TetFusion: An Algorithm For Rapid Tetrahedral Mesh Simplification

Prashant Chopra and Joerg Meyer*
Mississippi State University Engineering Research Center

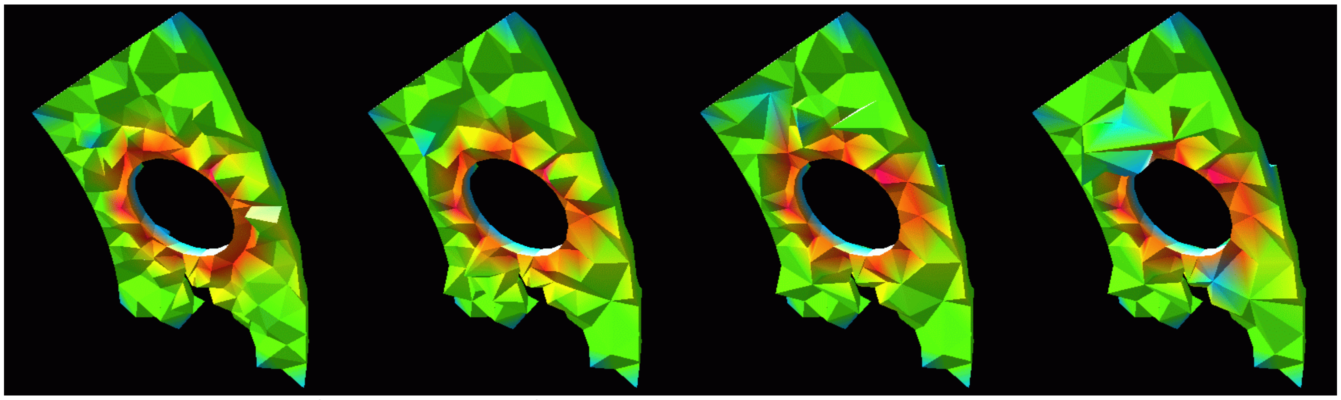


Figure 1: Four *macroLoD*s (defined in section 3.1) of the 12,936 elements spx dataset, created in less than 1 second on an SGI R10000 194MHz, 2048 MB RAM. (a) The original mesh (b) 20.06% reduced (c) 31.20% reduced (d) 38.29% reduced. Only a portion of the mesh (cells that intersect a vertical cutting plane in the XY plane at a specific Z value) is rendered to show the interior elements. (Dataset courtesy: Peter Williams, Lawrence Livermoore National Laboratory).

Abstract

This paper introduces an algorithm for rapid progressive simplification of tetrahedral meshes: *TetFusion*. We describe how a simple geometry decimation operation steers a rapid and controlled progressive simplification of tetrahedral meshes, while also taking care of complex mesh-inconsistency problems. The algorithm features a high decimation ratio per step, and inherently discourages any cases of self-intersection of boundary, element-boundary intersection at concave boundary-regions, and negative volume tetrahedra (flipping). We achieved rigorous reduction ratios of up to 98% for meshes consisting of 827,904 elements in less than 2 minutes, progressing through a series of level-of-details (LoDs) of the mesh in a controlled manner. We describe how the approach supports a balanced re-distribution of space between tetrahedral elements, and explain some useful control parameters that make it faster and more intuitive than 'edge collapse'-based decimation methods for volumetric meshes [3, 18, 20, 21]. Finally, we discuss how this approach can be employed for rapid LoD prototyping of large time-varying datasets as an aid to interactive visualization.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – surfaces and object representations.

Keywords: mesh simplification, multi resolution, level-of-detail, unstructured meshes.

* {prash | jmeyer}@erc.msstate.edu
http://www.cs.msstate.edu/~graphics

1 INTRODUCTION

Large and highly detailed volumetric models are very common now in research environments due to improvements in hardware technology and computational methods. Lately, they have found their place in research areas like computational vector field simulation, finite element analysis, medical imaging, range scanning, and large-scale visualization. Scientists employ a number of techniques like contour detection and iso-contouring, contour connection, iso-surface generation, voxel-based tessellation, etc., to create these models, mostly to simulate a natural or engineering phenomenon. However, since these techniques do not always take the requirements for efficient rendering into account during creation stages, the resulting models often turn out to be very complex in terms of the number of geometric primitives and their spatial relationship. Thus, when these models are broken down to rendering primitives like triangles or quadrilaterals for the purposes of visualization, the large number of primitives poses a challenge to interactive rendering. Each of these primitives might have associated attributes like scalar values, vector tuples, etc., as results of scientific simulations [2, 11, 12], or direct visual attributes like color, transparency, texture, normal vector information, etc. It is well known that such meshes are often inefficient in terms of storage, access, and transmission over computer network media.

To add to these complexities, some scientific and engineering simulations output four- or higher-dimensional (e.g., time-varying) datasets in the form of higher order simplicial complexes (polyhedral meshes), both homogeneous and heterogeneous. The most common examples of homogeneous polyhedral meshes are tetrahedral grids, because of factors like the convex shape of the tetrahedral elements, which makes them suitable for volume rendering. A typical example is an earthquake simulation dataset (11,800,639 vertices, 69,448,288 tetrahedra), which takes about 3.3 gigabytes of storage space just for the basic node geometry and the tetrahedral connectivity information, and an additional 141 megabytes for associated vector information for 120 time-steps in each node. To visualize such large-scale volumetric meshes with time-varying attributes interactively is still an open research arena.

Mesh simplification has been addressed in a number of publications in recent years as a preprocessing step to reduce the geometric complexity of both polygonal and polyhedral meshes. Both refinement- and decimation-based strategies have been explored. Most of these publications extend 'edge collapse'-based decimation methods to volumetric meshes, focusing on accurate error evaluation metrics [3, 21], while also preventing mesh-inconsistencies [3, 18, 21]. These methods work very well as metrics-guided simplification tools. However, the time-complexities of these algorithms (please see section 2.2) discourage their application upon massive datasets of the order of one million or more elements.

This paper describes a rapid tetrahedral mesh simplification framework, which incorporates the constraints on mesh consistency, as well as it binds the geometric and attribute field errors; while also providing better spatial control over LoDs. Specifically, we employ a symmetric reduction operation: *TetFuse*, which suits the natural re-distribution of volume among a reduced number of elements upon every decimation step. We present *TetFusion* as a progressive simplification method to create *macroLoD*s of both static and time varying tetrahedral meshes. Finally, we discuss how this approach can assist in interactive visualization of time-varying volumetric datasets of tetrahedral nature.

2 RELATED WORK

We classify recent publications in the area of mesh simplification broadly into two categories: those based upon polygonal mesh reduction techniques, and those employing polyhedral mesh reduction techniques.

2.1 Polygonal Mesh Simplification

The class of algorithms that simplify polygonal (2-manifold) surface meshes fall into this category. Most of the work discussed here is based on geometry reduction techniques, that eliminate one or more geometric primitives at a time based on specific constraining criteria, and then consolidate the mesh making use of a smaller number of primitives. The area of polygonal mesh simplification has seen some significant improvements lately.

Schroeder et al. [16] propose a simple multi-pass vertex decimation algorithm for triangular meshes. The hole resulting from a vertex-removal is re-triangulated with a smaller number of triangles. Turk [22] describes a surface re-tiling algorithm for simplification, where new vertices are sprayed onto the original mesh following topological and geometry constraints, which are triangulated to yield an approximation of the original mesh.

The work by Kalvin et al. [10] describes a computationally efficient surface approximation algorithm, which identifies quasi-coplanar faces over the mesh in form of patches, which are re-triangulated with a smaller number of primitives. Their greedy face-merging algorithm has a time complexity of O(N), N being the number of polygon faces in the original mesh. Further, this algorithm is more effective in reducing the polygon count in a given polygonal mesh, because of the decimation of multiple primitives per step.

Hugues Hoppe [8] proposes a novel progressive continuous-resolution representation of 2-manifold triangular meshes. He claims that out of the three basic primitive mesh-simplification operations of edge collapse, edge split and edge swap, an 'edge collapse' is all that is needed for effective progressive simplification of meshes. The primitive operation of 'edge collapse' is presented as a completely reversible operation with 'vertex split' as its counter-operation. This work shows how a mesh could be represented as a basic mesh (the simplified version) and a series of 'vertex split' records that can be applied to refine the basic mesh back to the original representation at full resolution. This representation permits geo-morphing and progressive transmission along with significant compression and support for selective refinement.

Schroeder [17] extends his previous work on triangle mesh decimation [16]. He proposes a new algorithm that guarantees a specified level of reduction, but modifies the topology while performing local decimation to achieve the result. He includes two additional primitive operations in the algorithm: vertex split, and vertex merge. Compression ratios as high as 200:1 can be achieved for some models. Finally, Garland and Heckbert [5] suggest a more organized way of representing error metrics in terms of quadrics for efficient calculation of hierarchy for view-dependent simplification using edge collapses.

2.2. Polyhedral Mesh Simplification

Only a few attempts have been made towards polyhedral mesh simplification lately. Both refinement- and decimation- based strategies have been explored in this pursuit. For a detailed note on these strategies, we suggest a reference to the survey report on multi-resolution modeling by M. Garland [4]. This section focuses specifically on decimation strategies because they closely relate to our methods described in the next section.

Trotts et al. [20] extend polygonal geometry reduction techniques for tetrahedral meshes. The authors propose a tetrahedral collapse operation as a sequence of three edge collapses, while keeping the overall error (based on a unique spline-segment representation of each tetrahedron) below a tolerance range. The paper discusses problems and difficulties specific to tetrahedral mesh simplification, and presents a framework to employ edge-collapse to decimate tetrahedra. Because the decimation strategy is based upon successive 'edge collapses', the authors mention the overwhelming overheads for maintaining an edge data structure for massive volumetric datasets. Further, the time complexity of the algorithm presented was not evident in the results or discussion [20].

In their work on progressive tetrahedralization, Staadt and Gross [18] explore a specialized case of progressive simplicial complexes [14]: tetrahedra. Again, the most basic reduction operation proposed is 'edge collapse'. In this manner, the algorithm is very similar to the Progressive Meshes work by Hoppe [8]. Besides preserving the topological and geometric features as boundary, the algorithm elegantly handles previously undiscussed cases of 'negative tetrahedra' (flipping) and tetrahedron-boundary intersections at concave interiors. However, because of the expensive dynamic mesh-consistency tests for these special cases, the time complexity of this algorithm is discouraging for rapid simplification of very large datasets. For example, the algorithm took about 5 hours to simplify a 576,576 elements mesh [18].

Trotts et al. [21] extend their earlier work on Tetrahedral Mesh Simplification [20]. Their aim is to cause minimal error when degenerating the mesh by binding the degeneration error to the deviation of a simplified scalar field from the original field. This way, they claim that the complex energy terms and weights in error evaluation can be eliminated. Further, they revert to a single edge collapse as the atomic decimation operation, unlike their previous work, where they implement a sequence of three edge collapses to result in one 'Tetrahedron collapse' operation. The authors reason that the latter causes complex topological problems and degeneration. This publication reports the simplification time for a range of datasets, which make it evident that a more rigorous, faster, and yet considerate and balanced simplification approach could still be explored. Their best performing algorithm reportedly took 2,557 minutes (about 42 hours) to simplify an about 0.45 million elements Blunt Fin dataset by 83.9%.

Emphasizing on accurate error evaluation techniques, Cignoni et al. [3] present a framework for integrated error evaluation for both domain and field approximations. This paper focuses on accuracy in calculation and prediction of error introduced in the domain of a tetrahedral mesh as a result of 'edge-collapse'-based decimation strategies. Local accumulation, gradient difference, and brute force strategies are explored to predict and evaluate errors while incrementally simplifying a mesh. This paper is a good survey of more accurate error constraint implementations, and offers various options that need to be tested with rigorous decimation strategies like *TetFusion*. Some error metrics are given in the next section.

3 TETFUSION

We present *TetFuse* as a new reversible atomic decimation operation for tetrahedral meshes. The idea is simple and intuitive: take all the four vertices of a tetrahedron, and fuse them onto the barycenter (the geometric center) of the tetrahedron. (Please see Figure 2 for an illustration. A detailed discussion follows in section 3.2.)

3.1 Definitions

Before presenting *TetFuse* as a new decimation operation, we introduce some terms that will be used frequently hereafter in the paper.

3.1 *Prey Tetrahedron*: A tetrahedron that is selected for decimation next.

3.2 *Boundary Tetrahedron*: A tetrahedron one or more of whose vertices lie on the boundary surface. All the tetrahedra that are non-*boundary* shall be called *interior tetrahedra* hereafter in this paper.

3.3 *Boundary Face*: Triangle face of a *boundary tetrahedron* all three of whose vertices lie on the boundary surface.

3.4 *Affected Tetrahedron*: A tetrahedron, which shares exactly one vertex with a *prey tetrahedron*. This shared vertex stretches the affected tetrahedron towards, and onto the barycenter of the *prey tetrahedron* as a result of *TetFusion*.

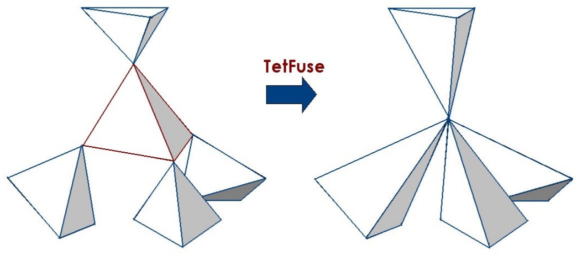


Figure 2: An illustration of the *TetFuse* operation. The (red) center tetrahedron is the one to collapse onto its barycenter (prey tetrahedron). The four other (blue) tetrahedra are the affected ones, which stretch in the direction of the prey tetrahedron's barycenter. Note that for any affected tetrahedron, the vertex it shares with the prey tetrahedron moves 'away' from the base plane formed by its other three vertices (flipping discouraged). Further, at least eleven tetrahedra collapse as a result of *TetFusion* of an interior tetrahedron in a complete mesh. (High decimation per step. Please see section 3.2).

3.5 Prey Vertex: The vertex of an *affected tetrahedron* that it shares with a *prey tetrahedron*.

3.6 *Base triangle*: A triangle formed by the vertices of an affected tetrahedron, excluding the prey vertex.

3.7 *Deleted Tetrahedron*: A tetrahedron, which shares two or more vertices with a *prey tetrahedron*, which collapses as a result of the collapse of the *prey tetrahedron*.

3.8 *Base Normal:* For an *affected tetrahedron*, its *base normal* is the vector from center of its *base triangle* to its barycenter.

3.9 *Normal Stretch Ratio (S_N)*: The ratio of lengths of the *base normal* of an *affected tetrahedron* before and after one instance of *TetFusion* that affects it.

3.10 *Stretch Factor (*STRETCH_FACTOR*)*: The maximum allowed value for *Normal Stretch Ratio (S_N)*, at any instance for an *affected tetrahedron* in the current representation of the mesh.

3.11 *MacroLoD*: A level of representation of a tetrahedral mesh at a macro level, where none of the affected tetrahedra has suffered a *Normal Stretch Ratio* greater than STRETCH_FACTOR in reference of its last m*acroLoD* representation.

3.2 TetFusion: Properties and Effects

We discuss the inherent properties and effects of *TetFuse* as a decimation operation for tetrahedral meshes:

Symmetry: The operation is symmetric in the sense that upon each instance of *TetFuse*, the volume of all the *deleted tetrahedra* is distributed symmetrically amongst *affected tetrahedra* in the local neighborhood.

Rigorous decimation: The operation performs a rigorous decimation. Upon each instance of *TetFuse*, at least 11 tetrahedra are collapsed for a *non-boundary prey tetrahedron*. This includes the prey tetrahedron; at least 4 tetrahedra each sharing one of the four faces with the prey tetrahedron; and at least 6 more tetrahedra each sharing exactly one of the six edges with the prey tetrahedra. This means a 'higher' lower bound on the decimation ratio per step than an 'edge collapse'.

Avoids flipping: Because of symmetry of the decimation operation, the vertex that an affected tetrahedron shares with the prey tetrahedron (*shared vertex*) tends to move away from its base plane (the plane formed by the other three vertices of the affected tetrahedron. please see figure 2). Hence, most of the times the ordering of vertices in an affected tetrahedron does not get changed from its original configuration; and its volume is represented correctly. However, in some cases, flipping is possible. Such special cases, and a solution to deal with them are discussed ahead in section 3.5.

Discourages self-intersections of the boundary: The operation gives a spatial control of the level of detail of a mesh depending upon the immediate neighborhood of a prey tetrahedron, and the STRETCH_FACTOR. It has been verified that self-intersections of boundaries occur only at sharp edges and corners [18], when an *affected tetrahedron* pierces through one or more of the boundary faces of a *boundary tetrahedron*. *TetFusion* gives control over this case that is inherent to the selection of a prey tetrahedron by not allowing any boundary tetrahedron to stretch as a result of collapsing the prey tetrahedron.

Discourages element-boundary intersections at concave boundary regions: Cases of element-boundary intersection occur when an interior tetrahedron stretches through and over a concave boundary region. Such cases cannot be avoided completely, but can be reduced largely by checking the affected tetrahedra in the mesh, i.e., by limiting the expansion of a tetrahedron (limiting the maximum volume of any tetrahedron at any instant), and by not allowing tetrahedra in the vicinity of the boundary surface to stretch as a result of collapsing a prey tetrahedron.

Locks the aspect ratio: If the relative edge-aspect ratios of an affected tetrahedron go beyond a pre-specified threshold value as a result of the fusion of a prey tetrahedron, fusion is not allowed.

3.3 Error Metrics

The error tolerance metrics employed in our algorithm are discussed in this section.

Preserving scalar values: We define a scalar tolerance measure, ΔSc_i, for each vertex *i* of a tetrahedron T.

$$\Delta Sc_i = | Scalar_i - Scalar_{barycenter} | / | Scalar_{range} |$$

where $Scalar_{range} = Scalar_{maximum} - Scalar_{minimum}$ for whole scalar domain in the original mesh.

If for each vertex i of a tetrahedron

$$\Delta Sc_i < \text{SCALAR\_TOLERANCE},$$

the tetrahedron is labeled 'scalar_FusionAllowed'.

Error-binding the stretch of affected tetrahedra: We limit the stretch of affected tetrahedra as a result of *TetFusion* of a prey tetrahedron. We use a pre-specified STRETCH_FACTOR as an upper bound on stretching of the *base normal* of affected tetrahedra. As is evident, for each non-boundary affected tetrahedron, exactly three edges get stretched towards the barycenter of the prey tetrahedron. We use the already defined *Normal Stretch Ratio*, S_N, for an affected tetrahedron to see if the associated prey tetrahedron can be fused:

For all the affected tetrahedra T_a associated with a prey tetrahedron T_p:

If $S_N (T_a) <$ STRETCH_ FACTOR

T_p is labeled 'stretch_FusionAllowed'.

3.4 Boundary Preservation

The current version of *TetFusion* does not affect (stretch) any of the *boundary tetrahedra* T_b. This is accomplished in the following manner:

- No *boundary tetrahedron* T_b is selected as a prey tetrahedron for *TetFusion*.
- No *interior tetrahedron* T_i that affects a boundary tetrahedron is selected as a prey tetrahedron for *TetFusion*. A tetrahedron T_i that is an interior tetrahedron and does not affect (stretch) any boundary tetrahedron (T_b) is labeled 'boundary_FusionAllowed'.

Thus, the current version of *TetFusion* preserves the geometry and topology of the boundary surface perfectly. We believe that an additional simplification pass in the current algorithm to simplify the boundary tetrahedra, while preserving the topology and the sharp features, based upon one of the efficient polygonal mesh simplification methods [5, 8, 15, 16, 18, 20, 21] would further improve the reduction ratios. We plan to extend this work as a future step.

3.5 Flipping: Special Cases

There might still be cases of flipping, which for the case of *TetFusion* would occur only in one way: when the shared vertex of affected tetrahedron moves below the base plane formed by the other three vertices (please see Figure 3). We propose an early-rejection test to avoid such flipping cases. The test is simple:

Given a prey tetrahedron T_p. For each affected tetrahedron T_a that shares exactly one vertex V_s with T_p. If V_s moves to the other side of the base plane P_T upon one instance of *TetFusion* of T_p, the collapse of T_p is not allowed.

If, however, V_s remains on the same side of the base plane P_T upon collapse of T_p, T_p is labeled 'flipping_FusionAllowed'.

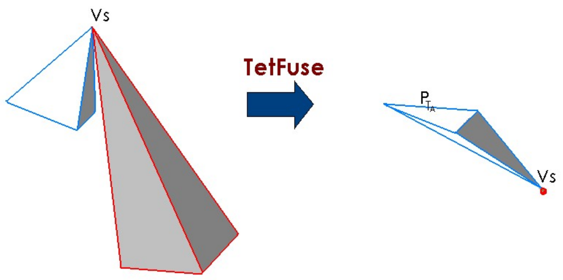


Figure 3: An illustration of a case of flipping because of *TetFusion*. The right (red) tetrahedron is the one to collapse onto its barycenter (prey tetrahedron). The left one (blue) is the affected one. As a result of collapse, V_s moves to the other side of the base plane P_{Ta}. Such cases are expected to be rare because of symmetric re-distribution of space amongst affected tetrahedra because of *TetFusion*. As a result, no two tetrahedra of asymmetric volume distribution can be vertex-adjacent as they are shown in the figure, unless they are present in the original mesh. For symmetrically vertex-adjacent tetrahedra that form a star pattern onto the shared vertex (Figure 2), the shared vertex always tends to remain on the same side of its respective base planes in reference of each of the affected tetrahedra.

3.6 A Locally Greedy Algorithm

We present *TetFusion* as a locally greedy algorithm that makes incremental passes over the whole mesh, examining each tetrahedron in turn for fusion under allowable error tolerance. An outline of the algorithm is as follows:

Select starting values for control parameters, namely SCALAR_TOLERANCE, STRETCH_ FACTOR, etc. (These starting values of the control parameters determine the resolution of the first *macroLoD* of the mesh.)

Next, for current values of SCALAR_TOLERANCE and STRETCH_ FACTOR:

(1) Label all tetrahedra as *non-affected*. Start from the first tetrahedron in the mesh:

(2) Look for the first *interior* and *non-affected* tetrahedron, which can be labeled

```
    scalar_FusionAllowed
        AND
    stretch_FusionAllowed
        AND
    boundary_FusionAllowed
        AND
    flipping_FusionAllowed
```

(3) If found:
    (3.1) Label it as a *prey tetrahedron*
    (3.2) Apply *TetFuse*
    (3.3) Label all *affected tetrahedra*

(4) Start from *prey tetrahedron.* Repeat steps (2)-(3) until no more fusions allowed.

(5) Increment the *macroLoD* counter of the output mesh by one.

Upon completion of one pass of this algorithm, the output will be a *macroLoD* where no tetrahedron will have an average stretch factor greater than the current STRETCH_ FACTOR, for a given value of SCALAR_TOLERANCE. To obtain multiple *macroLoD*s, multiple passes of the algorithm can be run incrementally over the last *macroLoD* obtained, until no more fusions are allowed.

3.7 Dynamic tests

It is important to note that using our approach, cases of element-boundary intersections cannot be avoided completely, especially when high decimation ratios are required. Thus, elegant intersection tests avoiding such cases [3, 18] can be employed, which might increase the time-complexity of the algorithm. However, because of the rigorous simplification approach employed in *TetFusion* (higher reduction ratio per step), the upper bound on time-performance is still expected to be lower than that of the other 'edge collapse'-based methods.

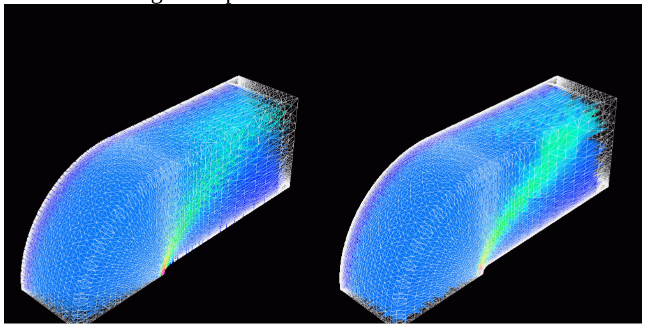


Figure 4: A view of two *macroLoD*s of the 1,499,160 elements blunt-finn dataset. Left: the original mesh, in wire-frame, right: 61.16% reduced, as smooth shaded tetrahedra. Both boundary and scalar attributes preserved.

4 RESULTS

Table 1 summarizes the results obtained from sample runs of the algorithm on a number of datasets. All the execution times are for an SGI R10000 194MHz with 2048 MB RAM, running Irix 6x.

5 CONCLUSION AND FUTURE WORK

We presented a framework for rapid progressive simplification of tetrahedral meshes. The atomic decimation operation employed (*TetFuse*) is symmetric, and better suited for 3D volumetric meshes than 'edge collapse'-based methods. We described how a combination of a few control parameters can provide a smooth and controlled transition through various LoDs of a mesh at a macro level (*macroLoD*s), and discussed how symmetric re-distribution of space and control parameters discourage the cases of self-intersection of boundaries, element-boundary intersections, and negative volume tetrahedra. Additional compression could still be achieved by decimation of external faces of the boundary tetrahedra using 'edge collapse'-based methods [1, 5, 8, 9, 10, 15, 16, 17, 22], or by concise representations of the boundary surfaces [7, 9]. For further offline compression of the datasets, schemes like the ones suggested by Gumhold et al. [6], Pajarola et al. [13], or Szymczak et al. [19] present a nice platform for integration with TetFusion.

At an application level, this framework can be employed in rapid creation of *macroLoD*s for large time-varying datasets, e.g., earthquake simulation. One such dataset, with over 69 million tetrahedra, has characteristic localized scalar and vector attributes per time-step [2, 11, 12]. (The region of interest (ROI) varies per time-step.) Because of the inherent spatial control of decimation, *TetFusion* can be used to rapidly create *macroLoD*s with rigorous decimation in regions outside the ROI. These *macroLoD*s (one per time-step) with high decimation ratios can be suitable for interactive temporal visualization. (The human visual system is understood to be less sensitive to details in a dynamic scene, with moving objects, than in a scene with static objects.) A considerable frame refresh rate (10-13 frames per second) can thus provide sufficient detail to visualize the temporal behavior of such a model. In future work, we plan to employ better error-evaluation techniques [3, 21], and explore the trade-off between time complexity of *TetFusion* and the accuracy of domain and field errors introduced as a result of tetrahedral fusion.

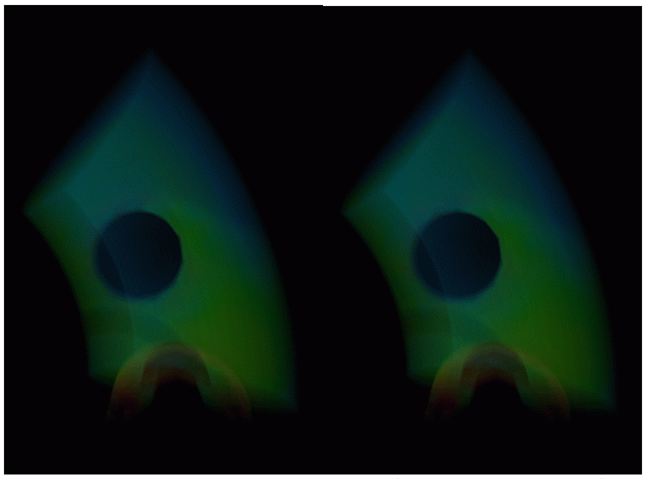


Figure 5: Volume rendered images of two *macroLoD*s of the 827,904 elements spx dataset. Left: the original mesh, right: 69.88% reduced. Both boundary and scalar attributes were preserved. Note that the images are visually indistinguishable.

Acknowledgements

We wish to thank Peter Williams, Lawrence Livermore National Laboratory for *spx* dataset; and Ricardo Farias, Mississippi State University, for providing tetrahedral mesh subdivision code. This work has been developed in part under National Science Foundation award grant no. 6066047–0121989 for the SPUR (Seismic Performance for Urban Regions) project [2, 11, 12]. SPUR Contributors: Gregory L. Fenves, Bozidar Stojadinovic (UC Berkeley), Jacobo Bielak et al. (Carnegie Mellon University), Tomasz Haupt, Joerg Meyer, Purushotham Bangalore, Michael L. Stokes (Mississippi State University).

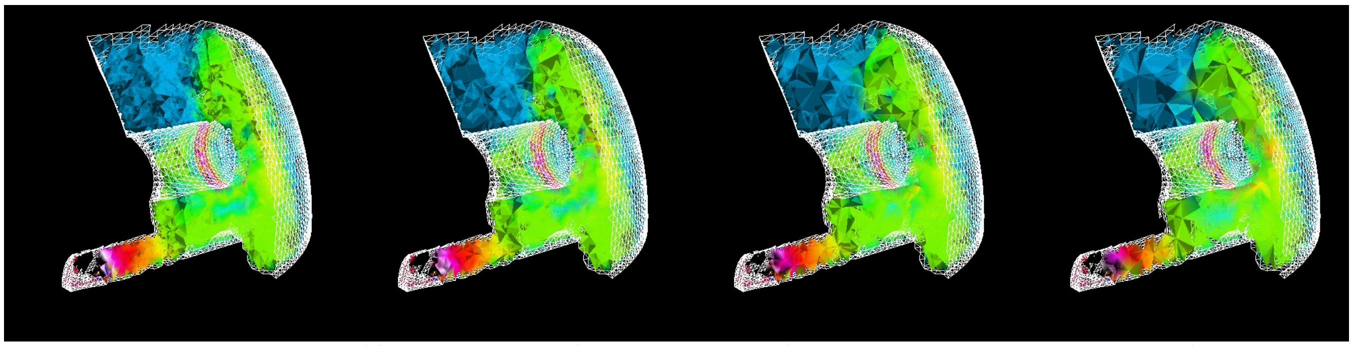
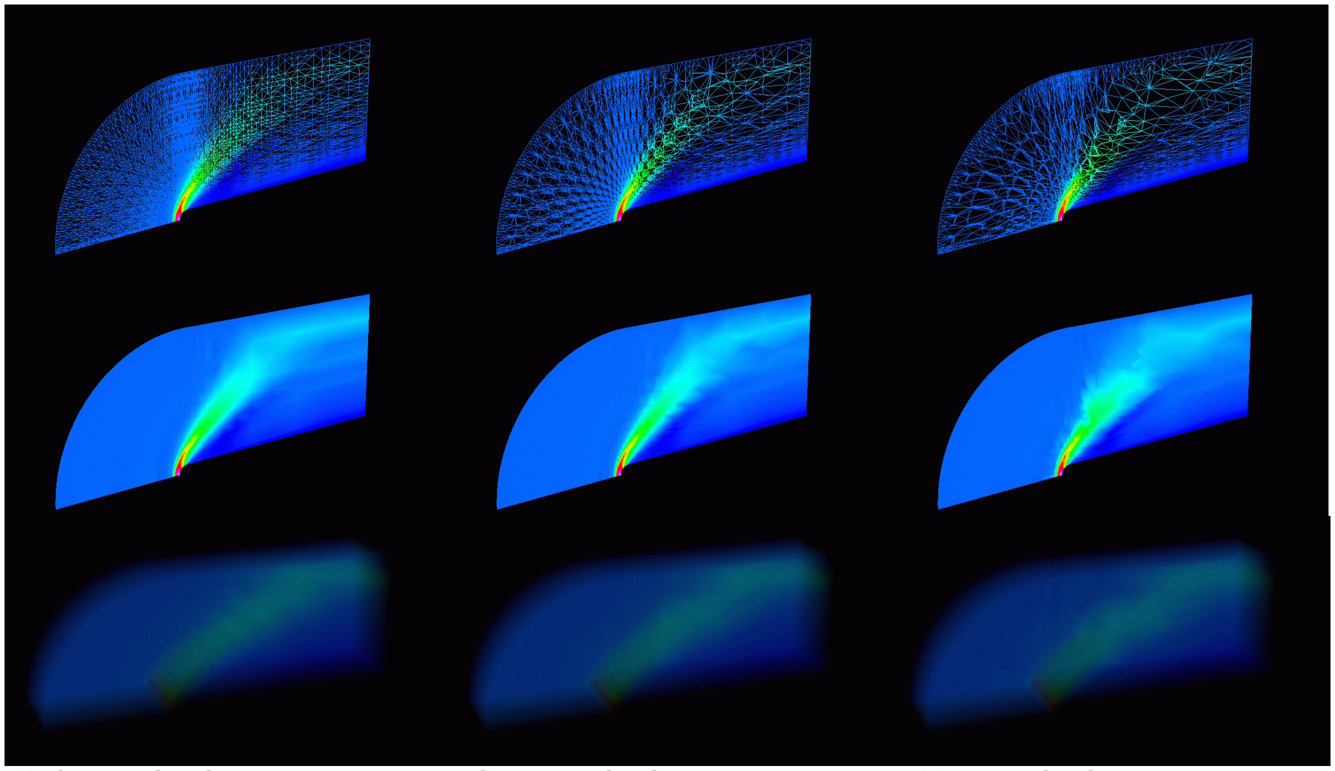


Figure 6: Another 'cut open' view of four *macroLoD*s (defined in section 3.1) of the 103,488 elements tetrahedral mesh of the spx dataset, created in 14.31 seconds on an SGI R10000 194MHz with 2048 MB RAM. (a) The original mesh (b) 43.19% reduced (c) 62.32% reduced (d) 75.5% reduced. Only a portion of the dataset is rendered to show the interior elements. (Dataset courtesy: Peter Williams, Lawrence Livermoore National Laboratory).

S. No.	Number of tetrahedra (% of boundary nodes)	Boundary Preserved	Scalar attribute preserved (SCALAR_TOLERANCE)	STRETCH_FACTOR	% Reduction (Tetrahedra)	CPU User Time (Sec.)
1.	12,936 (6.86%)	YES	YES (0.025)	5.0	41.14%	0.51
2.	78,600 (32.92%)	YES	NO	5.0	50.81%	8.0
3.	100,000 (9.3%)	YES	NO	5.0	50.77%	15.0
4.	103,488 (14.79%)	YES	NO	5.5	78.48%	7.26
5.	103,488 (14.79%)	YES	YES (0.025)	5.5	62.69%	5.85
6.	187,395 (16.5%)	YES	YES (0.025)	5.0	49.39%	15.48
7.	300,000 (9.3%)	YES	NO	5.0	70.39%	47.0
8.	827,904 (13.26%)	YES	YES (0.05)	5.5	83.44%	58.41
9.	827,904 (0%)	NO	NO	5.5	98.93%	91.32
10.	1,005,675 (9.8%)	YES	YES (0.05)	6.0	42.39%	85.47
11.	1,499,160 (9.79%)	YES	YES (0.5)	5.0	63.79%	187.23

Table 1: Summary of results from an implementation of *TetFusion* run on a number of datasets. All the *macroLoDs* have been created on an SGI R10000 194MHz with 2048 MB RAM. Dataset nos. 2, 3, and 7 are subsets of a 69,448,288 elements earthquake simulation model [2, 11, 12]. Spx: dataset nos. 1, 4, 5, 8 and 9 (courtesy Peter Williams, Lawrence Livermoore National Laboratory; and Ricardo Farias, Mississippi State University). Delta-wing: dataset no. 10 (courtesy NASA Ames Research Center). Blunt finn: dataset nos. 6 and 11.



(a) The original mesh (b) 44.7 % reduced (c) 57.9% reduced

Figure 7: Three *macroLoDs* of the 1,499,160 elements blunt-finn dataset. The rows show progressive simplification as a result of *TetFusion*. Each column represents one *macroLoD* of the mesh. Row 1: A cutting plane slicing through the dataset, displaying the primitives of intersection as wireframe. Row 2: Cutting plane displaying the primitives of intersection as filled polygons. Row 3: Volume rendered images of the three *macroLoD*s of the dataset.

References

[1] Alliez, Pierre, and Mathieu Desbrun. Progressive Compression For Lossless Transmission Of Triangle Meshes. In *Proceedings of SIGGRAPH'01 (Los Angeles, California, August 2001),* Computer Graphics Proceedings, Annual Conference Series, pages 198-205. ACM SIGGRAPH, ACM Press. August 2001.

[2] Chopra, P., J. Meyer, and Michael L. Stokes. Immersive Visualization Of A Very Large Scale Seismic Model. In *Sketches and Applications of SIGGRAPH'01 (Los Angeles, California, August 2001),* page 107. ACM SIGGRAPH, ACM Press. August 2001.

[3] Cignoni, P., D. Costanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification Of Tetrahedral Meshes With Accurate Error Evaluation. In Thomas Ertl, Bernd Hamann, and Amitabh Varshney, editors, *Proceedings of IEEE Visualization'00 (Salt Lake City, Utah, October 2000),* pages 85-92. IEEE Computer Society. 2000.

[4] Garland, M. Multi-resolution modeling: Survey & Future Opportunities. In *EUROGRAPHICS'99, State of the Art Report (STAR) (Aire-la-Ville, CH, 1999),* pages 111-131. Eurographics Association. 1999.

[5] Garland, M., and P. Heckbert. Surface Simplification Using Quadric Error Metrics. In *Proceedings of SIGGRAPH'97,* pages 115-122. ACM SIGGRAPH, ACM Press. 1997.

[6] Gumhold, Stefan, Stefan Guthe, and Wolfgang Straβer. Tetrahedral Mesh Compression With The Cut-Border Machine. In *Proceedings of IEEE Visualization'99 (San Francisco, California, October 1999),* pages 51-59. IEEE Computer Society Technical Committee on Computer Graphics, IEEE Computer Society. 1999.

[7] Guskov, I., Kiril Vidimce, Wim Sweldens, and Peter Schroder. Normal Meshes. In *Proceedings of SIGGRAPH'00 (New Orleans, Louisiana, July 2000),* pages 95-102. ACM SIGGRAPH, ACM Press. July 2000.

[8] Hoppe, Hugues. Progressive Meshes. In *Proceedings of SIGGRAPH'96 (New Orleans, Louisiana, August 1996),* pages 99-108. ACM SIGGRAPH, ACM Press. August 1996.

[9] Isenburg, Martin, and Jack Snoeyink. Face Fixer: Compressing Polygon Meshes With Properties. In *Proceedings of SIGGRAPH'00 (New Orleans, Louisiana, July 23-28, 2000),* pages 263-270. ACM SIGGRAPH, ACM Press. 2000.

[10] Kalvin, Alan D., and Russell H. Taylor. Superfaces: Polygonal mesh simplification with bounded error. In *IEEE Computer Graphics and Applications* 16 (3), pages 64-77. 1996.

[11] Meyer, J., and Prashant Chopra. Building Shaker: Earthquake Simulation In A CAVE^{TM}. *Work in progress, IEEE Visualization'01 (San Diego, California, October 2001),* Abstract, page 3. 2001.

[12] Meyer, J., and Prashant Chopra. Strategies For Rendering Large-Scale Tetrahedral Meshes For Earthquake Simulation. *SIAM/GD'01 (Sacramento, CA, November 2001)*, Abstract, page 30. 2001.

[13] Pajarola Renato, Jarek Rossignac, and Andrez Szymczak. Implant Sprays: Compression Of Progressive Tetrahedral Mesh Connectivity. In *Proceedings of IEEE Visualization'99 (San Francisco, California, October 1999),* pages 299-305. IEEE Computer Society Technical Committee on Computer Graphics, IEEE Computer Society. 1999.

[14] Popovic, J., and Hoppe, H. Progressive Simplicial Complexes. In *Proceedings of SIGGRAPH'97,* pages 217-224. ACM SIGGRAPH, ACM Press. 1997.

[15] Renze, K. J., and J. H. Oliver. Generalized Unstructured Decimation. *IEEE Computer Graphics and Applications* 16 (6), pages 24-32. 1996.

[16] Schroeder, William J., Jonathan A. Zarge, and William E Lorensen. Decimation Of Triangle Meshes. *Computer Graphics* 26(2), pages 65-70. 1992.

[17] Schroeder, William J. A Topology Modifying Progressive Decimation Algorithm. In *Proceedings of IEEE Visualization '97,* pages 205-212. 1997.

[18] Staadt, O. G., and M. H. Gross. Progressive Tetrahedralizations. In *Proceedings of IEEE Visualization '98 (October 1998),* pages 397-402. 1998.

[19] Szymczak, Andrzej, and Jarek Rossignac. Grow fold: Compression Of Tetrahedral Meshes. In *Proceedings of the fifth symposium on solid modeling and applications (Ann Arbor, Michigan, June 1999)*, pages 54-64. ACM, ACM Press. 1999.

[20] Trotts, Isaac J., Bernd Hamann, and Kenneth I. Joy. Simplification Of Tetrahedral Meshes. In *Proceedings of Visualization* 98 (October 1998), pages 287-296. 1998.

[21] Trotts, Isaac J., Bernd Hamann, and Kenneth I. Joy. Simplification Of Tetrahedral Meshes With Error Bounds. *IEEE Transactions on Visualization and Computer Graphics* 5 (3), pages 224- 237. 1999.

[22] Turk, Greg. Re-tiling Polygonal Surfaces. *Computer Graphics,* 26(2), pages 55-64. 1992.