GPU-accelerated mesh adaptation for structural analysis

M. Stegemann[1,2], J. Mueller-Roemer[1,2], D. Ströter[2], D. Weber[1,2]
[1](*Fraunhofer IGD, Germany);*
[2](*Technical University of Darmstadt, Germany);*

Abstract

Today, engineering processes rely on structural analysis using computer-aided design (CAD). This typically involves discretizing the geometry to apply the finite element method (FEM) solving the partial differential equations (PDEs) of elasticity. The accuracy of the FEM depends on the resolution of the discretization. However, a high resolution typically leads to slower runtime performance, because each element adds to the computational cost. Using a CAD geometry and specified load cases, computing the elasticity PDE requires multiple steps, each of which can become a bottleneck if executed on the CPU. For fast and automated computation, we suggest a GPU-accelerated adaptive simulation pipeline for structural analysis. Due to their capabilities in representing complex geometries and facilitating robust local adaptivity, unstructured tetrahedral meshes are well-suited for mesh adaptation. Since previous work presented fast simulation (Weber et al. [1]), massively parallel optimization and remeshing of unstructured tetrahedral meshes (Ströter et al. [2], [3]), and data structures for massively parallel matrix assembly algorithms (Mueller-Roemer [4]), this work focuses on a-posteriori adaptive mesh refinement of discretized models. This closes a remaining gap towards a fully automated GPU-accelerated adaptive structural analysis for CAD models. Our method achieves a speedup of $2\times$ to $10\times$ compared to the open-source mesh adaptor MMG [5] for tetrahedral meshes. By shifting the bottleneck away from mesh adaptation, the overall computation time of certain structural analysis tasks can be reduced by half. It utilizes the GPU for error estimation and sizing field processing. As a result, the proportion of these steps in the overall runtime is negligible. With heuristic adaptation and little data transfer between CPU and GPU, we achieved fast mesh adaptation to a sizing field. In combination with the fast structural analysis by Weber et al. [1], our pipeline quickly determines structural analysis results close to the so-called mesh-independent solution without laborious manual intervention.

1. Introduction

In structural analysis, we ideally want a “one-click solution” for any given CAD model and any given load case, but this implies many different internal steps. First, most structural analysis methods require a discretized model to

apply the FEM, so a mesh generator is necessary. The FEM numerically computes a solution to the given problem, but this solution is only as accurate as the underlying discretization allows. To obtain an accurate solution, either manual input to the mesh generator or (often partially supervised) a-posteriori adaptive refinement needs to be applied. Because this process is iterated until a convergence criterion is met, multiple time-consuming simulations and mesh adaptations are common. Due to adaptivity to CAD surfaces and mesh generation being out of scope for this work, we focus on situations where a discrete mesh already exists and compare GPU-accelerated adaptive mesh refinement to the well-established open-source mesh adaptor MMG.

2. Background and related work

Adaptive mesh refinement

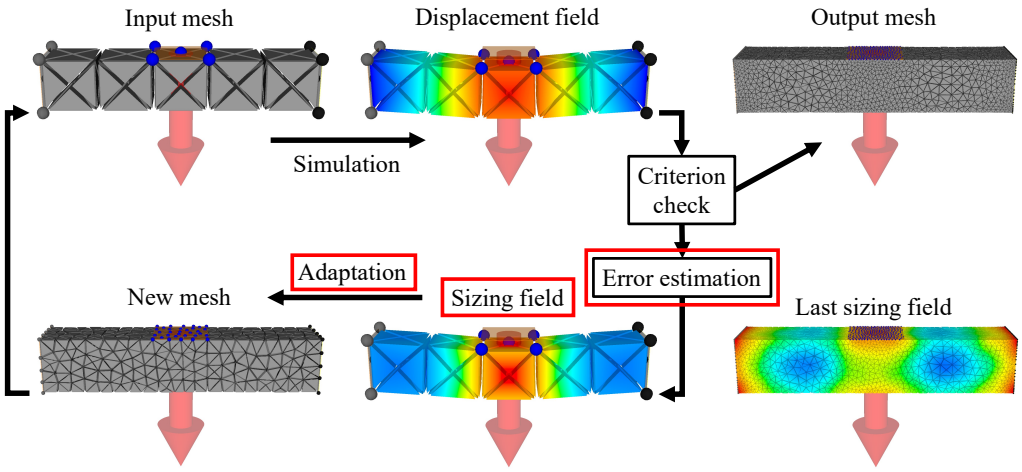

Figure 1: Exemplary pipeline for iterative adaptive mesh refinement. The scope of this work is marked in red.

The typical procedure for a-posteriori adaptive mesh refinement involves several steps. As shown in fig1, the input is usually a relatively coarse mesh to ensure fast simulation times. After the initial simulation, error estimation is used to compute a sizing field which gives us an estimate of the optimal element size for every part of the mesh. To determine which specific elements of the mesh should be subdivided or removed, a multitude of marking strategies for element selection exist. As a last step, splitting edges marked for refinement and collapsing edges marked for removal completes the "Solve Estimate Mark Refine Loop" (see [6]). As mesh quality usually degrades over multiple refinement steps, vertex relocation is applied to optimize element shape.

Error estimation

In three dimensions, structural analysis on a mesh usually results in a discrete displacement field per vertex $u_h(\mathrm{v}_i) \rightarrow \mathbb{R}^3$. This field is used to derive other

physical quantities of interest, e.g., values for stress σ_h and strain ε_h, for all elements in the mesh. Continuous fields such as $u: \mathbb{R}^3 \to \mathbb{R}^3$ need to be derived through interpolation. Among others, there are two widely used error estimation methods to assess the accuracy of these results, one by Kelly et al. [7] and one by Zienkiewicz & Zhu [8]. For our work, we chose the latter one (see [9] and [10] for further information).

Sizing fields

As shown by Lee and Lo [11], a per-element refinement index ρ_h can be derived from error estimation. For every cell in the current mesh, it tells us how many elements should replace it, so that a target error estimation e_0 will be satisfied after refinement. ρ_h can be converted to a sizing field S_h by several means (see [11] for more detail). As the volume of solids in R^3 scale with size cubed, the volume-based method of computing S_h from ρ_h is based on dividing the size of the i-th tetrahedron by $\sqrt[3]{\rho_i}$. This results in a per-element sizing field and averaging over the one-ring neighborhood of any given vertex will yield the sizing field S_h.

In the context of adaptive refinement, the mesh will change after every refinement operation. Due to its high performance on tetrahedral meshes, the octree linear bounding volume hierarchy (OLBVH) acceleration structure by Ströter et al. [12] is used to transfer S_h between meshes of different refinement stages, so S_h only needs to be computed once and stays valid until the adaptation is complete.

Mesh data structure

We use the mesh data structure by Mueller-Roemer and Stork [13], [4], which integrates concepts of Mueller-Roemer et al.'s ternary compressed sparse row (TCSR) volumetric mesh data structure [14] and Zayer et al.'s GPU-adapted structure for unstructured grids [15]. This combination results in a specialized data structure for homogeneous simplicial complexes, such as triangular or tetrahedral meshes. The fundamental representation of the mesh is a set of node positions and an $n \times (d+1)$ array of ordered node indices, where n is the number of triangles/tetrahedra, and d is the dimension of the mesh. Additional element relationships are dynamically computed as needed, using GraphBLAS-inspired [16] approaches and cached in either binary or ternary CSR format. This data structure optimizes parallel access and efficient element neighborhood queries, enabling applications such as GPU-accelerated system matrix assembly [13], [4] and GPU-accelerated mesh optimization [2], [17]. In-place modification is limited to adjustment of node positions; addition or removal of elements requires creating a new mesh. However, GPU-parallel mesh modification generally requires a double-buffered approach, as fine-grained global synchronization on the GPU incurs significant costs.

Simulation

For simulation, we use, but are not limited to using, RISTRA (Rapid Interactive Structural Analysis), which was previously presented by Weber et al. [1]. As RISTRA uses the same mesh data structure for GPU-accelerated system matrix assembly [13], [4] and a highly optimized, GPU-accelerated, iterative solver [18], [19], low-overhead interaction between the adaptive mesher and the simulation is possible.

Vertex relocation

A recent development for a-priori mesh refinement is the harmonic optimization, by Alexa [20]. It is a fast alternative to other r-refinement methods such as the optimal Delaunay triangulation (ODT, see [21]), Laplacian smoothing, or centroidal Voronoi tessellations (CVT). For our work, we use the iterative GPU-parallel modification of the harmonic optimization algorithm presented by Ströter et al. [2]. Coloring is used to obtain sets of vertices where the vertices of each set can be processed in parallel. The locally optimal vertex position is found through gradient descent regarding the harmonic index (see [20]). Brent's method [22] in an inversion free interval ensures fast root finding and proper orientation. To preserve the surface, all gradients of vertices on ridges or planar faces are projected onto the boundary (see [23]).

Subdivision

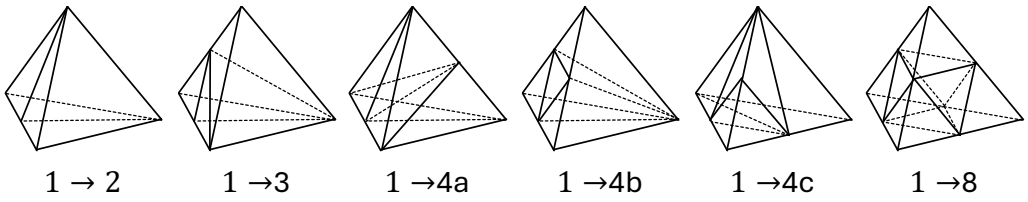

Figure 2: Subdivision is achieved by pattern-based splitting.

On input of the refinement marking, a massively parallel subdivision algorithm refines the marked elements. The subdivision refinement algorithm supports different subdivision patterns to mitigate overrefinement issues. fig2 shows the available subdivision patterns. If none of the supported patterns applies, the algorithm falls back to the regular pattern [24]. To guarantee a consistent triangulation, the algorithm checks edge/face neighbors propagating the subdivision patterns until adjacent subdivision patterns match. As shared faces have to be subdivided consistently during massively parallel application of subdivision patterns, the algorithm applies a global ordering of face vertices for subdividing faces [25].

Coarsening

Collapsing edges in the unstructured tetrahedral mesh eliminates elements to coarsen the local resolution of the mesh. For massively parallel processing, Ströter et al. [3] determine a dense set of conflict-free edges for collapsing. To achieve good results, the edge collapse operations is prioritized by a quality function. The position for the remaining vertex is computed based on the quality function. Edge collapse operations on the surface are only performed if they do not change the shape.

Convergence criteria

Common convergence criteria include metrics such as stress (e.g., von Mises stress), or displacement at specific locations. A possible way to determine when to stop the adaptive refinement process is to use a gradient threshold such as a 3% change between the value on the last and the current mesh. For displacement convergence, this can be fully automated, while for stress convergence, singularities need to be accounted for. In many practical cases, the values for stress or displacement are relevant to solve a derived optimization goal such as determining the required thickness of a steel bar to ensure its maximum displacement remains below 2 mm under a 100 kg load. These cases can be automated through specifically tailored routines which internally rely on the previously mentioned criteria.

3. Approach

Our approach focuses on GPU-parallel methods for error estimation, sizing-field processing and marking strategies. The error estimator by Zienkiewicz and Zhu [8] is local by design, so the per-element refinement index ρ_h as well as the resulting sizing field can be computed in parallel.

The sizing-field is a simple scalar field and therefore, most operations are computed in parallel. The following is a short list of useful operations for sizing fields which we use in our approach. Smoothing is a method to reduce steep gradients. Gradation is used to set a limit for the difference in sizes between neighboring edges (see e.g. [26]) by smoothing the sizing-field of all vertices and their neighbors above a given threshold. Mesh-density control (a-priori) is accomplished by multiplying S_h with a factor for a set of vertices (e.g., on the surface). The number of elements resulting from mesh adaptation to S_h can be estimated by computing ρ_h from S_h and summing over all ρ_h. This is especially useful for targeting a specific overall number of tetrahedra in the refinement process. Edge lengths can be constrained in the output by clamping S_h between a minimum and a maximum value. In cases where singularities are identified (e.g., SPCs with zero displacement after the structural analysis), replacing S_h for these vertices with their average edge length will ignore the estimated sizing field at these locations and prevent overrefinement. Because

singularities lead to high estimated errors in their surroundings, this process is performed gradually for the n-neighborhood.

To assemble all the above into an adaptive refinement procedure, we use strategies to mark specific edges for subdivision or removal. An obvious choice is to split any edge E where the average sizing field for that edge is smaller than $Length(E)/\sqrt{2}$ and remove any tetrahedron where the average sizing field for at least one edge is longer than $Length(E) \times \sqrt{2}$. As subdivision does not ensure the improvement of low-quality tetrahedra and increases the tetrahedron count, we also attempt to remove low-quality elements performing edge collapse operations. To identify which elements should be removed, we use the scale invariant harmonic index η_{SI} (smaller is better, analogous to [20])

$$\eta_{SI} = \frac{1}{|v|}\sqrt{\sum_{i=0}^{3} a_i^3},$$

where v is the tetrahedron volume and a_i is the area of the i-th face. It has a lower bound of $\eta_{SI,min} = 1.5 \times \sqrt[4]{27} \times \sqrt{2} \approx 4.84$ for regular tetrahedra and we consider all tetrahedra with more than $3 \times \eta_{SI,min}$ to be ill-shaped, so all their edges will be marked for collapsing. There are still many other possible strategies left to reduce undesirable patterns in the resulting mesh, but we leave the exploration of these to future research.

Mesh adaptation to a sizing field

After sizing-field computation, we use a heuristic to control the execution order of operations. As we support three operations, harmonic optimization, subdivision, and coarsening, the parameters β_h, β_s, and β_c represent the necessity of the respective operations. An operation is considered significant if the relative change to the number of elements is above a set threshold. If an operation was insignificant, the necessity for that operation will be reduced while the necessity for the other two operations will be increased. If an operation was significant, its own necessity remains unchanged and the necessity for the other two increases. Finding the optimal significance threshold and other parameters for the heuristic is an optimization problem in itself. For details, see Appendix1.

Pipeline

The following contains the proposed pipeline which is a possible implementation of a “Solve Estimate Mark Refine Loop” analogous to [6] where the “Mark” part is spread out into sizing field processing and marking strategies inside the $\boldsymbol{Subdiv}$ and $\boldsymbol{Coarsen}$ functions. It applies adaptive refinement to a given input mesh M with corresponding boundary conditions B until a convergence criterion C is fulfilled.

```
AdaptiveSA(M, B, C, maxIter):
  loop: i = 0, maxIter
    array u_h = RISTRA(M, B)
    array ρ_h = ZZErrorEstimator(M, u_h)
    array S_h = Convert(M, ρ_h)
    S_h = SFProcessing(M, S_h, C)
    if Converged(M, u_h, ρ_h, S_h, C)
      exit loop
    end if
    Adapt(M, S_h, C.α)
  end loop
```

4. Results

All tests were conducted on an i7-14700HX and an Nvidia RTX4060 Laptop GPU with a Samsung (MZVL21T0) NVMe drive. The code was compiled using MSVC (19.41.34120) and NVCC (CUDA 11.8.89). MMG (mmg3d_O3 5.7.0, [5], [27]) is used with "-sol x.sol -i x.mesh -hgrad 1.5 -hausd $0.01 \times diameter(AABB)$" and otherwise default settings. RISTRA uses a block Jacobi preconditioner. The material for models (see fig3) is steel with a Young's modulus of 200 GPa and a Poisson's ratio of 0.3. Gravity is turned off.

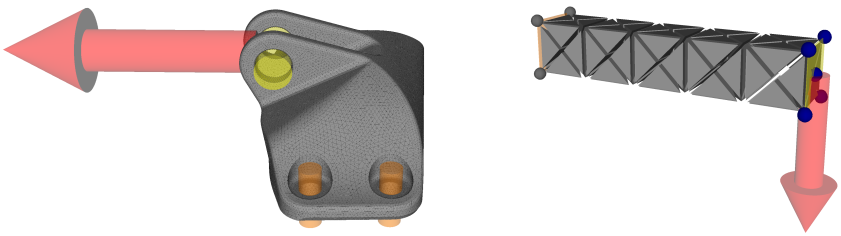

Figure 3: Left: The first model from the SimJEB dataset [28]. SPCs are set through cylinders and the load is $100kN$*. Right: Bar model:* $0.2 \times 0.2 \times 1$ *meters.*

In the following, convergence of maximum displacement is used as termination criterion. A comparison of quality and speed between our mesh adaptation method and MMG is conducted, and simulation, error estimation, and sizing field processing are performed by our pipeline. To account for the time MMG needs to read and write files, we assume that our (non-optimized) functions for file processing take similar time which we can subtract from MMG's runtime to approximate the speed of MMG if it was completely integrated in our pipeline.

A simple example

As an introduction, we show how a-posteriori mesh adaptation can be utilized for structural analysis in the simplest way possible. The bar model from fig3 is loaded with 4×10^4kN of force on one side, while fixed on the other. As seen in fig4, different mesh resolutions lead to different results, quadratic elements (p2) vastly outperform linear elements (p1), and a-posteriori mesh adaptation outperforms regular subdivision.

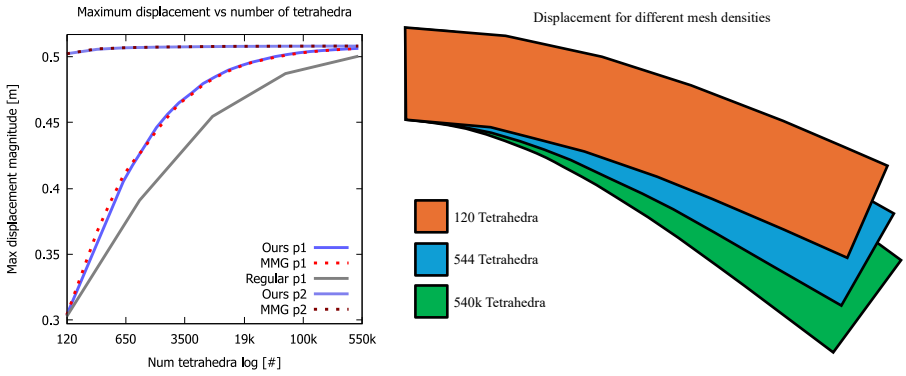

Figure 4: Left: Achieved maximum displacement for $\alpha = 1.00$ and MMG for linear (p1) elements. The quadratic (p2) elements were created through trivial conversion. Right: Illustration of the deformed bar for different numbers of p1 elements.

For a brief runtime comparison with MMG we first need to establish which α results in meshes with similar quality. A four-step a-posteriori mesh adaptation with 350k tetrahedra targeted, provides meshes for which the quality (η_{SI}) is computed for every tetrahedron. For this simple example, the results were averaged over four runs. The results in fig5 show that choosing a conservative $\alpha = 1.66$ leaves a margin of error and we are about 2.8 × faster than MMG.

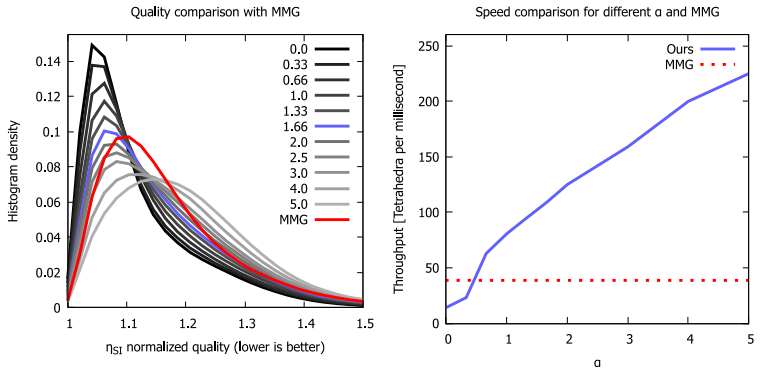

Figure 5: Left: Histogram of tetrahedron quality for different α and MMG. Right: Throughput for different α and MMG. Only the number of resulting tetrahedra from the last step and the accumulated time for every step are considered. Our method runs into the iteration-limit of 250 operations for $\alpha \leq 0.33$.

A more practical example

Leaving the academic example aside, more practical scenarios are found in the SimJEB dataset by Whalen et al. [28]. Due to differences in preserving the original boundary, MMG can remove low-quality elements at the surface, while our method keeps the surface shape mostly unchanged (see fig6).

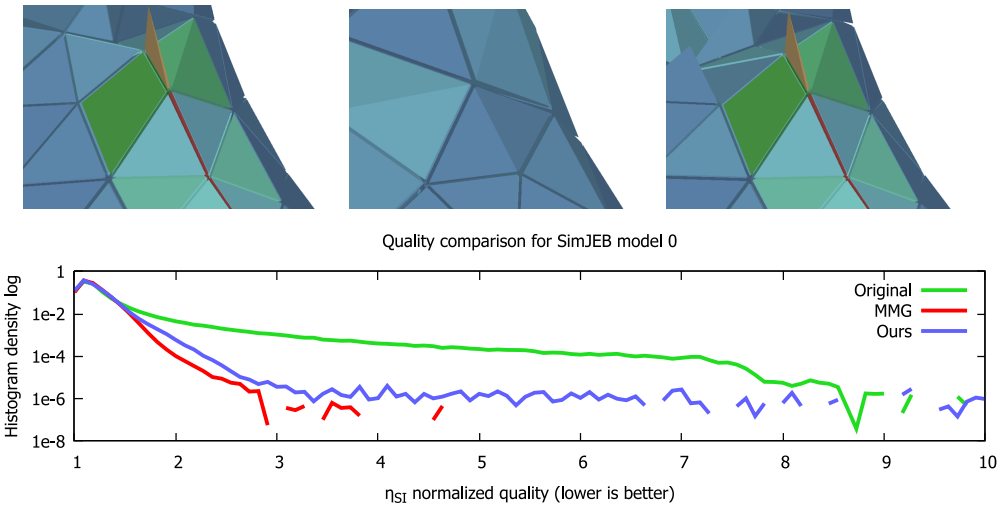

Figure 6: Top: Closeup on a specific area of SimJEB model 0. *Top-Left: Original. Top-Center: Refined by MMG. Top-Right: Refined by our method. Bottom: Quality comparison after mesh adaptation for* $\alpha = 1.33$. *Both methods significantly reduce the number of low-quality elements and MMG removes all tetrahedra with* $\eta_{SI} > 4.7 \times \eta_{SI,min}$.

A three-step a-posteriori adaptive refinement is performed for the first 39 (see Appendix2) models to emulate the process of increasing the mesh resolution by a factor of $\sqrt{2}$ until maximum displacement converges. A limit of 2000k tetrahedra is set throughout sizing field processing.

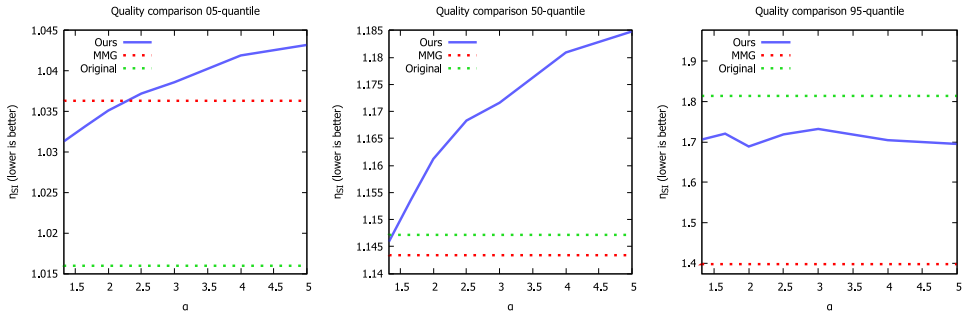

Figure 7: Quality comparison for different quantiles of η_{SI}. *The quantiles are computed for every model after refinement and averaged over all models for each* α.

Executing three iterations of adaptive refinement leads to a consistent comparison because it removes the effect of preexisting a-priori refinement which would lead to different convergence behavior for every model. To account for timing inconsistencies, all benchmarks were performed three times, and the fastest time for each model was used. The quality vs. speed tradeoff also translates to a bigger dataset, as seen in fig7. As most low-quality elements are located at the surface, which is preserved, the 95-quantile of η_{SI} is not affected.

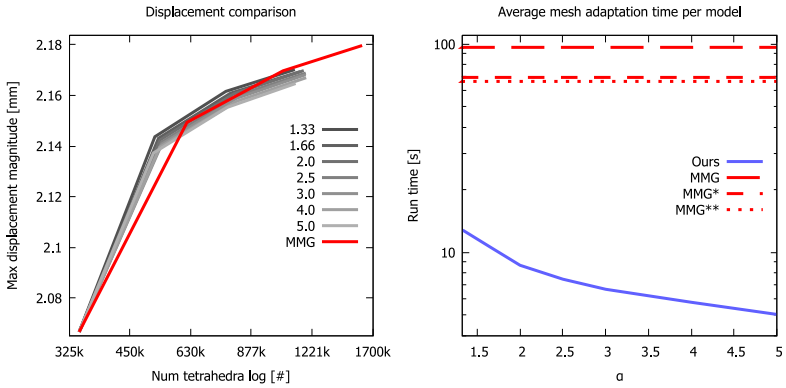

Figure 8: Maximum displacement magnitude averaged over all models for each α and MMG. Right: Average mesh adaptation time for each model. MMG is corrected for the larger number of tetrahedra and MMG** is additionally corrected for IO-time.*

Although the difference is small, we can clearly see that smaller values of α not only increase the overall quality but also lead faster convergence. The runtime comparison is shown on the right side of fig8. MMG produced 39% more tetrahedra than our method, despite using sizing fields with identical adjustments for element count. This is rooted in MMG always refining along a continuous isosurface instead of the discrete mesh surface. In a raw comparison of required runtime, our method would be 8 to 19 times faster, when linearly adjusting the runtime of MMG for the number of tetrahedra and correcting for reading and writing files, this is reduced to 5 to 13 times.

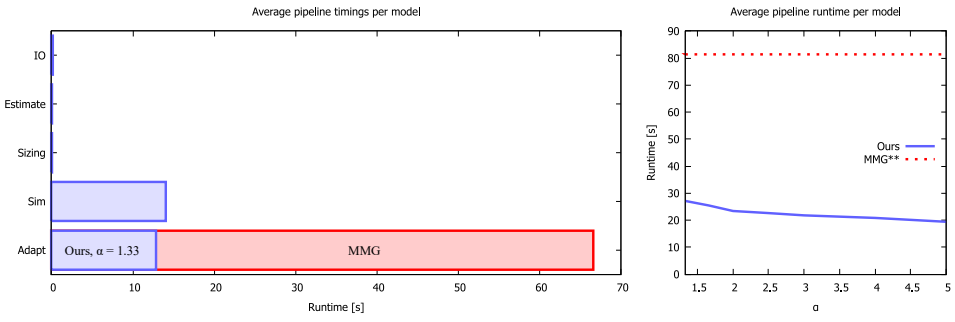

Figure 9: Left: Key timings for different parts of the pipeline if MMG was fully integrated. Right: Average runtime of the overall pipeline for different α and MMG.

By shifting the bottleneck from mesh adaptation to the simulation step, we reduced the time requirements of the proposed pipeline for structural analysis by a factor of 3 to 4 for $\alpha \in [1.33,5]$.

5. Limitations

Despite the advancements, there are several limitations to our current approach. The proposed pipeline runs unsupervised only for the specific use cases we have implemented, such as convergence of maximum displacement, stress (without strong singularities, besides SPCs), sizing field, or error estimation. While parts of the pipeline (such as RISTRA) can process higher-order elements, the mesh adaptation method is limited to using linear tetrahedra. This prevents its applicability to problems which require higher-order elements or other geometries. Lastly, the mesh adaptation method focuses on already discretized models and has no mesh generation or CAD surface approximation capabilities.

6. Conclusion and Future work

In this work, we have shown that mesh adaptation for structural analysis tasks can be accelerated by a factor of approximately 10 times compared to MMG through GPU-parallelization. Our method includes a parameter to adjust the speed-quality tradeoff, so high quality meshes can be generated through the same method without other changes. By shifting the bottleneck away from mesh adaptation in our proposed pipeline, the whole process can be accelerated by a factor of 3. While some remeshing operations that could further improve mesh quality, such as cavity remeshing and specific operations on the mesh surface, are not yet implemented, the current methods provide a solid foundation for future work. The potential for warm starting simulations by transferring displacement fields using OLBVH may enhance the speed of concurrent simulations, shifting the bottleneck back to mesh adaptation, so further improvements on adaptation speed will still be valuable. It is often desirable to use higher-order meshes. Therefore, a promising direction could be to extend massively parallel tetrahedral mesh adaptation to support higher-order elements and operations along CAD-surfaces (or, e.g., p3 approximations of these surfaces, as seen in [29]). Our mesh adaptation tool could also be combined with interactive mesh editing [30] to ensure simulation accuracy after the customization of models.

References

[1] D. Weber, T. Grasser, J. Mueller-Roemer and A. Stork, “Rapid Interactive Structural Analysis,” 2020.

[2] D. Ströter, J. S. Mueller-Roemer, D. Weber and D. W. Fellner, “Fast harmonic tetrahedral mesh optimization,” *The Visual Computer,* vol. 38, p. 3419–3433, June 2022.

[3] D. Stroeter, A. Stork and D. Fellner, “Massively Parallel Adaptive Collapsing of Edges for Unstructured Tetrahedral Meshes,” 2023.

[4] J. S. Mueller-Roemer, “GPU Data Structures and Code Generation for Modeling, Simulation, and Visualization,” TUprints, 2020.

[5] C. Dobrzynski, “MMG3D: User Guide. [Technical Report] RT-0422, INRIA ⟨hal-00681813⟩,” 2012.

[6] R. Stevenson, “Optimality of a Standard Adaptive Finite Element Method,” *Foundations of Computational Mathematics,* vol. 7, p. 245–269, July 2006.

[7] D. W. Kelly, J. P. De S. R. Gago, O. C. Zienkiewicz and I. Babuska, “A posteriori error analysis and adaptive processes in the finite element method: Part I—error analysis,” *International Journal for Numerical Methods in Engineering,* vol. 19, p. 1593–1619, November 1983.

[8] O. C. Zienkiewicz and J. Z. Zhu, “A simple error estimator and adaptive procedure for practical engineerng analysis,” *International Journal for Numerical Methods in Engineering,* vol. 24, p. 337–357, February 1987.

[9] O. C. Zienkiewicz and J. Z. Zhu, “The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique,”

International Journal for Numerical Methods in Engineering, vol. 33, p. 1331–1364, May 1992.

[10] O. C. Zienkiewicz and J. Z. Zhu, “The superconvergent patch recovery and a posteriori error estimates. Part 2: Error estimates and adaptivity,” *International Journal for Numerical Methods in Engineering,* vol. 33, p. 1365–1382, May 1992.

[11] C. K. Lee and S. H. Lo, “Automatic adaptive refinement finite element procedure for 3D stress analysis,” *Finite Elements in Analysis and Design,* vol. 25, p. 135–166, March 1997.

[12] D. Ströter, J. S. Mueller-Roemer, A. Stork and D. W. Fellner, “OLBVH: octree linear bounding volume hierarchy for volumetric meshes,” *The Visual Computer,* vol. 36, p. 2327–2340, July 2020.

[13] J. S. Mueller-Roemer and A. Stork, “GPU-based Polynomial Finite Element Matrix Assembly for Simplex Meshes,” *Computer Graphics Forum,* vol. 37, p. 443–454, October 2018.

[14] J. S. Mueller-Roemer, C. Altenhofen and A. Stork, “Ternary Sparse Matrix Representation for Volumetric Mesh Subdivision and Processing on GPUs,” *Computer Graphics Forum,* vol. 36, p. 59–69, August 2017.

[15] R. Zayer, M. Steinberger and H.-P. Seidel, “A GPU-Adapted Structure for Unstructured Grids,” *Computer Graphics Forum,* vol. 36, p. 495–507, May 2017.

[16] T. Mattson, D. Bader, J. Berry, A. Buluc, J. Dongarra, C. Faloutsos, J. Feo, J. Gilbert, J. Gonzalez, B. Hendrickson, J. Kepner, C. Leiserson, A. Lumsdaine, D. Padua, S. Poole, S. Reinhardt, M. Stonebraker, S. Wallach and A. Yoo, “Standards for graph algorithm primitives,” in *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, 2013.

[17] D. Ströter, “Massively Parallel Editing and Post-Processing of Unstructured Tetrahedral Meshes for Virtual Prototyping,” TUprints, 2024.

[18] D. Weber, J. Bender, M. Schnoes, A. Stork and D. Fellner, “Efficient GPU Data Structures and Methods to Solve Sparse Linear Systems in Dynamics Applications,” *Computer Graphics Forum,* vol. 32, p. 16–26, October 2012.

[19] D. Weber, “Interactive physically based simulation - efficient higher-order elements, multigrid approaches and massively parallel data structures,” 2015.

[20] M. Alexa, “Harmonic triangulations,” *ACM Transactions on Graphics,* vol. 38, p. 1–14, July 2019.

[21] L. Chen and M. Holst, “Efficient mesh optimization schemes based on Optimal Delaunay Triangulations,” *Computer Methods in Applied Mechanics and Engineering,* vol. 200, p. 967–984, February 2011.

[22] W. H. Press, Ed., Numerical recipes, 3. ed. ed., Cambridge [u.a.]: Cambridge University Press, 2007.

[23] D. Ströter, A. Halm, U. Krispel, J. S. Mueller-Roemer and D. W. Fellner, “Integrating GPU-Accelerated Tetrahedral Mesh Editing and Simulation,” in *Simulation and Modeling Methodologies, Technologies and Applications*, Springer International Publishing, 2023, p. 24–42.

[24] J. Bey, “Tetrahedral grid refinement,” *Computing,* vol. 55, p. 355–378, December 1995.

[25] H. L. De Cougny and M. S. Shephard, “Parallel refinement and coarsening of tetrahedral meshes,” *International Journal for Numerical Methods in Engineering,* vol. 46, p. 1101–1125, November 1999.

[26] S. H. Lo, “Generation of high-quality gradation finite element mesh,” *Engineering Fracture Mechanics,* vol. 41, p. 191–202, January 1992.

[27] *MMG - https://github.com/MmgTools/mmg,* 2025.

[28] E. Whalen, A. Beyene and C. Mueller, “SimJEB: Simulated Jet Engine Bracket Dataset,” *Computer Graphics Forum,* vol. 40, p. 9–17, August 2021.

[29] A. Loseille and L. Rochery, “P3 Bézier CAD Surrogates for anisotropic mesh adaptation,” *Computer-Aided Design,* vol. 160, p. 103515, July 2023.

[30] D. Ströter, U. Krispel, J. Mueller-Roemer and D. Fellner, “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.

Appendix

1. Algorithm for mesh adaptation

The following shows a possible implementation of heuristic driven mesh adaptation of M to the sizing-field S_h with speed parameter $\alpha \geq 0$ and the heuristic H. To prevent division through zero, the default float epsilon of 1.2×10^{-7} is added to α. Subdivision and coarsening thresholds are modified to $\sqrt{2.01 + \alpha/10}$. With these settings, $\alpha = 0$ is mostly a brute force approach that runs the algorithm until no further operation can be performed and any $\alpha > 0$ will steer the speed vs. quality tradeoff in the direction of faster termination.

$$H_0 = \begin{pmatrix} H_{0,h} \\ H_{0,s} \\ H_{0,c} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ \beta_h \mathrel{+}= 0.2 & \beta_s \mathrel{-}= 1 & \beta_c \mathrel{+}= 0.33 \\ \beta_h + 0.05 & \beta_s \mathrel{+}= 0.25 & \beta_c \mathrel{-}= 1 \end{pmatrix}$$

$$H_1 = \begin{pmatrix} H_{1,h} \\ H_{1,s} \\ H_{1,c} \end{pmatrix} = \begin{pmatrix} \beta_h \mathrel{-}= 1 & \beta_s = 1 & \beta_c = 1 \\ \beta_h \mathrel{+}= 0.33 & 0 & \beta_c \mathrel{+}= 0.66 \\ \beta_h \mathrel{+}= 0.15 & \beta_s \mathrel{+}= 0.5 & 0 \end{pmatrix}$$

The heuristic is different for significant (H_0) and insignificant (H_0) changes and resets β_s and β_c after each harmonic optimization to 1. The significance is defined as the relative change to the number of tetrahedra through an operation, e.g. if an operation increased the number of tetrahedra by 5%, these 5% are the significance.

1. GPU-accelerated mesh adaptation for structural analysis

```
Adapt(M, S_h, α, H, maxIter):
  var β_h = 1, β_s = 0, β_c = 0
  loop: i = 0, maxIter
    if β_h > 0
      HarmonicOptimization(M, iter: 1)
      Apply H_{1,h} to β_h, β_s, and β_c
    end if
    if β_s > 0
      var significance = Subdiv(M, S_h)
      if √α − 100 × significance/√α > 0
        Apply H_{0,s} to β_h, β_s, and β_c
      else
        Apply H_{1,s} to β_h, β_s, and β_c
      end if
    end if
    if β_c > 0
      var significance = Coarsen(M, S_h)
      if √α − 100 × significance/√α > 0
        Apply H_{0,c} to β_h, β_s, and β_c
      else
        Apply H_{1,c} to β_h, β_s, and β_c
      end if
    end if
    if β_c ≤ 0 And β_s ≤ 0
      exit loop
    end if
  end loop
```

2. SimJEB model numbers

0, 4, 6, 8, 9, 10, 12, 14, 15, 16, 19, 20, 21, 22, 23, 25, 27, 28, 29, 30, 33, 34, 35, 38, 39, 40, 50, 51, 53, 55, 56, 58, 59, 61, 62, 64, 65, 66, 69.