Earthquake Visualization Using Large-scale Ground Motion and Structural Response Simulations

Joerg Meyer and Thomas Wischgoll

University of California, Irvine, Department of Electrical Engineering and Computer Science, 644E Engineering Tower, Irvine, CA 92697-2625
{jmeyer|twischgo}@uci.edu

Summary. Earthquakes are not predictable with current technology. However, it is possible to simulate different scenarios by making certain assumptions, such as the location of the epicenter, the type and magnitude of the eruption, and the location of a fault line with respect to buildings of a particular type. The effects of various earthquakes can be studied, and the aftermath of the simulation can be used by first responders and emergency management agencies to better prepare and plan for future disasters.

This article describes methods for visualizing large-scale, finite element simulations of ground motion based on time-varying tetrahedral meshes, and explains how such a simulated earthquake scenario can be visually combined with a simulation of the structural response. The building response is based on a simulation of single-degree-of-freedom (SDOF) structural models.

The amount of data generated in these simulations is quite substantial (greater than 100 gigabytes per scenario). Real-time, interactive visualization and navigation in a 3-D virtual environment is still challenging. For the building simulation, a number of structural prototypes that represent a typical urban building infrastructure is selected. For the ground motion simulation, an efficient, topology-sensitive tetrahedral mesh decimation algorithm suitable for time-varying grids and based on a feature-preserving quadric error metric is used. The algorithms presented in this chapter have the potential for being applied to other scienctific domains where time-varying, tetrahedral meshes are used.

Key words: finite element simulation, tetrahedral mesh simplification, level-of-detail, mesh reduction, hybrid rendering, earthquake simulation, ground motion, structural response, scientific visualization

1 Introduction

According to the National Earthquake Information Center (NEIC), millions of earthquakes occur every year. The majority of events are insignificant microquakes (less than 2.2 on the logarithmic Richter scale), but on average approximately 7,000 earthquakes occur annually with a potential for a significant impact on the infrastructure in an urban setting (Richter magnitude greater than 4.5).

In the United States and in many other countries, the Richter scale is used to measure energy released during an earthquake. On the Richter scale, intensity increases in geometric ratio (Table 1). An increase of one number means the energy released is ten times greater. Thus an earthquake of 4.0 on the Richter scale is ten times stronger than an earthquake of 3.0 on the Richter scale [20]. Significant earthquakes (Richter magnitude greater than 4.5) are characterized by the potential to destroy or to cause considerable damage to buildings, bridges, dams, power and gas lines, etc.

Table 1. Richter scale and potential damage/loss

Richter Scale	*Type of damage in a populated area*
< 2.2	Microquake
2.2	Most people aware that an earthquake has occurred
3.5	Slight damage
4.0	Moderate damage
5.0	Considerable damage
6.0	Severe damage
7.0	Major earthquake, capable of widespread heavy damage
8.0	Great earthquake, capable of total damage/loss

According to the Southern California Earthquake Center (SCEC), a large number of significant earthquakes occurred over the past few decades (Fig. 1), many of them close to known fault lines (Fig. 2).

The data structure for the ground motion simulation is a large-scale, deformable tetrahedral grid. This time-varying grid represents a layered soil model, as it is typical for sedimentary basins, such as the one that we have chosen for our simulation (Fig. 3). The basin is modelled as a three-dimensional isotropic, heterogeneous anelastic medium. The domain is limited by absorbing boundaries that restrict the

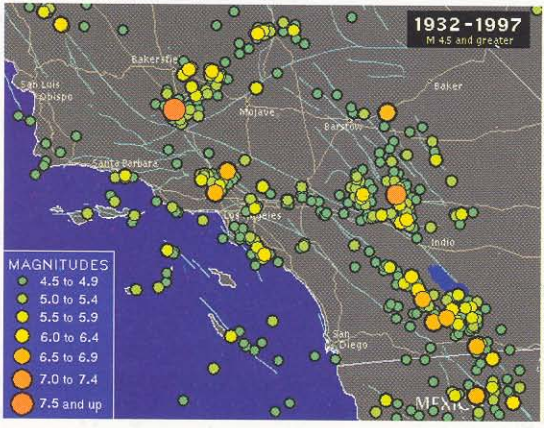


Fig. 1. Earthquake magnitudes greater than 4.5 (1932–1997)

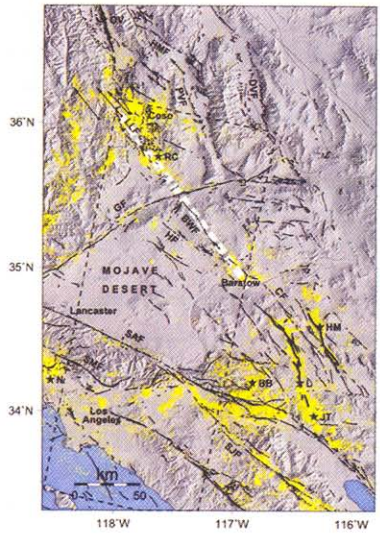


Fig. 2. Southern California earthquake fault lines and locations

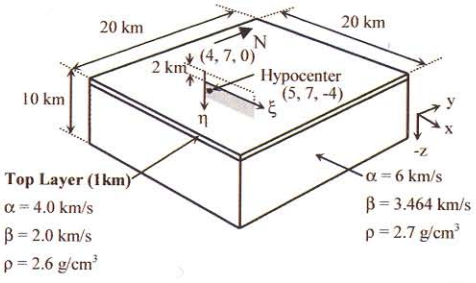


Fig. 3. Layered half-space with extended source fault

amount and magnitude of spurious reflections. Different configurations can be tested by varying the size and the layer composition of the volume represented by the grid.

A finite element simulation is performed over the idealized model shown in Fig. 3. Idealized models are essential to understand physical phenomena such as seismic wave propagation, and to verify calculation methodologies. The model incorporates an idealized extended strike-slip fault aligned with the coordinate system. The shaded area in Fig. 3 represents such a fault.

The tetrahedral mesh generated in this simulation consists of approximately 12 million nodes. For every node, a maximum velocity vector for 800 time steps is computed, representing a time history of the node for a time period of eight seconds in 10 milliseconds increments. The total amount of data generated by this simulation is approximately 130 GB. This is a typical size for a ground motion simulation of the given size and complexity.

This chapter describes a complete, self-contained method for efficient, topology-sensitive time-varying tetrahedral mesh decimation based on a feature- and boundary-preserving quadric error metric. The goal is to visualize such a large-scale dataset interactively, and to enable the user to navigate the space defined by the grid both from an external point of view as well as from a view point within the volume (immersive visualization). This type of visualization enables an interactive analysis of the data, which is essential for an in-depth understanding of the complex processes of ground motion and structural response.

2 Related Work

In earthquake-related ground motion simulations, volumes are typically represented as large-scale, time-varying tetrahedral meshes. The resulting datasets are usually large, and datasets that exceed the size of the processor's main memory are difficult to store, and in the case of remote visualization require significant amounts of time for data transfer over the Internet. The amount of geometry data generated from such simulations is usually too complex for rendering in an interactive display environment. The complexity of those meshes mainly results from the large number of nodes considered in the simulation, from the significant number of time steps necessary to capture various earthquake frequencies, and from the actual vector data (usually displacement or velocity data) associated with the nodes [1, 4, 19].

For interactive visualization, it is critical to reduce the geometric complexity of such large, time-varying meshes with minimal loss of detail in the spatial and in the temporal domain. The algorithm should be scalable, so that an arbitrary number of time steps can be processed and visualized.

In the past, a significant number of algorithms have been developed to address aspects of this complex problem. Tetrahedral meshes have been established as a standard to represent large finite element meshes. Most other data structures, such as voxel grids or scattered data fields can be either broken down or tessellated into tetrahedral meshes. Tetrahedral meshes are preferred for discrete volume representations because of the simplicity of the primitives (tetrahedra).

We distinguish between surface and volume mesh simplification techniques. Most volume decimation algorithms evolved from surface or polygon mesh simplification algorithms [10, 13, 14, 16, 24–27, 32], and most algorithms provide solutions for topological inconsistencies that may occur during or after the mesh simplification. Hierarchical representations are important for interactive rendering of large tetrahedral meshes. Zhou et al. [33], Gerstner et al. [11], and Ohlberger et al. [21] present frameworks for hierarchical representations of tetrahedral meshes. However, none of the described methods are applicable to time-varying data without modifications.

After an initial overview of surface decimation algorithms (section 2.1) and volume decimation algorithms (Sect. 2.2), we discuss a new method called *TetFusion* [2] and some of its properties and limitations (Sect. 2.3). Specific results for this method are presented in Sect. 2.4. Section 3 explains the error metric, and Sect. 4 addresses

time-varying data sets. Section 5 finally gives some results for the new method and explains how it can be used for the given application domain (ground motion and structural response simulation).

2.1 Surface Mesh Simplification

Some ideas that were developed for surface mesh simplification can also be applied to volume meshes. Therefore, we provide a short overview of existing surface mesh simplification methods.

Surface or polygon mesh simplification algorithms are either based on geometry reduction or iterative refinement. Since we are mostly interested in a direct reduction rather than the creation of a new mesh using refinement which usually requires global or local access to the reference mesh making the algorithm less scalable, we focus primarily on the first technique, i.e., geometry reduction. Most methods represent the mesh with a smaller number of nodes that are taken from the original mesh or by calculating new nodes that represent several nodes in the original mesh.

Schroeder et al. [26, 27] propose a method for decimation of triangle meshes. Turk [32] uses a set of new points to re-tile a polygonal mesh. Hoppe [14] takes a different approach and introduces progressive meshes that can be used for streaming data (level-of-detail representation). This method was considered a major milestone in polygon mesh simplification. For further study, Garland [9] provides a comprehensive overview of other mesh simplification techniques.

Garland and Heckbert [10] introduce a quadric metric as an error metric for polygonal mesh simplification. A detailed discussion of the application of this metric to volume meshes is given in Sect. 3.

2.2 Volume Mesh Simplification

Trotts et al. [30] were amongst the first to implement a tetrahedral volume mesh decimation algorithm. In their work, they extend a polygonal geometry reduction technique to tetrahedral meshes. They define a tetrahedral collapse operation as a sequence of three *edge collapses*, while keeping the overall geometric error within a certain tolerance range. The error metric is based on a spline segment representation of the tetrahedra. Since an *edge collapse* is used as the atomic decimation operation, several topological problems, such as volume flipping, are addressed and discussed. The algorithm suffers from overwhelming overhead for storing and updating the connectivity information (edge list).

Staadt and Gross [28] also discuss an extension of the *edge collapse* algorithm for polygonal meshes to tetrahedral meshes. They interpret tetrahedra as a specialized class of simplicial complexes [24] and extend Hoppe's work on progressive meshes [14] to tetrahedral meshes. The article offers solutions to the previously mentioned problem of volume flipping (negative volume), and other topological changes that might occur, such as self-intersection, and tetrahedra intersecting the boundary regions. Kraus et al. [17] use a similar approach and also address the specific case of non-convex tetrahedral meshes.

Trotts et al. [31], in an extension to their earlier work [30], incorporate an error metric that not only incorporates modifications to the geometry, but also to the scalar attributes associated with the vertices. These attributes are usually interpolated between the two nodes that constitute the collapsing edge.

Cignoni et al. [5] use the same idea and apply it to tetrahedral meshes by presenting a framework for integrated error evaluation for both domain and field approximation during simplification. The article elaborately explores local accumulation, gradient difference, and brute force strategies to evaluate and predict domain errors while incrementally simplifying a mesh. The algorithm also uses a quadric error metric, which is compared to other metrics in this article [5].

Topology preservation is the main topic of the work published by Edelsbrunner [8]. He provides an extensive algorithmic background for ensuring topological correctness during *edge-collapse*-based mesh simplification. Dey et al. [7] provide detailed criteria for topological correctness, which can be generalized to polyhedral meshes.

In the next section, we present a computationally efficient tetrahedral volume mesh simplification method that combines metrics for accurate field (attribute) data representation with techniques for restraining volumetric topology.

2.3 A Combined Mesh Decimation Technique: *TetFusion*

Recently, an efficient volume mesh decimation algorithm, *TetFusion*, was published [2]. It addresses both geometry and attribute data preservation. We summarize the properties and limitations of this algorithm, which lead to the development of an improved method, *QTetFusion*, which uses a different error metric and addresses problems such as volume flipping, boundary intersection, and other critical changes in the topology (Sect. 3).

The *TetFusion* algorithm employs a tetrahedral collapse as an atomic operation (*TetFuse*) for mesh decimation. The idea is simple and intuitive: take all four vertices of a tetrahedron, and fuse them onto the barycenter (the geometric center) of the tetrahedron (Fig. 4).

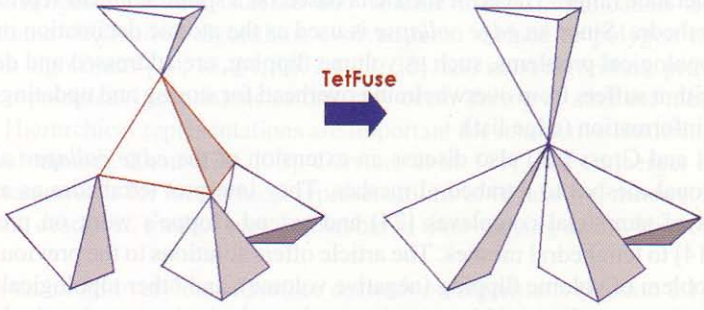


Fig. 4. An illustration of an instance of the *TetFuse* operation

The center tetrahedron is the target object that is supposed to be collapsed onto its barycenter. The four other tetrahedra are affected by this change and stretch in the direction of the target tetrahedron's barycenter. Note that for any affected tetrahedron, the vertex it shares with the target tetrahedron moves *away* from the base plane formed by its other three vertices. In a fully connected mesh, at least eleven tetrahedra collapse as a result of *TetFuse* applied to an interior tetrahedron. This includes the *target tetrahedron*, the four tetrahedra sharing one of the four faces with the *target tetrahedron*, and at least six more tetrahedra that share one of the six edges with the *target tetrahedron*. This means that each instance of an application of *TetFuse* causes an efficient decimation of the mesh.

The *TetFusion* algorithm is based on multiple, error-controlled executions of a primitive operation called *TetFuse*. The following paragraphs summarize the inherent properties and limitations of *TetFusion*.

Symmetry: The volume of the collapsed tetrahedron is distributed symmetrically with respect to the barycenter between the affected tetrahedra in the local neighborhood.

Efficient decimation: Each instance of *TetFuse* causes at least eleven tetrahedra to collapse for a *non-boundary target tetrahedron*, as explained in one of the previous paragraphs. This results in a much higher mesh decimation rate per atomic operation than in the case of an *edge-collapse*-based algorithm.

Avoiding Flipping: Because of the symmetry of the decimation operation, the vertex that an affected tetrahedron shares with the *target tetrahedron* (*shared vertex*) tends to move away from its base plane (the plane formed by the other three vertices of the affected tetrahedron, see Fig. 4). Hence, most of the time the ordering of vertices in an affected tetrahedron does not get changed from the original configuration, and the volume is represented correctly. However, if the barycenter of the *target tetrahedron* is located on the other side of the base plane of an affected tetrahedron, flipping is possible (Fig. 5). Such special cases can be avoided simply by checking

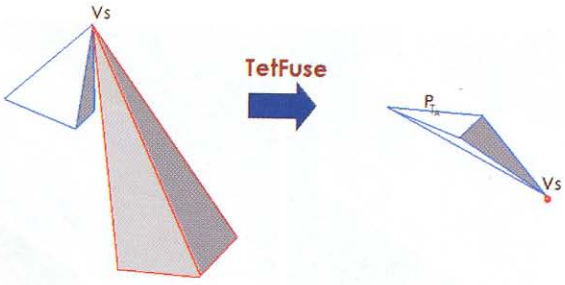


Fig. 5. Flipping may occur if the barycenter is on the other side of the base plane

if the point has moved to the other side of the base plane of the affected tetrahedron, which would result in a rejection of the current instance of *TetFuse*.

Prohibiting Self-Intersections of the Boundary: *TetFusion* does not allow any boundary tetrahedra to be affected. It has been verified that self-intersections of boundaries occur only at sharp edges and corners [28], when an affected tetrahedron pierces through one or more of the boundary faces of a boundary tetrahedron. However, this is a serious limitation of the algorithm that drastically affects the decimation ratio. An improved solution is discussed in Sect. 3.

Prohibiting Boundary Intersections at Concave Boundary Regions: Cases of boundary intersection occur when an interior tetrahedron stretches through and over a concave boundary region. Such cases cannot be avoided completely by employing the given error metric. *TetFusion* addresses this problem by limiting the expansion of an affected tetrahedron, and by not allowing tetrahedra in the vicinity of the boundary surface to stretch as a result of the collapse of a *target tetrahedron*. This is also a limitation, which is address in Sect. 3.

Locking the Aspect Ratio: Tetrahedra that exceed a pre-specified threshold of the edge aspect ratio (long, skinny tetrahedra) are usually difficult to render and therefore trigger an early rejection of the execution of *TetFuse*.

2.4 Example and Results

Figure 6 shows an example of a decimated mesh. The 1,499,160 element blunt-fin dataset was decimated using *TetFusion* in 187.2 seconds, and rendered on an SGI R10000 194MHz with 2048 MB RAM, running Irix 6.52. The boundary is perfectly preserved, while some of the data attribute values appear to be slightly blurred due to repeated interpolation in the reduction step.

In summary, the original *TetFusion* algorithm [2] was limited to interior tetrahedra, thus leaving the surface intact, but at the same time unfortunately also limiting the decimation ratio. The next section discusses an extension of this algorithm,

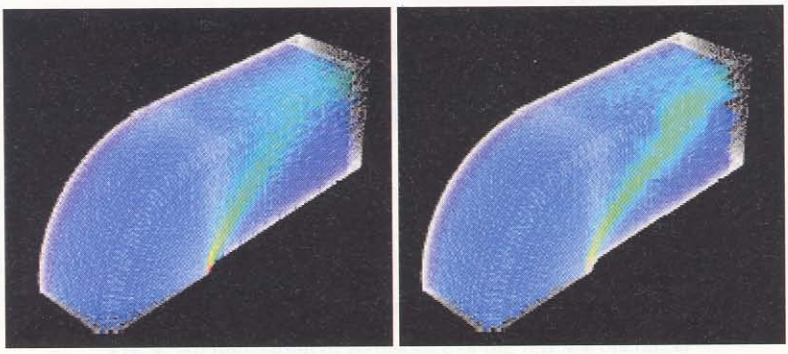


Fig. 6. Original (100%) and decimated mesh (36.21%) of the blunt-fin dataset

called *QTetFusion* (*Q*uadrics-guided *Tet*rahedron *Fus*ion) [3]. It is a volume mesh simplification algorithm that employs a *planar quadric error metric* to guarantee minimum geometric and attribute error upon each instance of *TetFuse*. We show how this atomic tetrahedral collapse operation, along with an efficient geometric error metric, can take care of complex mesh inconsistency problems with minimal additional computational overhead. The algorithm ensures that a mesh stays within (and infinitesimally close to) its *boundary envelope* at all levels of resolution during simplification.

Almost all of the existing work addressing volume mesh simplification evolved from *edge-collapse*-based decimation strategies that were originally proposed for polygonal meshes. However, surface mesh simplification algorithms cannot simply be scaled up to handle higher order simplicial complexes because of additional geometric and topological constraints. Cases like degenerate simplices, violation of Delaunay tessellation, loss of topological genus (undesired closing of holes or creation of new ones), violations of the convex hull and boundary preservation properties, etc., must be specially taken care of during volume mesh simplification. Such cases have already been identified and can be avoided with special computational methods [7, 8]. However, algorithms that present computationally less expensive simplification schemes either do not handle all of these cases, or are spatially selective during decimation and hence are limited in the achieved decimation ratio [2].

3 Tetrahedral Fusion with Quadric Error Metrics

To overcome the shortcomings listed in the previous section, we use *QTetFuse* as a reversible atomic decimation operation for tetrahedral meshes [3]. The algorithm operates on a list of nodes with associated attribute data, and a separate array that contains the connectivity information for every tetrahedron. Both the point list and the list of tetrahedra are updated in the decimation process. This is necessary because both tetrahedra are collapsed and new points are generated by fusing some of the tetrahedra to a single point. For a better understanding of the algorithm, we summarize the most important definitions (Sect. 3.1) that are needed for describing the algorithm (Sect. 3.2), analyze the properties (Sect. 3.3) of the algorithm, and provide some details on the derivation of the employed *planar quadric error metric (PQEM)* from previous polygonal methods (Sect. 3.4) [10]. Finally, we explain how the *fusion point* is calculated (Sect. 3.5), and provide an example of the results of the algorithm (Sect. 3.6).

3.1 Definitions

In this section, we summarize notations and definitions that are necessary for understanding the algorithm.

a) *Target Tetrahedron*: A tetrahedron that is selected for decimation.

b) *Boundary Tetrahedron*: A tetrahedron with one or more of its vertices lying on

the boundary surface. All other tetrahedra that are not on the boundary are called *interior tetrahedra*.

c) *Boundary Face*: Triangle face of a *boundary tetrahedron* where all three vertices lie on the boundary surface.

d) *Fusion Point*: Point of collapse of the four vertices of a *target tetrahedron*. One *target tetrahedron* may have more than one valid fusion point depending on the specified planar quadric error tolerance value.

e) *Affected Tetrahedron*: A tetrahedron that shares exactly one vertex with a *target tetrahedron*. This shared vertex (*target vertex*) stretches the *affected tetrahedron* towards and onto the *fusion point* of the *target tetrahedron* as a result of *QTetFuse*.

f) *Target Vertex*: The vertex of an *affected tetrahedron* that it shares with a *target tetrahedron*.

g) *Base Triangle*: A triangle formed by the vertices of an *affected tetrahedron*, excluding the target vertex.

h) *Deleted Tetrahedron*: A tetrahedron that shares two or more vertices with a *target tetrahedron*, which collapses as a result of the collapse of the *target tetrahedron*.

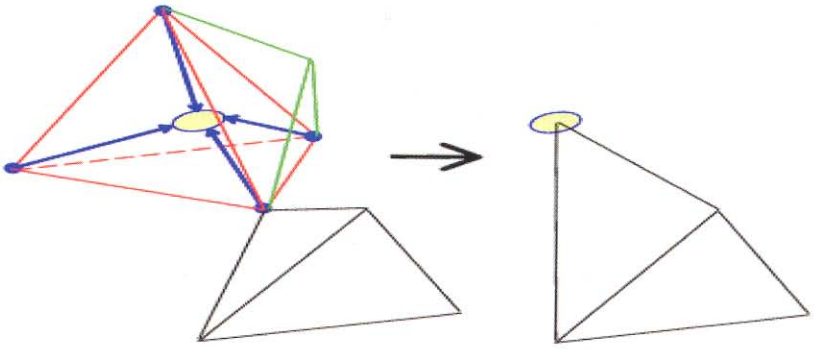
3.2 Algorithm

The basic idea is to fuse the four vertices of a tetrahedron into a point (the *fusion point*, see definitions in Sect. 3.1). The *fusion point* (Fig. 7) is computed so that minimum geometric error is introduced during decimation. We employ a *planar quadric error metric (PQEM)* to measure and restrict this error (see Sect. 3.4).

The algorithm is driven by an efficient space redistribution strategy, handles cases of mesh-inconsistency, while preserving the *boundary envelope* of the mesh, and employs a *PQEM* to guarantee minimum error in the geometric domain when collapsing the tetrahedral elements. It maintains the input mesh's topological genus in the geometry domain at low computational costs.

In Fig. 7, the upper-left *target tetrahedron* collapses onto its *fusion point*. The upper-right tetrahedron degenerates to an edge, which is consequently removed (*deleted tetrahedron*). The lower *affected tetrahedron* stretches in the direction of the corresponding *fusion point*. Note that for the *affected tetrahedron*, the vertex it shares with the *target tetrahedron* tends to move *away* from the base plane formed by its other three vertices. If the shown *target tetrahedron* is an *interior* one, at least eleven tetrahedra collapse as a result of this operation (see Sect. 2.3).

The following section describes the main algorithm. *QTetFusion* is a locally greedy algorithm. It features a pre-processing phase that evaluates *PQEM*s for all the tetrahedra in the input mesh, and stores them in a heap data structure. We employ a *Fibonacci* heap (because of its better amortized time complexity compared to



(a) Before collapse (b) After collapse

Fig. 7. An illustration of one instance of the *QTetFuse* operation

a simple binomial heap) to maintain the priority queue of tetrahedral elements addressed by their *PQEM* keys [3]. In fact, both a binomial and a Fibonacci heap have a worst-case time complexity of $O(m+n \log n)$, where m is the number of edges, and n is the number of nodes. However, in a Fibonacci heap the insert operation is more efficient: O(1) vs. O(log n), while the delete operation remains the same: O(log n). The main algorithm is outlined below:

while heap is not empty
  extract T with minimum $\Delta(T)$ from the heap
  if none of **adjacentTetrahedra**(T) would flip as a result of T's collapse,
    QTetFuse (T)
  update heap

The procedure **adjacentTetrahedra**(T) returns a list of all the tetrahedra adjacent to T. The geometric error $\Delta(T)$ is explained in Sect. 3.4.

3.3 Properties

This section discusses the inherent properties of *QTetFusion* as a volume mesh decimation algorithm for tetrahedral meshes.

Efficient decimation: similar to *TetFusion* (Sect. 2.3).

Avoiding flipping: similar to *TetFusion* (Sect. 2.3).

Simplified mesh restricted to the inside of (and infinitesimally close to) *the boundary envelope of the source mesh*: Self-intersections of boundary elements might occur when an *affected tetrahedron* pierces through one or more of the boundary faces of a *boundary tetrahedron*. We prevent such cases by restricting the simplified mesh to

remain inside its *boundary envelope*.

Avoiding changes of the topological genus of a mesh: The *boundary envelope* of a polyhedral mesh defines its topological genus. Consequently, if the topological genus of the envelope is preserved, topology preservation for the enclosed volume is guaranteed. As a result, the algorithm guarantees that the simplified mesh remains confined to its *boundary envelope*. Therefore, the algorithm cannot change the topology of the mesh, i.e., it is prevented from closing any existing holes or from creating new ones. The latter is an inherent problem of all *edge-collapse*-based decimation algorithms and usually requires complex consistency checking. The proposed method requires only local testing of the *affected* and *deleted tetrahedra* and is therefore relatively efficient.

3.4 Planar Quadric Error Metric (PQEM)

This section describes the error metric we employ to control the domain errors during simplification. Garland and Heckbert [10] developed a computationally efficient and intuitive algorithm employing a *Quadric Error Metric (QEM)* for efficient progressive simplification of polygonal meshes. The algorithm produces high quality approximations and can even handle 2-manifold surface meshes.

To obtain an error minimizing sequence of *QTetFuse* operations, we first need to associate a *cost* of collapse with each tetrahedron in the mesh. As described in [10], we first associate a quadric error measure (a 4×4 symmetric matrix Q) with every vertex v of a tetrahedron that indicates the error that would be introduced if the tetrahedron were to collapse. For each vertex v of a tetrahedron, the measure of its squared distance with respect to all incident triangle faces (planes) is given by:

$$\Delta(v) = \Delta(\begin{bmatrix} v_x & v_y & v_z & 1 \end{bmatrix}^T) = \sum_{p=faces(v)} (a_p p^T v)^2 \tag{1}$$

where $p = \begin{bmatrix} p_x & p_y & p_z & d \end{bmatrix}^T$ represents the equation of a plane incident on v such that the weight a_p represents the area of the triangle defining p. Further, if n represents the normal vector of p, then d is given by

$$d = -nv^T \tag{2}$$

Equation (1) can be rewritten as a quadric:

$$\begin{aligned} \Delta(v) &= \sum_{p=faces(v)} v^T (a_p^2 p p^T) v \\ &= v^T \left(\sum_{p=faces(v)} (a_p^2 p p^T) \right) v \\ &= v^T \left(\sum_{p=faces(v)} (Q_p) \right) v \end{aligned} \tag{3}$$

where Q_p is the area-weighted error quadric for v corresponding to the incident plane p.

Once we have error quadrics $Q_p(i)$ for all the four vertices of the tetrahedron T in consideration, we simply add them to obtain a single *PQEM* as follows:

$$PQEM(T) = \sum_{i=1}^{4} Q_P(i) \tag{4}$$

If T were to collapse to a point v_c, the total geometric error (for T) as approximated by this metric would be:

$$\Delta(T) = v_c^T PQEM(T) v_c \tag{5}$$

3.5 Computing the Fusion Point

Consider a tetrahedron $T = \{v_1, v_2, v_3, v_4\}$. We compute a point of collapse (*fusion point* v) for T that minimizes the total associated *PQEM* as defined in (5). According to [10], this can be done by computing the partial derivatives of $\Delta(T)$, and solving them for 0. The result is of the form

$$\bar{v} = Q_1^{-1}[0\ 0\ 0\ 1]^T \tag{6}$$

where

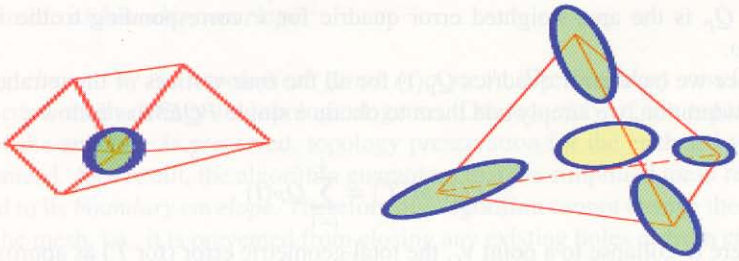
$$Q_1 = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

Note that the terms q_{ij} are coefficients of the respective *PQEM*. There might be cases when the quadric matrix used in the equation is not invertible. In that case, we settle on the barycenter of T as the *fusion point*.

Note that the central ellipsoid in Fig. 8b) represents a level-set surface for the *P*lanar *Q*uadric *E*rror for the *target tetrahedron* shown. This quadric error is the sum of quadric errors of the four constituent vertices of the tetrahedron.

3.6 Example

Figure 9 shows an example of a decimated mesh. The 12,936 element *super phoenix* dataset was decimated using *QTetFusion* in 31.174 seconds, and rendered on an SGI R12000 400MHz with 2048 MB RAM, running Irix 6.52. The increase of computing time over the standard *TetFusion* algorithm is significant (for instance, a factor of 61 for the *super phoenix* dataset, and 46 for the *combustion chamber*; see Table 2 and compare with [2]). However, the decimation rate in most cases was either slightly increased or at least preserved (+12.46% for the *super phoenix* dataset, and −2% for the *combustion chamber*. This is a good result, as the new *QTetFusion* algorithm has significant topological advantages over the standard *TetFusion* method. Diagram 10 and Tables 2 and 3 provide some additional statistics for other datasets.



(a) Polygonal mesh (b) Polyhedral mesh

Fig. 8. Error ellipsoids for affected vertices when the primitive to be decimated is (**a**) an edge and (**b**) a tetrahedron

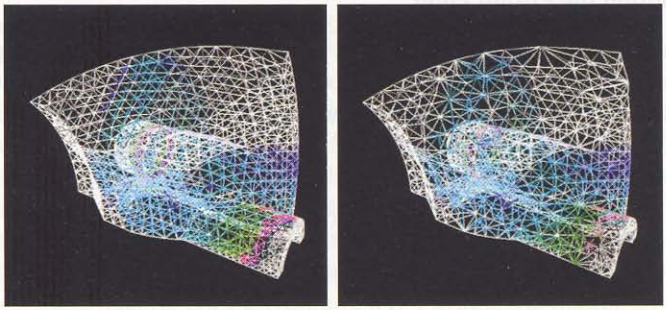


Fig. 9. (**a**) Original (100%) and (**b**) decimated mesh (46.58%) of the *super phoenix* dataset

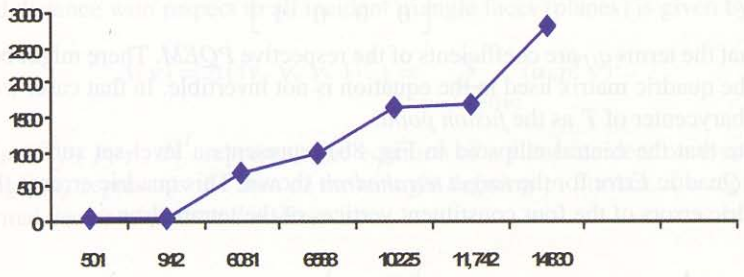


Fig. 10. CPU time in seconds (vertical axis) vs. number of *QTetFuse* operations (tetrahedral collapses) performed (horizontal axis)

Table 2. Number of tetrahedra, decimation ratio and CPU time for various datasets (*QTetFusion*)

#	mesh	n	**dec. rat.**	*QTetFusion (s)*
1.	*super phoenix*	12,936	53.6%	31.174
2.	*blunt fin*	187,395	49.3%	715.074
3.	*comb chamber*	215,040	47.0%	976.603
4.	*oxygen post*	513,375	46.0%	2,803.575

Table 3. Number of tetrahedra, number of decimated tetrahedra, number of *QTetFuse* operations required, average number of decimated tetrahedra for a single *QTetFuse* operation

#	mesh	n	n_{decim}	# *QTetFuse*	Avg. n_{decim}
1.	*super phoenix*	12,936	6,940	501	13.852
2.	*blunt fin*	187,395	92,375	6,081	15.684
3.	*comb chamber*	215,040	101,502	6,588	15.407
4.	*oxygen post*	513,375	238,527	14,830	16.084

4 Time-varying Tetrahedral Mesh Decimation

Extremely high decimation rates can be obtained by taking the temporal domain into account. Instead of tetrahedra, we consider a mesh of hypertetrahedra that consists of tetrahedra that are connected across the temporal domain (Fig. 11). We define a hypertetrahedron as a set of at least two (possibly more) tetrahedra where all four vertices are connected in the time domain. Intuitively, a hypertetrahedron represents a tetrahedron that changes shape over time. Every time step represents a snapshot of the current shape. Without loss of generality, we can assume that the time domain is a linear extension of a three-dimensional cartesian coordinate system. As a consequence, we connect corresponding vertices with linear, i.e. straight, line segments that indicate the motion of a vertex between two or more discrete time steps. Since many tetrahedra do not change significantly over time, hypertetrahedra can be collapsed both in the temporal domain as well as in the spatial domain. This results in hypertetrahedra that are either stretched in space or in time. Mesh decimation in time

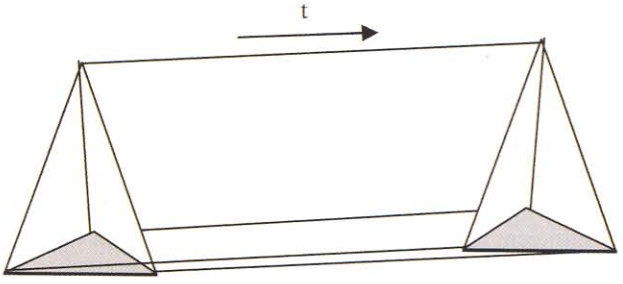


Fig. 11. Hypertetrahedron

means that a hypertetrahedron (4-D) that does not change over time can be represented by a simple tetrahedron (3-D), just like a tetrahedron can be represented by a single point. The opposite direction (expansion of a tetrahedron to a hypertetrahedron over time) is not necessary, because a hypertetrahedron is collapsed only if it does not change significantly in a later time step.

The latter of the previously mentioned cases turns out to restrict the decimation ratio significantly. In the given example of earthquake simulations, many tetrahedra in the peripheral regions do not change at the beginning and towards the end of the simulation, but they are affected during the peak time of the earthquake. Since we do not allow hypertetrahedron expansion from a tetrahedron (split in the temporal domain), a large potential for decimation is wasted. Also, for practical purposes the given approach is not very suitable, because we need to be able to access the position of each vertex in the mesh at every time step if we want to navigate in both directions in the temporal domain. The reconstruction of this information and navigation in time with VCR-like controls requires a global view of the data. This means that the data cannot be processed sequentially for an infinite number of time steps. Consequently, the algorithm is not scalable.

Figure 12 shows an example where one node is affected by a velocity vector. The velocity is proportional to the displacement, because all time steps have the same temporal distance. Therefore, the arrow indicates the position of the node in the next time step. Two tetrahedra are affected by this displacement and change over time. In this example, it would be sufficient to store the time history of the affected node (solid line) or the affected tetrahedra (dotted lines). In order to reconstruct the mesh, it would be necessary to search for the most recent change in the time history of each node, which would require keeping all time histories of all nodes in memory during playback. This becomes particularly obvious if forward and backward navigation in time is considered. Even though this method offers a very compact representation of a time-varying tetrahedral mesh, we propose a different approach that enables easier playback (forward and backward) of all the time steps in a simulation.

The standard method would be to decimate the mesh for each time step separately by applying *QTetFusion* or some other mesh decimation technique, resulting in very high decimation rates in the initial time steps (regular, undistorted grid, no disruption due to earthquake), and moderate decimation of the later time steps where more information needs to be preserved. However, this approach would result in different meshes for every time step, leading to "jumps" and flicker in the visualization. This would be very disrupting in an animation or on a virtual reality display (Sect. 5).

Therefore, we use a different approach. The idea is to preserve every time step, which is necessary for playback as an animation and for navigation in time. The mesh that has the greatest distortion due to the earthquake (the velocity vector values associated with each grid node) is identified, and then decimated. All the decimation steps that were necessary to reduce the complexity of this mesh are recorded. For the record, it is sufficient to store the IDs of the affected tetrahedra in a list, because for the given application the IDs of the tetrahedra are identical in all time steps. Since tetrahedra are only removed but never added, the IDs are always unique. These recorded steps are then used to guide the decimation of the remaining meshes, i.e.,

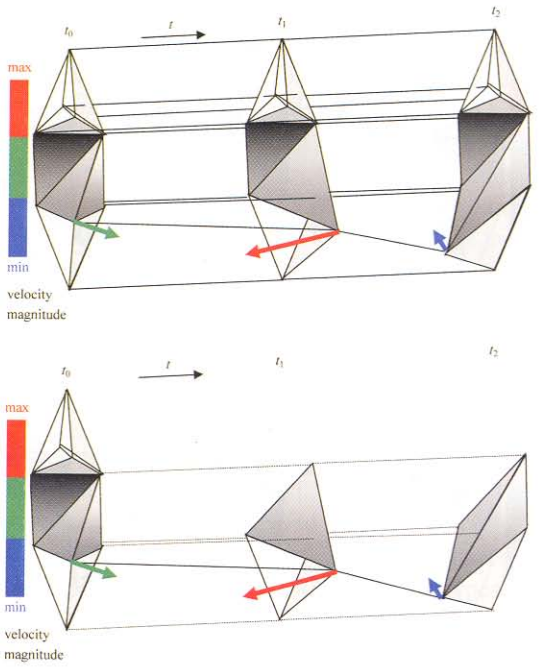


Fig. 12. One affected node, two affected tetrahedra

the meshes of the other time steps are decimated in the exact same manner as the one whose features are supposed to be preserved.

Figure 13 shows that the decimation of t_0 and t_1 is guided by t_2, because t_2 is more distorted than any of the others. The decimated mesh should represent all displaced nodes in the most accurate way, because these are the ones that represent the significant features in the given application. Isotropic regions, i.e., areas that are not affected by the earthquake, such as the three tetrahedra at the top that are simplified into a single tetrahedron, expose only little variance in the data attributes, and consequently do not need to be represented as accurately, i.e., with the same error margin, as the significantly changing feature nodes in the other time steps. Comparing the top scenario (before decimation) and the bottom scenario (after decimation), the image shows that the tetrahedra on the top that were simplified in the selected t_2 time step are also simplified in the other two time steps (t_0 and t_1).

The question that remains is how to identify this "most distorted" mesh. Instead of using complex criteria, such as curvature (topology preservation) and vector gradients (node value preservation), we simply divide the length of the displacement vector for each node by the average displacement of that node, calculate the sum of all these ratios, and find the mesh that has the maximum sum, i.e., the maximum activity. If there is more than one mesh with this property, we use the one with the smallest time index. The average activity of a node is the sum of all displacement

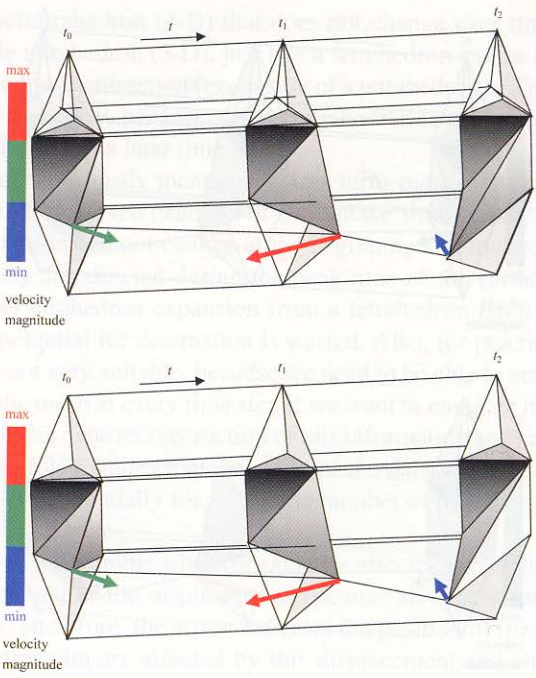


Fig. 13. Preservation of temporal resolution, decimation guided by t_2

vector lengths for all time steps divided by the number of time steps. This means that we consider those nodes in the mesh that expose a lot of activity, and try to represent the mesh that contains these active nodes in the best possible way. The mesh decimation algorithm is applied only to this one mesh. All other meshes are decimated according to this guiding mesh, using the same node indices for the collapse sequence.

The index $mesh_{guide}$ of the mesh that guides the decimation can be calculated in linear time using the following equation:

$$mesh_{guide} = min\{i \in \mathbb{N}_0 | f(i) = max\{f(i) | i \in \mathbb{N}_0\}\}$$

$$f(i) = \sum_{k=0}^{n-1} \frac{\left|\overrightarrow{d}_{k,i}\right|}{\sum_{l=0}^{n-1} \left|\overrightarrow{d}_{l,i}\right|}$$

n number of nodes

$\overrightarrow{d}_{k,i}$ displacement vector k in mesh i.

One drawback of this method is the search for the most active mesh nodes. However, the search is performed in linear time, and the extra time for the search is more than compensated for by the fact that all the other time steps do not need to be decimated by a complex decimation algorithm. Instead, the same list of collapse operations is applied to each time step, resulting in a fast and efficient decimation of the entire data set. This method is scalable, as it does not require loading of more than one time step into memory at one time. It works for an arbitrarily large number of time steps. Also, the algorithm is not restricted to the *QTetFusion* method. It should also work with other decimation algorithms (see Sect. 2).

5 Results from Ground Motion and Structural Response Simulation

The simulation of the effect of an earthquake on a set of buildings was performed using the OpenSees simulation software [22].

A map of the surface (Fig. 15) was used to place a group of buildings along the projected fault line. Two different heights of buildings were used (3 story and 16 story structures, Fig. 14) to simulate various building types in an urban setting. A structural response simulation was calculated for this scenario and then combined with the visualization of the ground motion. The buildings were represented as boxes that resemble the frame structure that was simulated using a SDOF model.

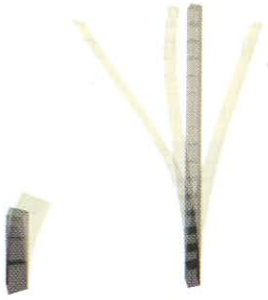
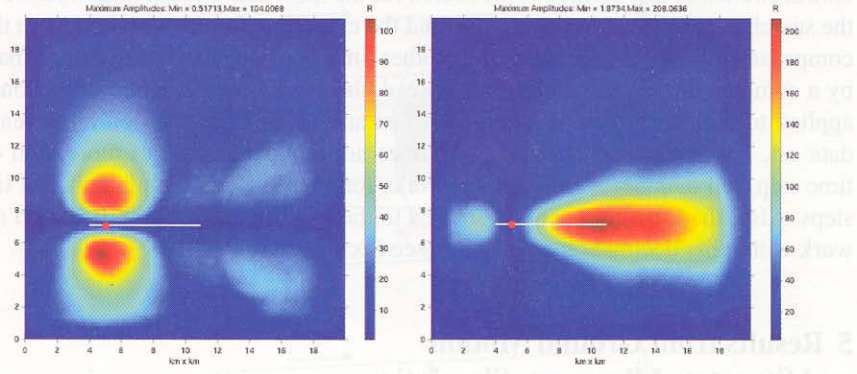


Fig. 14. 3 story vs. 16 story building (story drift horizontally exaggerated)

The scenario can be easily changed by selecting a different set of buildings and different locations on the surface map. The placement of the buildings and the selection of building types could be refined, based on real structural inventory data.

The finite element ground motion simulation was performed on a Cray T3E parallel computer at the Pittsburgh Supercomputer Center. A total of 128 processors took almost 24 hours to calculate and store an 8 second velocity history of an approximately 12 million-node, three-dimensional grid. The required amount of disk space for this problem was approximately 130 GB. Figure 15 shows a 2-D surface plot of the simulation in two coordinate directions.

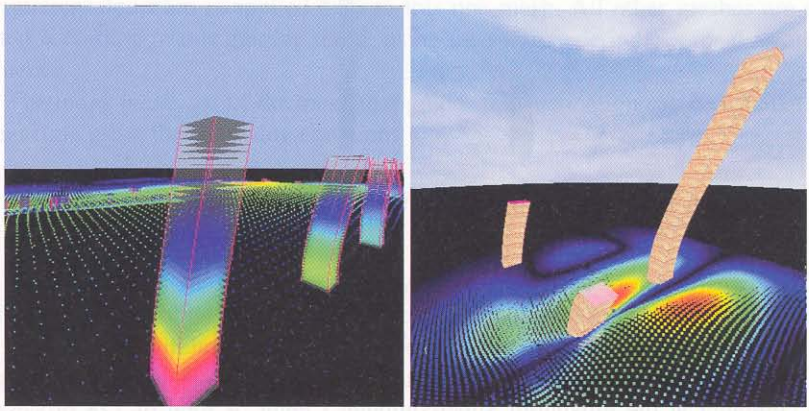


(a) Fault parallel (b) Fault normal

Fig. 15. Velocity plot

Figure 16a shows a 3-D rendering of the surface mesh combined with the structural response simulation. The various intensities (colors) on the buildings indicate the maximum story drift for each floor. Figure 16b shows a hybrid rendering of ground motion and structural response. Textures have been added for a more photorealistic representation of the buildings and the environment.

It turns out that some buildings experience more stress than others, even if they are in close proximity or in the same distance from the fault line as their neighbors. The determining factors are building height, orientation of the frame structure, and building mass.



(a) Ground motion and structural response simulation, (b) with textures.

Fig. 16. (**a**) Ground motion and structural response simulation, (**b**) with textures

Figure 17 shows a scenario in a CAVE^{TM}-like virtual environment (four stereoscopic rear-projection screens with LC shutter glasses and electro-magnetic head and hand tracking system) [18]. The user is fully immersed in the visualization and gets both visual and audio feedback while the shockwave approaches and the buildings start to collapse [4].

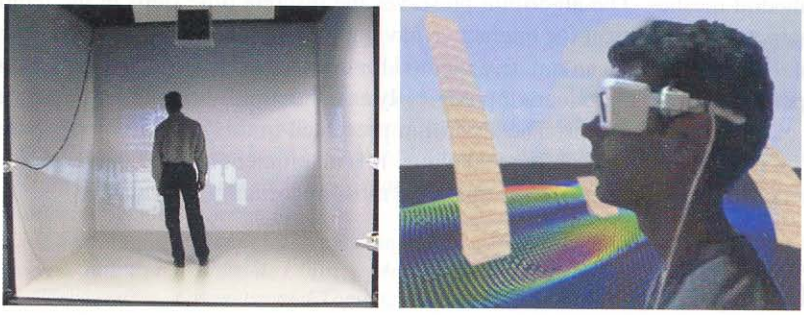


Fig. 17. Virtual environment visualization

6 Conclusions

We presented an integrated framework for domain- and field-error controlled mesh decimation. The original tetrahedral fusion algorithm (*TetFusion*) was extended by employing a *planar quadric error metric (PQEM)*. The additional computational overhead introduced by this error metric is justified by added features, such as topology preservation, and a better decimation ratio. The trade-off between time complexity of *QTetFusion* and the error in either the vertex domain or the attribute field introduced as a result of tetrahedral fusion needs to be investigated in more detail.

The atomic decimation operation employed (*TetFuse*) is symmetric, and better suited for 3-D volumetric meshes than *edge-collapse*-based methods, because it generates less topological inconsistencies (tetrahedra are usually stretched *away* from their base plane). Remaining cases of negative volumes are solved by an early rejection test for tetrahedral flipping. In *QTetFuse*, the barycenter as the center of the tetrahedral collapse has been replaced by a general *fusion point* that minimizes the *PQEM*. This improves mesh consistency and reduces the overall error of the decimated mesh.

A control parameter can be used to provide a smooth and controlled transition from one step to the next. Therefore, the method can be employed to implement a hierarchical level-of-detail set that can be used in multi-resolution rendering algorithms allowing for a smooth transition between multiple levels of detail (hierarchical refinement). One could also think of a view-dependent, localized refinement for applications such as flight simulation.

Our future work includes offline compression of the datasets, as suggested by Alliez et al. [1] and Isenburg et al. [15], and similar to the schemes suggested by Gumhold et al. [12], Pajarola et al. [23], or Szymczak et al. [29]. This would enable dynamic (on the fly) level-of-detail management for volume meshes similar to those methods that currently exist for polygonal meshes [6].

In this chapter, we presented a general method for decimation of time-varying tetrahedral meshes. The algorithm preserves the discrete time steps in the temporal domain, which is critical for interactive navigation in both directions in the time domain. It also represents an intuitive method for consistent mesh generation across the temporal domain that produces topologically and structurally similar meshes for each pair of adjacent time steps. The algorithm presented in this chapter is not restricted to a particular mesh decimation technique. It is an efficient method that exploits and preserves mesh consistency over time, and most importantly, is scalable.

Acknowledgements

This work was supported by the National Science Foundation under contract 6066047–0121989 and through the National Partnership for Advanced Computational Infrastructure (NPACI) under contract 10195430 00120410. We would like to acknowledge Gregory L. Fenves and Bozidar Stojadinovic, Department of Civil and Environmental Engineering, University of California at Berkeley, for providing us with the Structural Response Simulation and the OpenSees™ simulation software, Jacobo Bielak and Antonio Fernández, Department of Civil and Environmental Engineering, Carnegie Mellon University, Pittsburgh, MA, for providing us with the ground motion simulation data, and Prashant Chopra, Z-KAT, Hollywood, Florida, for the software implementation. We would like to thank Peter Williams, Lawrence Livermore National Laboratory, for the *super phoenix* dataset. We would also like to thank Roger L. King and his colleagues at the Engineering Research Center at Mississippi State University for their support of this research contract. Finally, we would like to thank Elke Moritz and the members of the Center of Graphics, Visualization and Imaging Technology (GRAVITY) at the University of California, Irvine, for their help and cooperation.

References

1. Pierre Alliez and Mathieu Desbrun. Progressive Compression for Lossless Transmission of Triangle Meshes. In *Proceedings of SIGGRAPH 2001, Los Angeles, CA*, Computer Graphics Proceedings, Annual Conference Series, pp. 198–205. ACM SIGGRAPH, ACM Press, August 2001.
2. Prashant Chopra and Joerg Meyer. Tetfusion: An Algorithm for Rapid Tetrahedral Mesh Simplification. In *Proceedings of IEEE Visualization 2002, Boston, MA*, pp. 133–140. IEEE Computer Society, October 2002.

3. Prashant Chopra and Joerg Meyer. Topology Sensitive Volume Mesh Simplification with Planar Quadric Error Metrics. In *IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2003), Benalmadena, Spain*, pp. 908–913. IASTED, September 2003.
4. Prashant Chopra, Joerg 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.
5. P. Cignoni, 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 2000, Salt Lake City, Utah*, pp. 85–92. IEEE Computer Society, October 2000.
6. C. DeCoro and R. Pajarola. XFastMesh: Fast View-dependent Meshing from External Memory. In *Proceedings of IEEE Visualization 2002, Boston, MA*, pp. 363–370. IEEE Computer Society, October 2002.
7. T. K. Dey, H. Edelsbrunner, S. Guha, and D. V. Nekhayev. Topology Preserving Edge Contraction. *Publications de l'Institut Mathematique (Beograd)*, 60(80):23–45, 1999.
8. H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001.
9. M. Garland. Multi-resolution Modeling: Survey & Future Opportunities. In *EUROGRAPHICS 1999, State of the Art Report (STAR) (Aire-la-Ville, CH, 1999)*, pp. 111–131. Eurographics Association, 1999.
10. M. Garland and P. Heckbert. Surface Simplification Using Quadric Error Metrics. In *Proceedings of SIGGRAPH 1997, Los Angeles, CA*, pp. 115–122. ACM SIGGRAPH, ACM Press, 1997.
11. T. Gerstner and M. Rumpf. Multiresolutional Parallel Isosurface Extraction Based on Tetrahedral Bisection. In *Proceedings 1999 Symposium on Volume Visualization*. IEEE Computer Society, 1999.
12. S. Gumhold, S. Guthe, and W. Strasser. Tetrahedral Mesh Compression with the Cut-Border Machine. In *Proceedings of IEEE Visualization 1999, San Francisco, CA*, pp. 51–59. IEEE Computer Society, October 1999.
13. I. Guskov, K. Vidimce, W. Sweldens, and P. Schroeder. Normal Meshes. In *Proceedings of SIGGRAPH 2000, New Orleans, LA*, pp. 95–102. ACM SIGGRAPH, ACM Press, July 2000.
14. H. Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH 1996, New Orleans, LA*, pp. 99–108. ACM SIGGRAPH, ACM Press, August 1996.
15. M. Isenburg and J. Snoeyink. Face Fixer: Compressing Polygon Meshes with Properties. In *Proceedings of SIGGRAPH 2000, New Orleans, LA*, pp. 263–270. ACM SIGGRAPH, ACM Press, July 2000.
16. A. D. Kalvin and R. H. Taylor. Superfaces: Polygonal Mesh Simplification with Bounded Error. *IEEE Computer Graphics and Applications*, 16(3):64–77, 1996.
17. M. Kraus and T. Ertl. Simplification of Nonconvex Tetrahedral Mmeshes. *Electronic Proceedings of the NSF/DoE Lake Tahoe Workshop for Scientific Visualization*, pp. 1–4, 2000.
18. Joerg Meyer and Prashant Chopra. Building Shaker: Earthquake Simulation in a CAVE^{TM}. In *Proceedings of IEEE Visualization 2001, San Diego, CA*, page 3, October 2001.
19. Joerg Meyer and Prashant Chopra. Strategies for Rendering Large-Scale Tetrahedral Meshes for Earthquake Simulation. In *SIAM/GD 2001, Sacramento, CA*, page 30, November 2001.

20. B. Munz. *The Earthquake Guide.* University of California at San Diego. http://help.sandiego.edu/help/info/Quake/ (accessed November 26, 2003).
21. M. Ohlberger and M. Rumpf. Adaptive projection operators in multiresolution scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):74–93, 1999.
22. OpenSees. *Open System for Earthquake Engineering Simulation.* Pacific Earthquake Engineering Research Center, University of California at Berkeley. http://opensees.berkeley.edu (accessed November 28, 2003).
23. R. Pajarola, J. Rossignac, and A. Szymczak. Implant Sprays: Compression of Progressive Tetrahedral Mesh Connectivity. In *Proceedings of IEEE Visualization 1999, San Francisco, CA*, pp. 299–305. IEEE Computer Society, 1999.
24. J. Popovic and H. Hoppe. Progressive Simplicial Complexes. In *Proceedings of SIGGRAPH 1997, Los Angeles, CA*, pp. 217–224. ACM SIGGRAPH, ACM Press, 1997.
25. K. J. Renze and J. H. Oliver. Generalized Unstructured Decimation. *IEEE Computer Graphics and Applications*, 16(6):24–32, 1996.
26. W. J. Schroeder. A Topology Modifying Progressive Decimation Algorithm. In *Proceedings of IEEE Visualization 1997, Phoenix, AZ*, pp. 205–212. IEEE Computer Society, 1997.
27. W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of Triangle Meshes. *Computer Graphics*, 26(2):65–70, 1992.
28. O. G. Staadt and M. H. Gross. Progressive Tetrahedralizations. In *Proceedings of IEEE Visualization 1998, Research Triangle Park, NC*, pp. 397–402. IEEE Computer Society, October 1998.
29. A. Szymczak and J. Rossignac. Grow Fold: Compression of Tetrahedral Meshes. In *Proceedings of the Fifth Symposium on Solid Modeling and Applications, Ann Arbor, Michigan*, pp. 54–64. ACM, ACM Press, June 1999.
30. I. J. Trotts, B. Hamann, and K. I. Joy. Simplification of Tetrahedral Meshes. In *Proceedings of IEEE Visualization 1998, Research Triangle Park, NC*, pp. 287–296. IEEE Computer Society, October 1998.
31. I. J. Trotts, B. Hamann, and K. I. Joy. Simplification of Tetrahedral Meshes with Error Bounds. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):224–237, 1999.
32. G. Turk. Re-tiling Polygonal Surfaces. *Computer Graphics*, 26(2):55–64, 1992.
33. Y. Zhou, B. Chen, and A. Kaufman. Multiresolution Tetrahedral Framework for Visualizing Regular Volume Data. In R. Yagel and H. Hagen, editors, *Proceedings of IEEE Visualization 1997*, pp. 135–142, Phoenix, AZ, 1997.