Interactive Point-Based Rendering of Higher-Order Tetrahedral Data

Yuan Zhou and Michael Garland

Abstract—Computational simulations frequently generate solutions defined over very large tetrahedral volume meshes containing many millions of elements. Furthermore, such solutions may often be expressed using non-linear basis functions. Certain solution techniques, such as discontinuous Galerkin methods, may even produce non-conforming meshes. Such data is difficult to visualize interactively, as it is far too large to fit in memory and many common data reduction techniques, such as mesh simplification, cannot be applied to non-conforming meshes.
We introduce a point-based visualization system for interactive rendering of large, potentially non-conforming, tetrahedral meshes. We propose methods for adaptively sampling points from non-linear solution data and for decimating points at run time to fit GPU memory limits. Because these are streaming processes, memory consumption is independent of the input size. We also present an order-independent point rendering method that can efficiently render volumes on the order of 20 million tetrahedra at interactive rates.

Index Terms—Interactive large higher-order tetrahedral volume visualization, point-based visualization.

1 Introduction

Irregular tetrahedral finite element meshes are used in a great many scientific and engineering simulations. Traditionally, these meshes are almost always conforming—vertices are prohibited from falling on an edge or face of an adjacent tetrahedron—and the solutions defined over them are often piecewise linear. However, large tetrahedral meshes that are non-conforming and which represent higher order solution fields are becoming increasingly common. For instance, discontinuous Galerkin finite element methods can easily be formulated on non-conforming meshes as they use independent higher-order basis functions within each element.

Providing effective visualizations of such data is a challenging problem. The meshes are usually quite large, containing many millions of elements. Higher order basis functions significantly increase the memory cost; a piecewise cubic field, for instance, has an order of magnitude more coefficients than a linear field. Consequently, these data sets are generally far too large to fit into main memory, making it very difficult to achieve interactive rates with any volume rendering technique that requires depth sorting. A number of simplification and compression methods have been proposed to help manage such large data, but they are generally not applicable to the kind of non-conforming meshes with potentially discontinuous higher-order solutions that arise in our target applications.

We have developed a point-based system for interactively visualizing higher-order tetrahedral finite element solutions on commodity desktop machines. By choosing a point-based architecture, we are able to cleanly handle non-conforming meshes and discontinuous data fields in the same manner as more typical datasets. While this results in some loss in rendering quality as compared to a mesh-based renderer, the loss is small and is more than outweighed by the resulting performance benefits.

The foundation of our system is a novel adaptive view-independent point sampling method based on a variant of Lloyd relaxation. Because of its size, we process the mesh in a streaming fashion, sampling points from each tetrahedron independently. The memory consumption of this process is thus bounded by a small constant. It is also guided by an error metric that attempts to minimize the error between the underlying algebraic solution and the point-based function approximation that we construct.

We have developed an importance-based stratified point decimation method that automatically tailors the sampled point set at run time to the capacity of the user's hardware. We propose an order-independent point rendering algorithm that replaces explicit depth sorting with a depth-based weighted blending and attenuation scheme. This preserves many depth cues while maintaining high point throughput. We also use a selective shading function to emphasize important interior features. Our rendering method allows us to perform all rendering and blending on the GPU with great efficiency.

Finally, we demonstate the use of our system for visualizing shock surfaces in spacetime elastodynamic simulations. The solutions are produced by a discontinuous Galerkin finite element method and contain up to 17.6 million elements, each of which defines a piecewise cubic displacement field. The shock surfaces that we visualize capture much of the interesting structure of the solution in this kind of problem domain, and can provide significant insight into the behavior of the physical process being simulated.

- *Yuan Zhou is with Department of Computer Science University of Illinois at Urbana-Champaign, E-mail: yuanzhou@cs.uiuc.edu.*
- *Michael Garland is with NVIDIA Corporation, E-mail: mjgarland@acm.org.*

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

2 Related Work

Volume rendering methods can be broadly classified as either indirect methods, such as isosurfacing [20], or direct methods such as ray casting [33], splatting [34], and cell-projection [26]. Many direct volume rendering methods can be accelerated by modern graphics hardware [18]. Most direct methods require depth sorting whereas a relatively small number, most notably maximum intensity projection (MIP) [14] and X-ray methods [31], are order independent. These have a significant performance advantage when handling huge data sets, but their quality is often rather poor because of the loss of depth cues, especially the occlusion. Several methods have been suggested to help provide useful depth cues, including perspective, shading, and stereo rendering [21].

Data reduction methods, such as simplification [11], compression [30] and other multiresolution methods [9], are often used to handle large volume meshes efficiently. However, these methods almost universally assume that the underlying mesh is conforming, and cannot be applied to meshes that violate this assumption. While any mesh may be *made* conforming by a sequence of edge and face splits, this is generally impractical as it can substantially increase the total data size.

Since Levoy [19] proposed using points as display primitives, many point rendering techniques have been developed. They have been

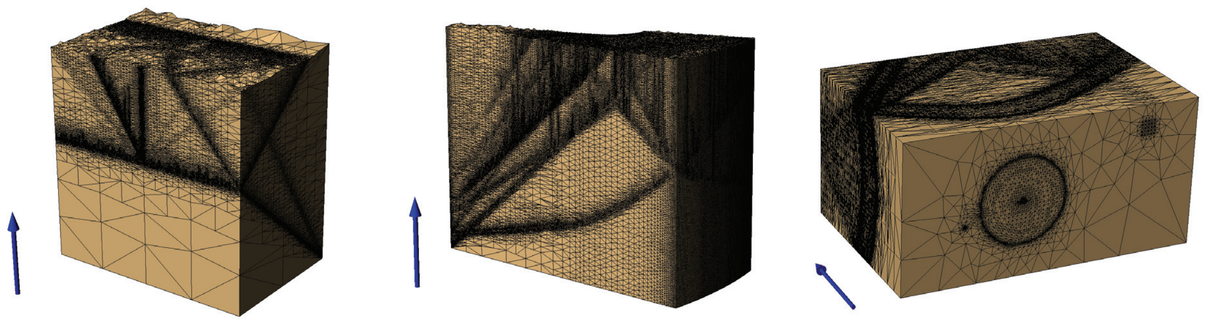

(a) Crack-tip scattering (11.6M tets) (b) Solid rocket section (17.6M tets) (c) Multiscale propagation (9M tets)

Fig. 1. Spacetime tetrahedral meshes for 2-D elastodynamic simulations. Elements vary considerably in size and contain numerous T-junctions. The blue arrows indicate the time axis.

used to interactively render large volume data [13, 22] and complex scenes [32, 10]. Hierarchical structures and multiresolution methods [27, 24, 6] are usually used to sample or render large numbers of points. Point-based models are also naturally suited to memory efficient stream processing [25, 8]. Point sets are often sampled by stochastic sampling [16, 7] or other importance-based techniques. Less common are methods based on point approximation theory. A notable recent example is the work of Grimm *et al.* [12], who use a Taylor series expansion to allocate points in a volume described by a regular grid. Point sets are often rendered by splatting [5, 15], but this normally requires depth sorting of the points. To save time and space when rendering large point sets, order-independent methods [7] can be used. Various techniques can be used to improve the quality of meshless methods so that they rival mesh-based alternatives [2].

Only a handful of existing methods address the visualization of higher order volume data. Most of those that do were designed for rendering surfaces within the volume either by plane slicing [35] or by ray casting [23, 3]. An alternative to directly processing higher order data is to build a sufficiently accurate piecewise linear refinement of the mesh [29]; however, this can also increase data size substantially. Sadowshy *et al.* [28] directly render higher-order volumes by computing higher-order attenuation integrals for projected tetrahedra. Unfortunately, performance can degrade quickly as the size of the integral grows exponentially in the degree of the field.

3 Overview

Our primary target application is the visualization of solutions produced by spacetime discontinuous Galerkin finite element methods [1]. The solution datasets typically consist of non-conforming tetrahedral meshes with tens of millions of elements. For the elastodynamic problems we study here, the solution itself is a displacement field defined per-element as a linear combination of higher order basis functions—cubics for all examples shown here.

Figure 1 shows the three spacetime mesh examples used in this paper. Each is a 3-D spacetime built over a 2-D spatial domain with time following the blue arrows. The simulation shown in Figure 1a models crack-tip wave scattering within an elastic solid subjected to shock loading. The presence of shock waves is implicit, and clearly visible, in the refinement of the mesh. In Figure 1b we see a simulation of wave scattering in a section of a solid rocket booster. Again, the mesh refinement alone hints at a complicated pattern of interweaving superimposed shocks in spacetime. Finally, Figure 1c represents a multiscale simulation of circular waves scattering from the middle of a plate through two arrays of void spaces.

The motivation for our work is to provide a more complete visualization of the obvious and important shock structure that is indirectly apparent in the pattern of mesh refinement. These shocks correspond to sudden changes in the velocity field. Shock waves in a 2-D spatial domain sweep out shock surfaces in 3-D spacetime.

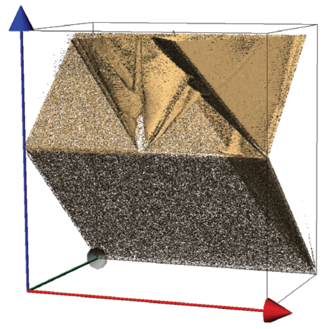


Fig. 2. Visualizing shock isosurfaces for the crack-tip problem (Fig. 1a) produces unsatisfactory results.

Assuming that we could define a scalar "shock strength" scalar field (see §7), it would be natural to consider displaying these surface-like shock features using isosurfacing. However, as we can see in Figure 2, which shows an example of applying Marching Tetrahedra to the crack-tip dataset, this is obviously unsatisfactory. The underlying displacement field is, by its nature, discontinuous between elements. These discontinuities can become even more pronounced in the derivative fields used to compute velocity gradients, and hence the presence of shocks. These discontinuities, coupled with the non-conforming nature of the mesh, make it very difficult to produce a good isosurface. Moreover, due to numerical imprecision, it is extremely difficult to identify a *single* isovalue that corresponds precisely with a given shock surface. Shock waves frequently interweave each other, resulting in complex pattern of shock surfaces, as shown in In Figure 1b.

Rather than isosurface rendering, we aim to produce interactive high-quality volume visualizations of these interweaving superimposed surface-like shock features. We accomplish this by building a point-based approximation of the shock strength field as described in Sections 4 and 5. To render these point sets in real time, we have developed a high-performance order-independent rendering algorithm described in Sec. 6.

4 Point Generation

We take a point-based approach to visualizing these large datasets. In this section, we describe our algorithm for generating a point-based representation of the volume from the initial tetrahedral mesh. This sampling algorithm attempts to produce a point set that provides a good approximation of the underlying scalar field. It is thus generated without regard to run-time constraints on rendering capacity or viewing parameters.

Because of the size of the meshes we wish to process, memory efficiency is a crucial concern. Therefore, we process each tetrahedron independently, treating the entire mesh as a single data stream. Total memory consumption for sampling is thus bounded by a small constant.

For each tetrahedral element, we wish to generate a set of one or more point samples. We assume that the data we wish to render within an element is a scalar field $f : \mathbb{R}^3 \to \mathbb{R}$ expressed as a linear combination of non-linear basis functions. This function f may be the solution itself or a derived field. In our elastodynamics examples, for instance, f is based on velocity gradient magnitudes derived from the underlying displacement field and the basis functions are cubic rational polynomials.

4.1 Sampling at a Point

For a single point at location $x_i \in \mathbb{R}^3$, we wish to record some data about the function f that will allow us to reconstruct it at run time for the purposes of rendering. Like Grimm *et al.* [12], we define a generic sample at x_i to be a partial prefix of the coefficients of the Taylor expansion of f about x_i. However, unlike their work, we operate on continuous functions defined within an element rather than discrete functions defined over regular grids. Therefore we can algebraically compute the Taylor expansion of f

$$f(x_i+h) = f(x_i) + \nabla f(x_i)^T h + \frac{1}{2} h^T [Hf(x_i)] h + \cdots \tag{1}$$

where ∇f and Hf are the gradient vector and Hessian matrix of f, respectively. A generic point sample will thus consist of selected coefficients from the sequence $(f(x_i), \nabla f(x_i), Hf(x_i), \cdots)$.

For the rendering system we present in this paper, we require only the coefficients through first order. Thus for each sample point x_i we record the pair $(f(x_i), \nabla f(x_i))$. Furthermore, because the renderer restricts point footprints to be isotropic, we will use only the order-0 terms for reconstructing the function f and the order-1 terms for normal and shading computations.

To each point sample x_i we assign a spherical *influence region* of radius ρ_i. For a tetrahedron containing k samples, we set these radii so that the volume of each ball is $1/k$ the volume of the tetrahedron. Within each spherical influence volume, we define the approximation error

$$Er(x_i) = \int (f(x_i) - f(x))^2 \, dV \tag{2}$$

using the value $f(x_i)$ that was sampled at x_i. The total approximation error for a tetrahedron containing a set of samples X is simply the sum

$$Er(X) = \sum_{x_i \in X} Er(x_i) \tag{3}$$

4.2 Estimating Sample Set Size

Before allocating point samples within an element, we wish to make a rough estimate of the number of sample points needed. This will make the relaxation algorithm described in the next section more efficient.

Since we are rendering scalar volumetric data with piecewise-constant samples, the number of points to be sampled from each element should be related to the range of the scalar field over this element. With a tetrahedron τ, we define the local contrast S as the ratio of the function range within τ to the function range over the entire volume:

$$S = \frac{\max f_\tau - \min f_\tau}{\max f - \min f} \tag{4}$$

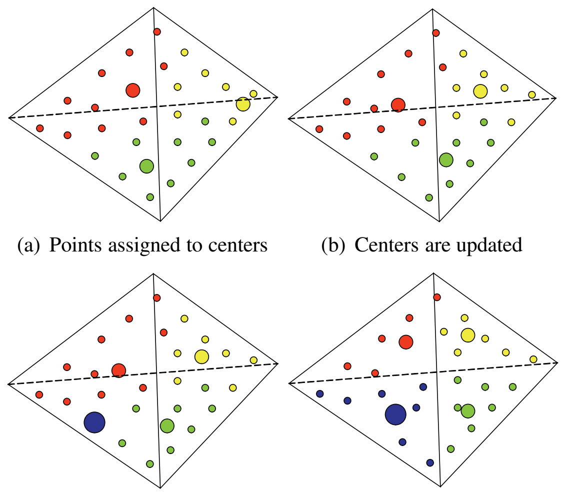
We choose the initial number of points K to be sampled from τ to be

$$K = \left\lfloor \frac{S}{S_T} \right\rfloor + 1 \tag{5}$$

where S_T is user-alterable contrast threshold, for which we generally use a value of 10^{-3}. Also note that this definition guarantees at least 1 sample point per tetrahedron.

4.3 Picking Sample Locations

We now have an initial estimate of how many points should be sampled from a given tetrahedron. Using this as a starting point, we apply a Lloyd relaxation method to find a good number of point samples and their proper positions so that the point approximation error is small.


(a) Points assigned to centers (b) Centers are updated

(c) New center inserted (d) Points are reassigned

Fig. 3. An illustration of one iteration of Lloyd relaxation, with which we position sample centers in each tetrahedron.

We begin by picking a discrete set of N "testing" points within the tetrahedron at which to evaluate the field. This approximation in lieu of actually computing the error integrals defined above avoids considerable overhead. These points are distributed on a regular grid in the barycentric space of the element, and we find a sampling density of about 300 points per tetrahedron works well. We pick K testing points at random to become sample points, and associate each testing point with the sample point which minimizes the approximation error at the testing point. This partitions the testing points into clusters with each sample point being the "representative" of its corresponding cluster. Having constructed these clusters, we move the representative sample points to the geometric barycenter of their associated testing points. We repeat this process of clustering and sample relocation until the samples are not moved or the total error reaches a specified threshold. If the process converges with an error higher than the threshold, or if it does not converge within some maximum number of iterations, we insert additional sample points. Figure 3 illustrates this process.

This algorithm is quite similar to the well known Lloyd relaxation method for k-means clustering. However, notice that we use the Taylor approximation error to grow clusters and the Euclidean metric for repositioning samples in clusters. Consequently, the approximation error will not always shrink monotonically. Therefore, in cases where we terminate relaxation due to hitting the maximum iteration count, we may need to look back through past iterations to find the sample point configuration with smallest total error.

5 Point Decimation

After generating points within all tetrahedra, the entire point set has been built on disk. Since the sampling is performed with respect to approximation error, it may well contain more points than can be efficiently rendered on the target PC. Therefore, we must be able to select a subset of the sampled points that will fit within the rendering capacity of the user's hardware while still faithfully reproducing the solution data.

5.1 Importance Culling

For most simulations, including those we examine here, there are large regions of the solution domain in which relatively little of interest

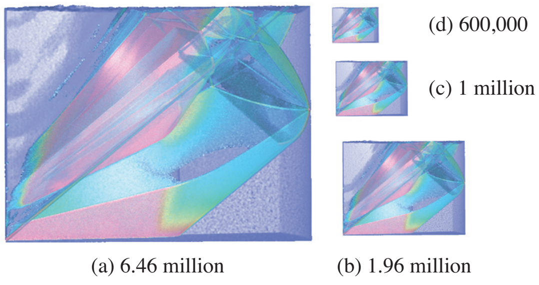


Fig. 4. Decimation of the point set for the solid rocket section at multiple levels of detail. Labels indicate number of points rendered, and images are shown at proportionally reduced resolutions.

is happening. In our elastodynamic examples, these are the regions where the velocity gradient is near 0. We consider it safe to cull points in these less important regions—excluding points on the volume boundary—which are devoid of "interesting" features. In typical examples, a very large number of points can be culled in this manner. In our solid rocket dataset, for instance, a total of 10.4 million points, about 56% of the total, can be culled due to lack of importance.

When a suitable importance threshold is known *a priori*, as it is with our examples, we can integrate culling into the sampling phase. After the set of samples for a single tetrahedron has been generated, any sampled point which falls below the threshold is simply discarded.

5.2 Stratified Decimation

At this stage, we have a collection of "important" points. Based on the memory size and rendering efficiency of the target GPU, we can compute a target number of points to retain that can be rendered at interactive rates. To achieve this target, we need a decimation procedure that can be executed when the renderer is initialized. It is obviously essential that the decimation process be extremely efficient, so that it adds only minimal overhead to the total initialization time. To achieve this goal, we use an importance-based stratified field decimation method that removes important points uniformly.

One stratified field sampling approach, such as the one used by Callahan *et al.* [4], is to divide the range of f into uniform intervals and to randomly select equal numbers of samples from each interval. This is quite effective if the histogram of samples is fairly uniform over the range of f. However, for cases such as those of interest to us, where the histogram is extremely uneven and where the majority of samples occur in small ranges of the histogram, this approach does not work well.

Instead of uniformly stratifying the range of f, we uniformly stratify by sample density. First, we sort all points by their sample value. We can now construct m intervals with (nearly) equal numbers of points. If our point target is n, then we will randomly select n/m points from each interval. For the solid rocket section shown in Figure 4a, about 1.5 million points were removed from the raw 7.5 million point set.

After decimation, we must adjust the influence regions of the surviving points so that the volume is covered appropriately. Since we remove points uniformly, we can simply enlarge the volume of each influence region by the ratio of the sizes of the original and surviving point sets.

Figure 4 illustrates the results of our decimation process on the solid rocket section data. The point set size for Figure 4a was chosen to achieve interactive rendering rates. In addition, we selected progressively smaller sizes while simultaneously decreasing the output resolution. As we can see, the output quality is maintained quite well by our decimation approach.

6 Point Rendering

Our aim is to produce high-quality visualizations of complex interweaving shock surfaces in such a way that their structure will be clearly revealed. We also want to enhance these surface-like features so that they stand out from the surrounding volume. Kraus [17] renders isosurfaces with order-dependent volume rendering techniques, enhancing them with effects such as silhouette illumination. However, this method renders the isosurfaces with uniform opacity and color, independently of the local gradient of the visualized scalar field, and this is not suitable for our data where multiple surface interweave with wide scalar value ranges.

We have already discussed the fact that it is difficult to visualize datasets that are too large to fit into memory using typical volume rendering methods. One of the primary reasons for this is their reliance on depth sorting. Disregarding the use of parallel clusters for visualization, establishing efficient data structures and out-of-core sorting are expensive and will not in general allow us to achieve interactive rendering rates. For our problem, the somewhat atypical order-independent methods, such as MIP and X-ray rendering, are attractive alternatives. They independently combine any sampled value in any order to obtain the final value. But while order-independent methods have good performance characteristics for very large volumes, they necessarily lose occlusion depth cues. Therefore, we augment the basic order-independent method with additional terms that help provide enhanced depth cues without depth sorting.

The core of our order-independent rendering method is a weighted accumulation technique where the weight of a point depends on its depth. The color $\mathbf{I}(q)$ of a pixel q is a weighted sum over the set of points $\{p\}$ whose screen space footprints contain q

$$\mathbf{I}(q) = \frac{\sum_p \mathbf{I}_p W_p}{\sum_i W_p} \tag{6}$$

The contribution $\mathbf{I}_p$ for each point p is determined using the selective shading function

$$\mathbf{I}_p = \begin{cases} C(f), & f < F \\ S(f), & f \geq F \end{cases} \tag{7}$$

We distinguish points by whether their function value f is above or below a threshold F. For those below, we simply use a direct color mapping indicated by $C(f)$; all examples in this paper use a rainbow color ramp for this mapping. Those points above the threshold are considered surface features, which we shade using the standard Phong illumination model. This shading function, indicated by $S(f)$ uses the sampled gradient of the scalar field f as the normal for shading. The purpose of this selective shading is to enhance the important interior features (e.g., shocks in our elastodynamic examples).

The weight W_p for a point at depth d is

$$W_p = \begin{cases} \alpha_0 e^{-d}, & f < F \\ \alpha_1 e^{-d}, & f \geq F \end{cases} \tag{8}$$

Here again, we assign different attenuation rates to non-feature (α_0) and feature (α_1) points. By choosing $\alpha_1 > \alpha_0$ we can prevent distant feature points from being unnecessarily obscured by intervening non-feature regions.

Recall that we assign each point a spherical influence region. Therefore, the screen projection of the point should be the projection of the corresponding sphere. However, for efficiency we wish to approximate this using a quadrilateral which bounds the projection of the sphere.

To take advantage of the performance of modern graphics hardware, our weighted blending scheme is implemented entirely on the GPU. This requires two passes where all processing is performed in fragment programs. The first pass processes each point in a streaming fashion. For each point, we compute its contribution to every pixel within its projection. Each pixel has its own accumulator storing the contribution from all points. The contribution is the shading value weighted by the distance attenuation function. The distance weight is also recorded for normalization in second pass. All accumulation values are stored in a texture containing one texel per screen pixel. After all points are rendered to this texture, on the second pass, the accumulated values for each pixel are fetched and normalized by the accumulated weights. We use 16-bit floats for these computations as 8-bit pixels have insufficient dynamic range and 32-bit floats incur an unacceptably large performance penalty on current hardware.

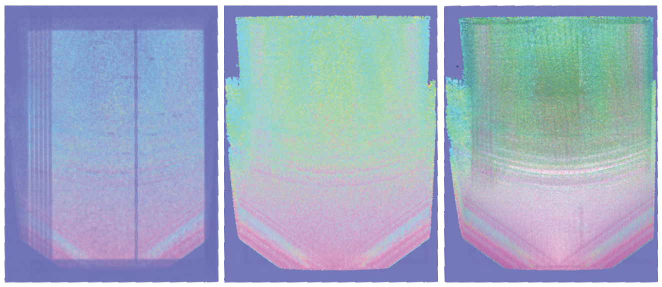

(a) Simple blending (b) ... depth weighting (c) ... and shading

Fig. 5. Adding depth weighting and selective shading to simple color blending produces substantially enhanced images.

Figure 5 shows the impact of our rendering enhancements on the multi-scale propagation dataset. This group of pictures are the side view of the model in Figure 7. Figure 5a is an image generated using order-independent unweighted color blending. Since all depth information is lost, it is very hard to see the structure of the shock surfaces that we are viewing. For instance, it is unclear whether the left vertical lines are closer or further than the right vertical lines. Figure 5b is rendered using our depth-weighted blending. Clearly, more of the structure of the solution has become evident. However, the picture looks monotone because distant features are attenuated by the distance-based weight function. Figure 5c is generated by our method. We assign a different diminishing factor for feature and non-feature parts. And large scalar field regions are selectively shaded to enhance the feature. This rendering clearly provides much clearer depth cues than either of the other images.

7 Shocks in Spacetime Elastodynamic Simulations

For the examples used in this paper, the solution field is a 2-dimensional displacement field. They are represented using a cubic polynomial barycentric basis, containing 20 individual functions. For each point (x,y,t) in spacetime, $\overrightarrow{\alpha} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ represents the barycentric coordinates of (x,y,t) within the surrounding tetrahedron, and the displacement is:

$$\mathbf{u}(\overrightarrow{\alpha}) = \sum_{i=1}^{20} \mathbf{c}_i \mathbf{m}^\mathbf{i}(\overrightarrow{\alpha}) \tag{9}$$

Here, the 2-D coefficient vectors $\mathbf{c}_i$ are the finite element solution data and $\mathbf{m}^\mathbf{i}$ ranges over the barycentric coordinate bases.

We wish to detect shocks in the solution. They are among the most important structural features of the solutions. Shocks are discontinuities in nature. They are abrupt changes of some attribute. To find these discontinuities, the natural approach is to look for very high gradients. The simulation problems on which we focus lie in a spacetime domain, which is different from the general 3-D domain since the spatial and temporal dimensions can be decoupled. Rather than using gradients in the general 3-D domain, we use *spatial* derivatives in the spacetime domain. Therefore, to find the shocks of some field in spacetime, we compute the spatial derivatives of that field. Specifically, we will detect shocks by computing gradients of the velocity field.

The spacetime velocity field $\mathbf{v}(\overrightarrow{\alpha})$ is the temporal derivative of the displacement field.

$$\mathbf{v}(\overrightarrow{\alpha}) = \sum_{i=1}^{20} \mathbf{c}_i \left(\sum_{j=1}^{4} \frac{\partial \mathbf{m}^\mathbf{i}(\overrightarrow{\alpha})}{\partial \alpha_j} \cdot \frac{\partial \alpha_j}{\partial t} \right) \tag{10}$$

We are interested in the *spatial derivatives* of the velocity field:

$$\mathbf{v}_\mathbf{x}(\overrightarrow{\alpha}) = \sum_{i=1}^{20} \mathbf{c}_i \left(\sum_{j=1}^{4} \left(\sum_{k=1}^{4} \frac{\partial \dot{\mathbf{m}}^\mathbf{i}(\overrightarrow{\alpha})}{\partial \alpha_j \partial \alpha_k} \cdot \frac{\partial \alpha_k}{\partial x} \right) \frac{\partial \alpha_j}{\partial t} \right) \tag{11}$$

$$\mathbf{v}_\mathbf{y}(\overrightarrow{\alpha}) = \sum_{i=1}^{20} \mathbf{c}_i \left(\sum_{j=1}^{4} \left(\sum_{k=1}^{4} \frac{\partial \dot{\mathbf{m}}^\mathbf{i}(\overrightarrow{\alpha})}{\partial \alpha_j \partial \alpha_k} \cdot \frac{\partial \alpha_k}{\partial y} \right) \frac{\partial \alpha_j}{\partial t} \right) \tag{12}$$

Note that only 4 components of the basis functions' spatial temporal derivatives $\frac{\partial \dot{\mathbf{m}}^\mathbf{i}(\overrightarrow{\alpha})}{\partial \alpha_\mathbf{j} \partial \alpha_\mathbf{k}}$ are non-zero, therefore only 4 components from $\mathbf{c}_i$ are relevant to this computation.

We define the *shock function* $\mathbf{S}$ as the squared magnitude of the velocity gradient.

$$\mathbf{S}(\overrightarrow{\alpha}) = \mathbf{v}_\mathbf{x}(\overrightarrow{\alpha})^2 + \mathbf{v}_\mathbf{y}(\overrightarrow{\alpha})^2 \tag{13}$$

We allow the user to define a suitable threshold T such that any point with a shock value above T will be considered to lie on a shock.

8 Results and Discussions

In this section, we examine the visual performance of our visualization system. All images were generated interactively at a resolution of 800×600 at roughly 20 frames per second. The machine used for these tests was a Windows XP desktop system with a 3 GHz Pentium 4 Xeon processor, 2 GB of memory, and an nVidia QuadroFX 4500 graphics card.

Figure 6 shows an elastodynamic simulation of a shock wave scattering off a crack in a solid metal plate. Sudden traction loading along one edge of the domain initiates a shock wave that travels across the plate and reflects off the opposite side. This is clearly visible as two planar sheets extending through the spacetime volume. When the shock strikes the crack tip, it creates a new circular wave. In this spacetime rendering, it is visible as a cone-shaped feature. The apex of the cone-shaped region indicates the initial scattering event, as both shown in the left-most images on spacetime mesh in Figure 1 and visualization in Figure 6. As shown in the side view images in Figure 6, the outer perimeter of the cone indicates the progress of the faster-moving dilatational shock wave, while the dark circular band within the cone traces the trajectory of the slower shear shock wave. Our visualization faithfully reflects this physical simulation. The visible presence of a Rayleigh wave—the lighter and steeper wave ascending from the initial edge—is a particularly important feature of use to the engineers engaged in this project. These waves are typically too weak to be found and visualized, whereas our method makes them quite clear. About 6.54 million points are rendered.

Figure 7 shows a multiscale model where pressure and shear shock waves propagate from a circular plate in the domain through two arrays of voids at either end of the circular plate. The faster pressure shock waves are reflected, the shear waves at right are slower and are not reflected during this simulation. Two vertical bands are generated as the waves pass through two arrays of voids. For the coarser array at right, the shear waves are scattered in different directions because of the disturbance from the void. For the dense array of voids at left, there is almost no scattering of shocks to be captured. Viewing this obvious abnormal phenomenon leads our engineering collaborators to examine their simulation more carefully, discovering areas of insufficient numerical precision and overly weak boundary conditions. Thus, we have first-hand evidence that our visualization system can serve as a useful and practical diagnostic tool for engineers engaged with a particular simulation problem. About 6.95 million points are rendered.

Figure 8 is an example showing that our distance weighted order-independent rendering method can visualize complicated interweaving shocks without losing depth information. This is a model of a single sector of a 2-D cross section of a star shaped solid rocket grain. As combustion initiates within the rocket core, the grain is subjected to sudden pressurization. Pressure and dilatational waves are transmitted through the section, and surface waves are moving along the sector boundaries. As time advances, these waves inter-reflect and intersect

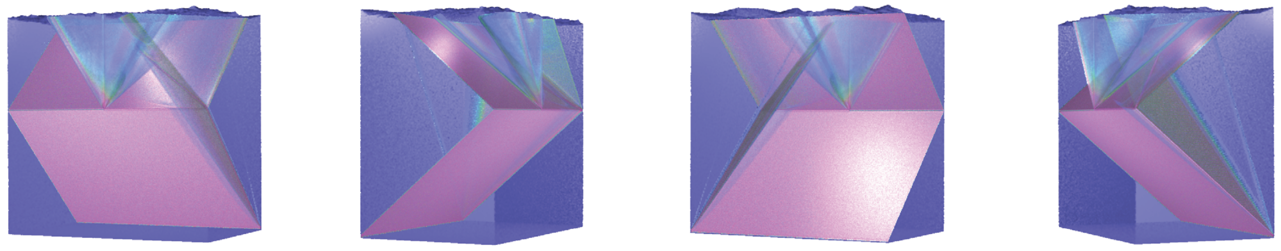


Fig. 6. Shock visualization in crack-tip scattering problem from different views. About 6.54 million points are rendered interactively.

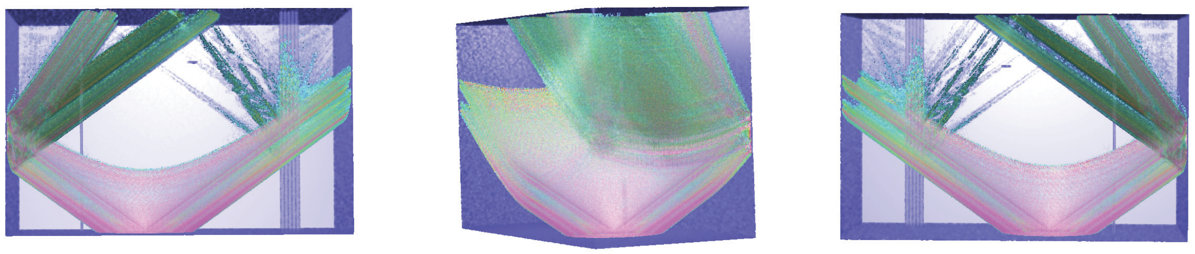


Fig. 7. Shocks visualization for multiscale simulation model from different views. About 6.95 million points are rendered.

other waves, forming a complex wave pattern (see, for instance, the upper right corner of the volume in the left-most image). Our pictures clearly convey the history of even these complicated shock wave behaviors. About 6.5 million points are rendered.

The quality of our meshless point-based volume visualization method result is very close to the quality of rendering by direct ray casting. This comparison is illustrated in Figure 9. We rendered this 11.6 million tetrahedra mesh using both our interactive renderer (Figure 9a) and a ray caster (Figure 9b). Both used the same order-independent rendering equation and the same transfer function. The interactive point renderer produced an image in 0.05 seconds for about 6.54 million points while the tetrahedron ray caster took 10 minutes. Despite the huge discrepancy in running time, the rendered results are practically identical.

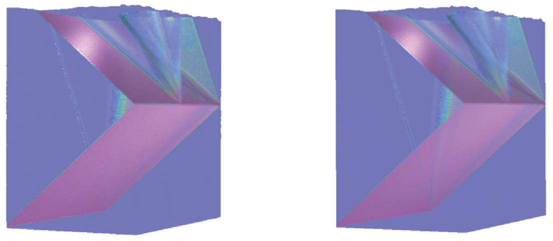

(a) Point rendering (0.05 s) (b) Ray casting (10 min)

Fig. 9. Using 6.54M points, our renderer achieves quality comparable to ray casting of 11.6M tetrahedra in a fraction of the time.

Our visualization method was developed with a specific application—rendering spacetime shock surfaces—foremost in mind. However, there is nothing that prevents us from using it to visualize more traditional kinds of finite element solution data. Shown in Figure 10 is the well-known blunt fin dataset, which we have converted to a fully conforming tetrahedral mesh with linear basis functions. Here we are visualizing the energy field of the solution. Our system is able to produce quality renderings that highlight much of the structure of the solution while maintaining a 20 fps rendering rate.

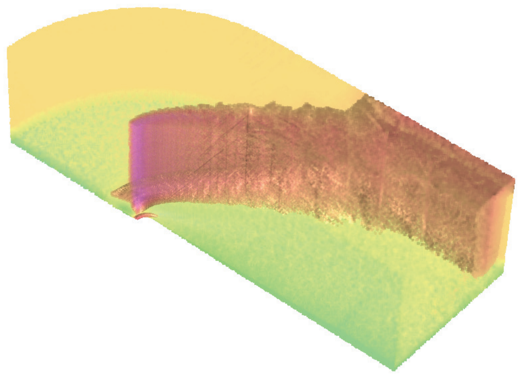


Fig. 10. Visualizing the energy field of the blunt fin dataset.

Table 1 summarizes the flow of data through our system. The input meshes range in size up to 17.62 million tetrahedra. The number of points sampled is generally fairly close to this size; the solution order is low enough that only a small number of points (usually 1) are required in each tetrahedron. Large numbers of points can frequently be culled as “unimportant”. The final point set sizes are all roughly the same as they were all rendered on the same hardware and were thus subject to the same capacity constraints.

Table 2 summarizes the overall performance of the various stages of our system. Point generation is clearly the dominant cost in the system. However, this is done off-line and must be done only once for each dataset. Culling of unimportant points requires very little time. Decimation, which must be performed during initialization of the renderer, requires on the order of 15 seconds. While it would be ideal if this were instantaneous, we note that this amount of time is roughly comparable to the time it takes to parse and load the data without decimation. We also note that, for large meshes with higher order solution

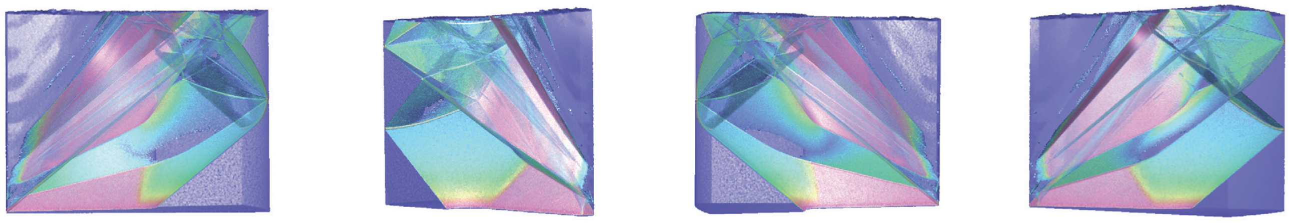


Fig. 8. Shock visualization in solid rocket model from different views. About 6.5 million points are rendered.

		Points (million)			
	Tets (mil.)	Sampled	Culled	Decimated	Rendered
Crack-tip	11.59	11.85	5.374	0.450	6.542
Rocket	17.62	17.96	10.47	1.449	6.509
Multiscale	9.009	9.013	2.522	0.496	6.946
Bluntfin	0.225	7.879	0.929	0.400	6.549

Table 1. Measurements for the point generation and decimation process for the models shown.

	Time (s)				Size (MB)	
	Gen.	Cull	Decimate	Draw	Input	Point set
Crack-tip	2400	1.0	15.0	0.05	1920	213
Rocket	3600	1.3	16.8	0.05	2920	242
Multiscale	1800	1.0	14.5	0.05	1500	227
Bluntfin	290	0.89	14.9	0.05	4.68	212

Table 2. Running time and space requirements for sampling and rendering our example models.

fields, our point conversion approach results in a fairly significant reduction in space. The exception is the blunt fin dataset, which is fairly coarse to begin with and, more importantly, uses only linear basis functions.

9 Conclusion and Future Work

In this paper, we have introduced a point-based visualization system for interactive rendering of large, potentially non-conforming, tetrahedral meshes with high order solutions. We proposed an adaptive view-independent point sampling scheme considering the high order nature of the data. We presented a new importance-based stratified point decimation method which automatically adjust target decimation size for target PC, and a depth-based order-independent point rendering methods. Our system can visualize shocks from tens of millions tetrahedra with cubic order solution in real time.

While our system has already proved useful in a number of ways, there are several areas for possible future work. More sophisticated polynomial approximations could be used in place of Taylor approximation. Our point decimation approach is fairly direct; algorithmic improvements that would result in lower memory consumption should be possible. Using higher-order point primitives in rendering also appears to be a particularly promising avenue for improvement.

Acknowledgements

This research was funded in part by the National Science Foundation under an ITR grant DMR-0121695. We thank the numerous people involved in the CPSD spacetime discontinuous Galerkin project for producing the simulation results visualized here. We would like to specifically thank Reza Abedi for providing us with his simulation data.

References

[1] R. Abedi, B. Petracovici, and R. B. Haber. A spacetime discontinuous Galerkin method for linearized elastodynamics with element-wise momentum balance. *Computer Methods in Applied Mechanics and Engineering*, 195:3247–3273, 2006.

[2] M. Botsch and L. Kobbelt. High-quality point-based rendering on modern gpus. In *Proc. of Pacific Graphics*, pages 335–343, 2003.

[3] M. Brasher and R. Haimes. Rendering planar cuts through quadratic and cubic finite elements. In *Proc. of IEEE Visualization*, pages 409–416, 2004.

[4] S. P. Callahan, J. Comba, P. Shirley, and C. Silva. Interactive rendering of large unstructured grids using dynamic level-of-detail. In *Proc. of IEEE Visualization*, pages 199–206, 2005.

[5] W. Chen, L. Ren, M. Zwicker, and H. Pfister. Hardware-accelerated adaptive EWA volume splatting. In *Proc. of IEEE Visualization*, pages 67–74, 2004.

[6] C. Co, B. Heckel, H. Hagen, B. Hamann, and K. Joy. Hierarchical clustering for unstructured volumetric scalar fields. In *Proc. of IEEE Visualization*, pages 325–332, 2003.

[7] B. Csbfalvi and L. Szirmay-Kalos. Monte carlo volume rendering. In *Proc. of IEEE Visualization*, pages 449–456, 2003.

[8] C. Dachsbacher, C. Vogelgsang, and M. Stamminger. Sequential point trees. *ACM SIGGRAPH*, 22(3):657–662, 2003.

[9] E. Danovaro, L. de Floriani, M. Lee, and H. Samet. Multiresolution tetrahedral meshes: An analysis and a comparison. In *International Conference on Shape Modeling and Applications*, page 83, 2002.

[10] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In *Proc. of IEEE Visualization*, pages 219–226, 2002.

[11] M. Garland and Y. Zhou. Quadric-based simplification in any dimension. *ACM Transaction on Graphics*, 24(2):209–239, Apr. 2005.

[12] S. Grimm, S. Bruckner, A. Kanitsar, and M. E. Gröller. VOTS: VOlume doTS as a point-based representation of volumetric data. *Computer Graphics Forum*, 23(3):668–661, Sept. 2004.

[13] S. Guthe, M. Wand, J. Gonser, and W. Straer. Interactive rendering of large volume data sets. In *Proc. of IEEE Visualization*, pages 53–59, 2002.

[14] W. Heidrich, M. McCool, and J. Stevens. Interactive maximum projection volume rendering. In *Proc. of IEEE Visualization*, pages 11–18, 1996.

[15] M. Hopf and T. Ertl. Hierarchical splatting of scattered data. In *Proc. of IEEE Visualization*, pages 433–440, 2003.

[16] A. Kalaiah and A. Varshney. Statistical geometry representation for efficient transmission and rendering. *ACM Transaction on Graphics*, 24(2):348–373, Apr. 2005.

[17] M. Kraus. Scale-invariant volume rendering. In *Proc. of IEEE Visualization*, pages 295–302, 2005.

[18] J. Krüger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proc. of IEEE Visualization*, pages 287–292, 2003.

[19] M. Levoy and T. Whitted. The use of points as a display primitive. In *University of North Carolina at Chapel Hill Technical Report 85-022*, 1985.

[20] W. Lorenson and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Proc. of SIGGRAPH*, 21(4):163–169, 1987.

[21] B. Mora and D. S. Ebert. Instant volumetric understanding with order-independent volume rendering. In *Proc. of Eurographics*, pages 487–497, 2004.

[22] K. Museth and S. Lombeyda. Tetsplat: Real-time rendering and volume

clipping of large unstructured tetrahedral meshes. In *Proc. of IEEE Visualization*, pages 433–440, 2004.

[23] B. Nelson and R. M. Kirby. Ray-tracing polymorphic multidomain spectral/hp elements for isosurface rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):114–125, 2006.

[24] M. Pauly, M. Gross, and L. Kobbelt. Efficient simplification of point-sampled surfaces. In *Proc. of IEEE Visualization*, pages 163–170, 2002.

[25] P. Renato. Stream-processing points. In *Proc. of IEEE Visualization*, pages 239–246, 2005.

[26] S. Rottger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proc. of IEEE Visualization*, pages 109–116, 2000.

[27] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proc. of SIGGRAPH*, pages 343–352, 2000.

[28] O. Sadowsky, J. Cohen, and R. Taylor. Rendering tetrahedral meshes with higher-order attenuation functions for digital radiograph reconstruction. In *Proc. of IEEE Visualization*, pages 303–310, 2005.

[29] W. Schroeder, F. Bertel, M. Malaterre, D. Thompson, P. Pebay, R. O'Bara, and S. Tendulkar. Framework for visualizing higher-order basis functions. In *Proc. of IEEE Visualization*, pages 43–50, 2005.

[30] S.Gumhold, S. Guthe, and W. Straber. Tetrahedral mesh compression with the cut-border machine. In *Proc. of IEEE Visualization*, pages 51–58, 1999.

[31] T. Totsuka and M. Levoy. Frequency domain volume rendering. In *Proc. of SIGGRAPH*, pages 271–278, 1993.

[32] M. Wand, M. Fischer, I. Peter, F. Meyer, and W. Straßer. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Proc. of SIGGRAPH*, pages 361–370, 2001.

[33] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based ray casting for tetrahedral meshes. In *Proc. of IEEE Visualization*, pages 333–340, 2003.

[34] L. Westover. Footprint evaluation for volume rendering. *Proc. of SIGGRAPH*, 24(4):367–376, 1990.

[35] Y. Zhou and M. Garland. Pixel-exact rendering of spacetime finite element solutions. In *Proc. of IEEE Visualization*, pages 425–432, 2004.