification using Quadric Error Metrics". *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '97*. ACM Press, 1997 2.

[GK23] GAUTRON, PASCAL and KUBISCH, CHRISTOPH. *Interactive GPU-based Remeshing of Large Meshes*. NVIDIA GTC Developer Conference. Mar. 2023. URL: https://register.nvidia.com/flow/nvidia/gtcspring2023/attendeeportal/page/sessioncatalog/session/1666622202853001BIHK 3, 6–8.

[Has20] HASTINGS, DAVID. *Atlas Shaper Crank S7-100*. GRABCAD. July 2020. URL: https://grabcad.com/library/atlas-shaper-crank-s7-100-1 6.

[HB15] HOEFLER, TORSTEN and BELLI, ROBERTO. "Scientific Benchmarking of Parallel Computing Systems". *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Association for Computing Machinery ACM, Nov. 2015 7.

[Hop96] HOPPE, HUGUES. "Progressive Meshes". *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. 1996, 99–108 2.

[HSW*20] HU, YIXIN, SCHNEIDER, TESEO, WANG, BOLUN, ZORIN, DENIS, and PANOZZO, DANIELE. "Fast Tetrahedral Meshing in the Wild". *Transactions on Graphics* 39.4 (Aug. 2020) 6, 9.

[Iba22] IBANEZ, DANIEL. *Omega_h*. GitHub. [Online; accessed Apr-2023]. 2022. URL: https://github.com/sandialabs/omega_h 2, 3.

[IBK*17] IBANEZ, DANIEL, BARRAL, NICOLAS, KRAKOS, JOSHUA, LOSEILLE, ADRIEN, MICHAL, TODD, and PARK, MIKE. "First Benchmark of the Unstructured Grid Adaptation Working Group". *Procedia Engineering* 203 (2017), 154–166 3.

[JDH*22] JIANG, ZHONGSHI, DAI, JIACHENG, HU, YIXIN, ZHOU, YUNFAN, DUMAS, JEREMIE, ZHOU, QINGNAN, BAJWA, GURKIRAT SINGH, ZORIN, DENIS, PANOZZO, DANIELE, and SCHNEIDER, TESEO. "Declarative Specification for Unstructured Mesh Editing Algorithms". *ACM Transactions on Graphics* 41.6 (Nov. 2022), 1–14 3, 6–8.

[KBK13] KREMER, MICHAEL, BOMMES, DAVID, and KOBBELT, LEIF. "OpenVolumeMesh – A Versatile Index-Based Data Structure for 3D Polytopal Complexes". *International Meshing Roundtable (IMR)*. Springer Berlin Heidelberg, 2013, 531–548 4.

[KE03] KRAUS, MARTIN and ERTL, THOMAS. "Simplification of Nonconvex Tetrahedral Meshes". *Hierarchical and Geometrical Methods in Scientific Visualization*. Springer. 2003, 185–195 2.

[LLGZ14] LIU, YAN, LO, S.H., GUAN, ZHEN-QUN, and ZHANG, HONG-WU. "Boundary Recovery for 3D Delaunay Triangulation". *Finite Elements in Analysis and Design* 84 (July 2014), 32–43 10.

[LM14] LOSEILLE, ADRIEN and MENIER, VICTORIEN. "Serial and Parallel Mesh Modification Through a Unique Cavity-Based Primitive". *International Meshing Roundtable (IMR)*. Springer International Publishing, 2014, 541–558 2, 3, 10.

[LMA15] LOSEILLE, ADRIEN, MENIER, VICTORIEN, and ALAUZET, FRÉDÉRIC. "Parallel Generation of Large-size Adapted Meshes". *Procedia Engineering* 124 (2015), 57–69 3.

[Lo14] LO, DANIEL S. H. *Finite Element Mesh Generation*. CRC Press, 2014, 1–7 1.

[MAS17] MUELLER-ROEMER, JOHANNES SEBASTIAN, ALTENHOFEN, CHRISTIAN, and STORK, ANDRÉ. "Ternary Sparse Matrix Representation for Volumetric Mesh Subdivision and Processing on GPUs". *Computer Graphics Forum* 36.5 (2017), 59–69 4.

[MBAE09] MISZTAL, MAREK KRZYSZTOF, BÆRENTZEN, JAKOB ANDREAS, ANTON, FRANÇOIS, and ERLEBEN, KENNY. "Tetrahedral Mesh Improvement using Multi-face Retriangulation". *International Meshing Roundtable (IMR)*. Springer Berlin Heidelberg, 2009, 539–555 2.

[MS18] MUELLER-ROEMER, JOHANNES SEBASTIAN and STORK, ANDRÉ. "GPU-based Polynomial Finite Element Matrix Assembly for Simplex Meshes". *Computer Graphics Forum* 37.7 (2018), 443–454 4.

[NE04] NATARAJAN, VIJAY and EDELSBRUNNER, H. "Simplification of Three-dimensional Density Maps". *IEEE Transactions on Visualization and Computer Graphics* 10.5 (Sept. 2004), 587–597 2.

[New94] NEWELL, ALLEN. *Unified Theories of Cognition*. Harvard University Press, 1994. Chap. 8 8.

[NVI23] NVIDIA. *Displacement Micro-Map Toolkit*. [Online; accessed Apr-2023]. Apr. 2023. URL: https://github.com/NVIDIAGameWorks/Displacement-MicroMap-Toolkit 3.

[Par22] PARK, MIKE. *Refine*. GitHub. [Online; accessed Apr-2023]. 2022. URL: https://github.com/nasa/refine 2, 3.

[PP14] PAPAGEORGIOU, ALEXANDROS and PLATIS, NIKOS. "Triangular Mesh Simplification on the GPU". *The Visual Computer* 31.2 (Nov. 2014), 235–244 3.

[RGD22] REED, DANIEL, GANNON, DENNIS, and DONGARRA, JACK. *Reinventing High Performance Computing: Challenges and Opportunities*. 2022 2.

[SG98] STAADT, O.G. and GROSS, M.H. "Progressive Tetrahedralizations". *Proceedings Visualization '98*. IEEE, 1998 2.

[She02] SHEWCHUK, JONATHAN RICHARD. "What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures." *International Meshing Roundtable (IMR)*. 2002, 115–126 2.

[SHK*23] STRÖTER, DANIEL, HALM, ANDREAS, KRISPEL, ULRICH, MUELLER-ROEMER, JOHANNES S., and FELLNER, DIETER W. "Integrating GPU-Accelerated Tetrahedral Mesh Editing and Simulation". *Lecture Notes in Networks and Systems*. Springer International Publishing, 2023, 24–42 4.

[Si20] SI, HANG. *TetGen: A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator (Version 1.6—User's Manual)*. Tech. rep. Berlin: Weierstraß-Institut für Angewandte Analysis und Stochastik, Aug. 2020 2.

[SMSF20] STRÖTER, DANIEL, MUELLER-ROEMER, JOHANNES S, STORK, ANDRÉ, and FELLNER, DIETER W. "OLBVH: Octree Linear Bounding Volume Hierarchy for Volumetric Meshes". *The Visual Computer* 36.10-12 (2020), 2327–2340 9.

[SMWF22] STRÖTER, D, MUELLER-ROEMER, JOHANNES SEBASTIAN, WEBER, DANIEL, and FELLNER, DW. "Fast Harmonic Tetrahedral Mesh Optimization". *The Visual Computer* 38.9-10 (2022), 3419–3433 9.

[Uga22] UGALDE, JOAKIN. *T-Slot 3030 Corner Bracket 60x30*. GRABCAD. Sept. 2022. URL: https://grabcad.com/library/t-slot-3030-corner-bracket-60x30-1 6.

[WBS*12] WEBER, DANIEL, BENDER, JAN, SCHNOES, MARKUS, STORK, ANDRÉ, and FELLNER, DIETER. "Efficient GPU Data Structures and Methods to Solve Sparse Linear Systems in Dynamics Applications". *Computer Graphics Forum* 32.1 (Oct. 2012), 16–26 2.

[WJBK15] WANG, YU, JACOBSON, ALEC, BARBIČ, JERNEJ, and KAVAN, LADISLAV. "Linear Subspace Design for Real-time Shape Deformation". *ACM Transactions on Graphics* 34.4 (July 2015), 1–11 2.

[ZJ16] ZHOU, QINGNAN and JACOBSON, ALEC. *Thingi10K: A Dataset of 10,000 3D-Printing Models*. 2016 1, 6, 8, 9.

[ZWMW23] ZELLMANN, STEFAN, WU, QI, MA, KWAN-LIU, and WALD, INGO. "Memory-Efficient GPU Volume Path Tracing of AMR Data Using the Dual Mesh". (2023) 2.

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.