https://doi.org/10.2352/EI.2023.35.1.VDA-392

Mesh Distance for Dimension Reduction and Visualization of Numerical Simulation Data

Shawn Martin, Alex Sielicki, Matt Letter, Jaxon Gittinger, Warren L. Hunt, and Patricia J. Crossno; Sandia National Laboratories, Albuquerque, NM

Abstract

Computational modeling frequently generates sets of related simulation runs, known as ensembles. These simulations often output 3D surface mesh data, where the geometry and variable values of the mesh are changing with each time step. Comparing these ensembles depends on comparing not only geometric properties, but also associated field data. In this paper, we propose a new metric for comparing mesh geometry combined with field data variables. Our measure is a generalization of the well-known Metro algorithm used in mesh simplification. The Metro algorithm can compare two meshes but doesn't consider field variables. Our metric evaluates a single variable in combination with the mesh geometry. Combining our metric with multidimensional scaling, we visualize a low dimensional representation of all the time steps from a set of example ensembles to demonstrate the effectiveness of this approach.

Introduction

Numerical simulations are used to simulate physical phenomenon or predict the behavior of prototype devices before they are manufactured. The simulations considered in this work consist of 3D data produced over time. At each time step, the simulation will compute a 3D mesh[1] and the field variables associated with the nodes of the mesh. Field variables can be either scalar or vector. Scalar variables might include quantities such as temperature or pressure and vector variables might include quantities such as velocity or acceleration.

When analyzing numerical simulation data, practitioners often want to compare one simulation to another or understand how a given simulation evolves over time. Typically this is done by simply viewing a set of simulations and comparing them at each time step. This approach quickly becomes infeasible, however, when the analyst has to consider hundreds or thousands of time steps.

In this paper, we present a visualization approach to assist in understanding numerical simulation data by quantifying time step comparisons using a mesh distance. Most of the previous work in mesh distance comes from the field of mesh simplification [5, 10, 8, 9, 11]. In this field, the goal is to simplify an existing mesh, typically for use in computer graphics and visualization scenarios. The simplified mesh should represent the original mesh as accurately as possible but have fewer nodes and edges. In order to evaluate the simplified mesh, a mesh distance is required. Simplified mesh distance typically does not use field variables [4, 20, 19], but there are some exceptions [18, 17]. There are also metrics which consider visual quality that we do not consider [1]. Of the algorithms that incorporate field variables, our generalization of the Metro algorithm [5] is most similar to the algorithm described in the work by Roy *et al.*, 2004 [18].

Our use of a mesh distance is where the similarity of our work to the work in mesh simplification ends. Our end goal is not mesh comparison, *per se*, but rather the visualization and comparison of multiple time steps in a numerical simulation. Therefore, we use the mesh distance to produce a pairwise distance matrix. The pairwise distance matrix is then used with a dimension reduction algorithm known as multidimensional scaling [3]. Other algorithms that use pairwise distance matrices could also work, for example Isomap [21] or t-SNE [23]. Finally, the resulting dimension reduction is used as the basis of a visualization in a web application called Slycat [7].

Slycat comes from the field of ensemble visualization [12, 13]. Ensemble visualization explores the visualization of data not from a single numerical simulation, but from multiple runs (an ensemble) of numerical simulations. The goal of ensemble visualization is to understand not only a given simulation but also the context of that simulation within the wider scope of multiple similar simulations. Since our metric can be used just as easily on an ensemble of numerical simulations, it can also be used for ensemble visualization. Other algorithms in the field of ensemble visualization include iso-surfaces [2], topological analysis [14, 15], and comparative analysis [16] (among others). None of these algorithms are particularly similar to our approach.

Slycat is a system which provides a web server, a database, and a Python infrastructure for remote computation (on the web server). The user is not burdened with installation/updates and the only requirement is the presence of a Slycat supported browser (e.g. Firefox). In addition, Slycat supports management of multiple users, multiple datasets, and access control, therefore encouraging collaboration while maintaining data privacy. Slycat is implemented using HTML5, JavaScript, and Python. Slycat is open source (github.com/sandialabs/slycat). It is anticipated that an open-source version of our mesh distance software will also be released.

In this paper, we describe our approach in detail. We discuss in detail the algorithms we use to represent the simulation time steps and describe the Slycat user interface. We consider computational cost and demonstrate our system using a dataset obtained from a numerical simulation of a punch impacting a metal plate.

Although we use specific algorithms in this paper, it is important to note that the user interface is completely decoupled from the chosen algorithms. Alternative or new algorithms can be easily substituted for the algorithms described.

[1]It should be noted explicitly that in this work we are concerned with surface meshes, not volumetric meshes.

Algorithms

We use a few different algorithms to produce our time step visualizations, including our generalization of the Metro algorithm. We use the Python trimesh library (https://trimsh.org/index.html) to implement our metric, and the Python library sklearn (https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html) to implement multidimensional scaling. In this section we provide details for the algorithms.

Metro

Metro is an algorithm for comparing two meshes proposed by Cignoni *et al.* in 1998 [5]. Metro computes the approximation error between two mesh surfaces. We start by defining the distance $e(p,S)$ between a point p and a surface S:

$$e(p,S) = \min_{p' \in S} d(p,p'), \tag{1}$$

where $d(p,p')$ is the Euclidean distance between two points p and p'. Next, we define the maximum distance $E_{max}(S_1,S_2)$ between surfaces S_1 and S_2:

$$E_{max}(S_1,S_2) = \max_{p \in S_1} e(p,S_2). \tag{2}$$

We also consider the variations

$$E_{mean}(S_1,S_2) = \frac{1}{n}\sum_{p \in S_1} e(p,S_2), \tag{3}$$

which we call the mean distance, and

$$E_{RMS}(S_1,S_2) = \sqrt{\frac{1}{n}\sum_{p \in S_1} e(p,S_2)^2}, \tag{4}$$

which we call the root mean square (RMS) distance. In all cases, n is the number of samples considered in the computation. These distances are not symmetric since there exist cases where $E_*(S_1,S_2) \neq E_*(S_2,S_1)$, with $*$ representing *max, mean*, or *RMS*. Thus they are not metrics in the strict sense. However, the Hausdorff distance

$$E_H(S_1,S_2) = \max\,(E_*(S_1,S_2), E_*(S_2,S_1)) \tag{5}$$

is two-sided, as is the integral distance

$$E_\int(S_1,S_2) = \frac{1}{|S_1|}\int_{S_1} e(p,S_2)dS, \tag{6}$$

where $|S_1|$ is the area of the surface S_1.

Once one of the variations in Equations (2)-(4) has been computed, it is no more effort to compute the other two, so we generally report all three values. Further, we typically report the Hausdorff distance in Equation (5) based on the three variations. These distances can be also extended to give signed values for orientable surfaces, but we do not use such distances in this work.

Metro with Field Data

The Metro algorithm can be extended to consider both mesh geometry and field data by supplementing the distance $e(p,S')$ in Equation (1) with distances

$$\begin{aligned} e_f(p,S) &= e(p,S) + \sum_{i \in F} d_i(f_i(p), f_i(p')) \\ &= \left[\min_{p' \in S} d(p,p')\right] + \sum_{i \in F} d_i(f_i(p), f_i(p')), \end{aligned} \tag{7}$$

where F is the set of field variables, $f_i(p)$ is the ith field value at point p, $d(f_i(p), f_i(p'))$ is the Euclidean distance between field values at p and p', and p' is the argument $p' \in S$ which achieves the minimum in Equation (1). We then denote the distance between surfaces S_1 and S_2 including field variables using

$$E_f(S_1,S_2) = \sqrt{\frac{1}{n}\sum_{p \in S_1} e_f(p,S_2)^2}. \tag{8}$$

Note that the new distance $e_f(p,S)$ can be computed for minimal additional cost once the point p' is located according to Equation (1). Also note that different combinations of field variables can be easily computed in case the user is interested in a visualizing a particular quantity or group of quantities.

Metro Extension Notes

The Metro equations and our extension describe a variety of possible algorithms. While the "best" variation might depend on circumstances, we can make a few observations about our choices and why they are most likely to work well for our application to numerical simulation data.

First, we have chosen the *RMS* extension in Equation (8), even though we could have also used the *max* or *mean* equivalents by using Equations (2) or (3). The *RMS* extension was chosen in the hopes that it would be most compatible with the field variables. In other words, while it might make sense to compare mesh geometry by using a *max* metric as given in Equation (2), it probably doesn't make sense to compare two fields using a *max* metric. Thus, we compare our mesh fields using the underlying *RMS* metric for the geometry with the analogous *RMS* metric for the fields. Of course, the various metrics could be combined arbitrarily, and in fact other potentially more suitable field data metrics could also be used. We chose the *RMS* metrics because they are the most generic cross-compatible combination. It should also be noted that our choice of d_i in Equation (7) is also arbitrary, but is a similarly safe option since the Euclidean metric transitions well between field variables that maybe scalars or vectors.

Second, there is no reason to assume that the geometry and field data have similar scaling. To avoid having one or the other take on an unnatural importance, we by default scale both the geometry and the field variables to lie in the range $[0,1]$. This is not mentioned explicitly, but is used in the examples. However, it is optional and other scaling possibilities could be easily substituted. In fact, field variables could be weighted according to user expertise and expectation, either in scale or in the actual computations in Equation (8). Again, however, we use the simplest case and assume equals weights.

Third, it is worth noting a connection between mesh simplification [5, 10, 8, 9, 11] and metric mesh computation. In particular, when you compute a mesh simplification, you typically compute local metrics in order to decide how to simplify a mesh [8, 9, 11], and that some of these metrics involve field variables [17]. Since computing metrics can be costly (as will be discussed following), it might make sense to perform a mesh simplification in conjunction with and/or prior to computing our mesh metric. We haven't explored this idea here, but it could potentially lead to much faster algorithms.

Finally, a caveat. Any metric is an attempt to reduce potentially complex information to a single value for the purpose of comparison. Our metric is attempting to combine both geometry and field variables together to produce a meaningful value. In the examples following we demonstrate that the metric works well in the context of numerical simulation data, but we make no claims about general applicability. In the case of numerical simulation data, it can be expected that simulations do not vary greatly from one time step to the next and that even within ensembles we should have comparable fields and geometry. While the metric would still be computable in a general situation, it would not be expected to produce meaningful results if the geometries and fields were vastly different. The limits of the algorithm's ability to perform in general settings would be an interesting topic to explore in future work.

Multidimensional Scaling

After computing a pairwise distance matrix using the mesh distance we use classical multidimensional scaling (MDS) [3] to provide a 2D visualization of the time steps in a simulation. Each entry of the pairwise distance matrix contains the dissimilarity between two time steps in the simulation ensemble. After applying multidimensional scaling we obtain 2D coordinates in a reduced space, where each time step is assigned a 2D coordinate. These coordinates are used to produce a visualization that is displayed in the central pane of the user interface, as shown in Figure 1. To compute the MDS coordinates, suppose we have a surface mesh dataset $\{S_i\}$. We compute a pairwise distance matrix

$$D = \begin{bmatrix} E_f(S_1,S_1) & E_f(S_1,S_2) & \cdots \\ E_f(S_2,S_1) & E_f(S_2,S_2) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}, \tag{9}$$

where $E_f(S_i,S_j)$ gives a distance between mesh i and mesh j according to Equation (8).

Now we use the MDS algorithm to compute coordinates for the distance matrix D. The first step in MDS is to double center the distance matrix, obtaining

$$B = -\frac{1}{2}HD^2H, \tag{10}$$

where D^2 is the component-wise square of D, and $H = I - \mathbf{1}\mathbf{1}^T/n$, n being the size of D. Next, we perform an eigenvalue decomposition of B, keeping only the two largest positive eigenvalues λ_1, λ_2 and corresponding eigenvectors $\mathbf{v}_1, \mathbf{v}_2$. The MDS coordinates are given by the columns of $V\Lambda^{1/2}$, where V is the matrix containing the two eigenvectors $\mathbf{v}_1, \mathbf{v}_2$ and Λ is the diagonal matrix containing the two eigenvalues λ_1, λ_2.

As with the metric itself, we again note that our choice of MDS as a dimensionality reduction algorithm is somewhat arbitrary. Other popular algorithms for dimensionality reduction include Isomap [21] and t-SNE [23], both of which could be used in place of MDS. We use MDS because it is the simplest such algorithm, converging to a unique solution (up to repeated eigenvalues) without requiring additional parameters (neither of which is true for Isomap or t-SNE). In addition, although it would be interesting to compare the results of the different reductions, our metric is fundamentally a method for comparing simulation mesh data, not a method for dimension reduction. We therefore use the simpler MDS algorithm for our initial investigation.

Computational Costs

The main computational cost of the Metro algorithm is the identification of the closest point p' on S_2 to a point p on S_1. The cost depends on the surface area of S_1 and the number of faces on S_2 [5]. Once p and p' are identified, the additional cost for considering field variable data is constant. In our implementation of the Metro algorithm, we considered various approximations, both in terms of sampling the surfaces and computing the closest points. We considered combinations of vertex sampling, edge sampling, and face sampling while also considering approximations to p' in Equation (1) using nearest vertex values, nearest face values, and reverse sampling. We conducted a comparison of the accuracy of the various options using our punch-plate dataset. Finally, we used the embarrassingly parallel nature of the calculation to simultaneously compute blocks of the full pairwise distance matrix[2]. These comparisons will be described in further detail in the Example section.

User Interface

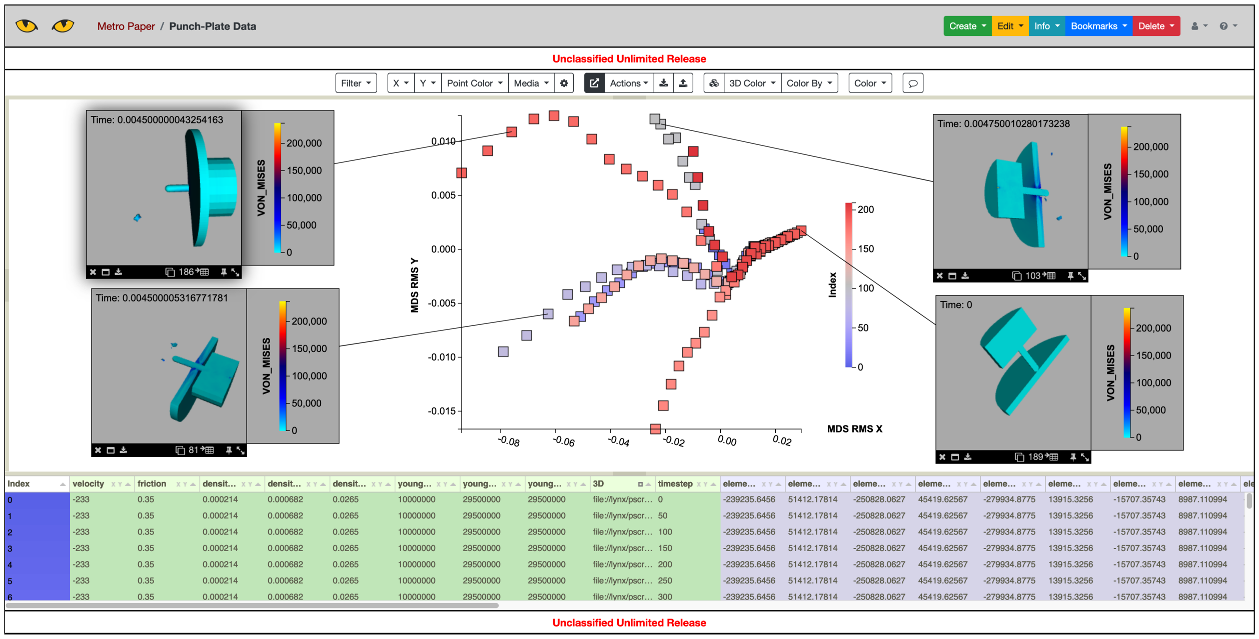
The user interface for our mesh visualization tool is centered around the use of the parameter space model in Slycat [6]. To accommodate 3D visualization, a 3D viewer was added using vtk.js (https://kitware.github.io/vtk-js/index.html). The 3D viewer allows interactive visualization of the surface mesh data.

The full interface consists of two large panes and a variety of controls, some of which open additional panes. The interface is shown in Figure 1. The controls are arranged above the central pane, which is used to display the MDS dimension reduction coordinates, where each point represents a surface mesh and the points are arranged so that proximity reflects mesh distance. The lower pane contains a metadata table, where each row corresponds to a point in the central pane. Points selected in the central pane will be highlighted in the metadata table and vice versa.

The 3D viewer is linked to the plots displayed in the central pane and is fully interactive. The mesh can be rotated and scaled, and can be colored according to any of the metadata provided. Further, multiple 3D views can be opened simultaneously and synchronized according to viewpoint. The viewers themselves can be organized arbitrarily within the central pane by the user.

Finally, the contents of the central pane can be altered to display any data provided when the visualization is created, including metadata and additional MDS computations. In fact, we provide multiple versions of the MDS calculations by computing mesh distance matrices for each field variable separately. The user

[2]It is worth noting that computing the pairwise distance matrix scales $O(n^2)$ with the number n of meshes in our dataset. Thus using a pairwise distance matrix is a fundamentally slow approach. For our initial investigation, however, we are more concerned with the usability of mesh metric comparison itself. Large gains in speed might be obtained by more cleverly computing sets of metric pairs and/or reducing the meshes, topics we leave for future consideration.



Index	velocity	friction	densit...	densit...	densit...	young...	young...	young...	3D	timestep	eleme...	eleme...	eleme...	eleme...	eleme...	eleme...	eleme...	eleme...
0	-233	0.35	0.000214	0.000682	0.0265	10000000	29500000	29500000	file://lynx/pscr...	0	-239235.6456	51412.17814	-250828.0627	45419.62567	-279934.8775	13915.3256	-15707.35743	8987.110994
1	-233	0.35	0.000214	0.000682	0.0265	10000000	29500000	29500000	file://lynx/pscr...	50	-239235.6456	51412.17814	-250828.0627	45419.62567	-279934.8775	13915.3256	-15707.35743	8987.110994
2	-233	0.35	0.000214	0.000682	0.0265	10000000	29500000	29500000	file://lynx/pscr...	100	-239235.6456	51412.17814	-250828.0627	45419.62567	-279934.8775	13915.3256	-15707.35743	8987.110994
3	-233	0.35	0.000214	0.000682	0.0265	10000000	29500000	29500000	file://lynx/pscr...	150	-239235.6456	51412.17814	-250828.0627	45419.62567	-279934.8775	13915.3256	-15707.35743	8987.110994
4	-233	0.35	0.000214	0.000682	0.0265	10000000	29500000	29500000	file://lynx/pscr...	200	-239235.6456	51412.17814	-250828.0627	45419.62567	-279934.8775	13915.3256	-15707.35743	8987.110994
5	-233	0.35	0.000214	0.000682	0.0265	10000000	29500000	29500000	file://lynx/pscr...	250	-239235.6456	51412.17814	-250828.0627	45419.62567	-279934.8775	13915.3256	-15707.35743	8987.110994
6	-233	0.35	0.000214	0.000682	0.0265	10000000	29500000	29500000	file://lynx/pscr...	300	-239235.6456	51412.17814	-250828.0627	45419.62567	-279934.8775	13915.3256	-15707.35743	8987.110994

Figure 1. *User Interface. Shown here is the user interface for the mesh distance visualization. The central pane contains the coordinates from the MDS calculation. Each point is selectable and can be tied to a 3D viewer showing the original surface mesh associated with that point. The lower pane provides an interactive table giving metadata for the time steps visualized (each row in the table corresponds to a point in the central pane). Finally, controls are provided in a toolbar which allow the user to color by different metadata variables, show alternate reductions, and perform various other functions (data export, figure adjustment, and so on).*

can then select the MDS coordinates according to whatever field variable is of interest. The points displayed in the central pane can be colored according to any variable provided as metadata, for example statistical averages or deviations.

Examples

When analyzing numerical simulations, there are two major considerations. First, we want to be able to compare not only mesh geometry, but other properties including location, orientation, field variable values, and potentially even mesh resolution. To ensure that our metric is capable of distinguishing between mesh objects having these differing characteristics, we studied the algorithm on a toy dataset consisting of ellipsoids with various locations, orientations, sizes, shapes, and resolutions.

Second, we want to ensure that the metric calculation runs in a reasonable time but still gives good accuracy. For this study we compared the computational cost and accuracy for different approximations compared to the full calculation. The approximations we considered all attempted to increase the speed of the closest point calculations between two meshes. We considered reduced sampling strategies such as vertex only sampling and sampling size, as well as approximations to closest point calculations including nearest vertex and nearest face values. We compared these strategies to the calculation using full vertex/edge/face sampling and exact closest point calculation.

To examine the behavior of the mesh metric using different sampling strategies and closest point approximations, we used an ensemble of simulations for a punch-plate system. We used this same system to examine the behavior of the metric and MDS visualization when using field variable values. Finally, we considered an additional canister-plate system to compare the mesh only metric with mesh with field value metric calculations.

Toy Data

Our toy data consists of ellipsoids constructed with different mesh properties held fixed and others varied. We generated seven datasets. Each dataset consisted of 30 randomly generated ellipsoids. For location, we generated spheres with the same shape and resolution, but randomly distributed between three clusters. For orientation, we generated ellipsoids of the same shape and resolution, but randomly distributed between three distinct orientations. For size we generated differently sized spheres; for shape we generated ellipsoids with different major and minor axis lengths; and for resolution, we generated spheres of the same size and shape, but with different resolutions. Finally, we generated a dataset with randomly mixed combinations of location, orientation, size, shape, and resolution. In all cases, the metric visualization distinguished between the varied parameter. We show the results for the orientation data in Figure 2, the resolution data in Figure 3, and the mixed dataset in Figure 4. The results for size, shape, and location are not shown due to the fact that the 3D mesh viewer automatically translates and scales the object in the viewer so that size, shape, and location are indistinguishable (although in those cases the MDS scatter plot still reveals the three clusters embedded in the data).

Accuracy and Speed

To test the metric algorithm performance using different sampling strategies and closest point approximations, we used an ensemble of numerical simulations generated with

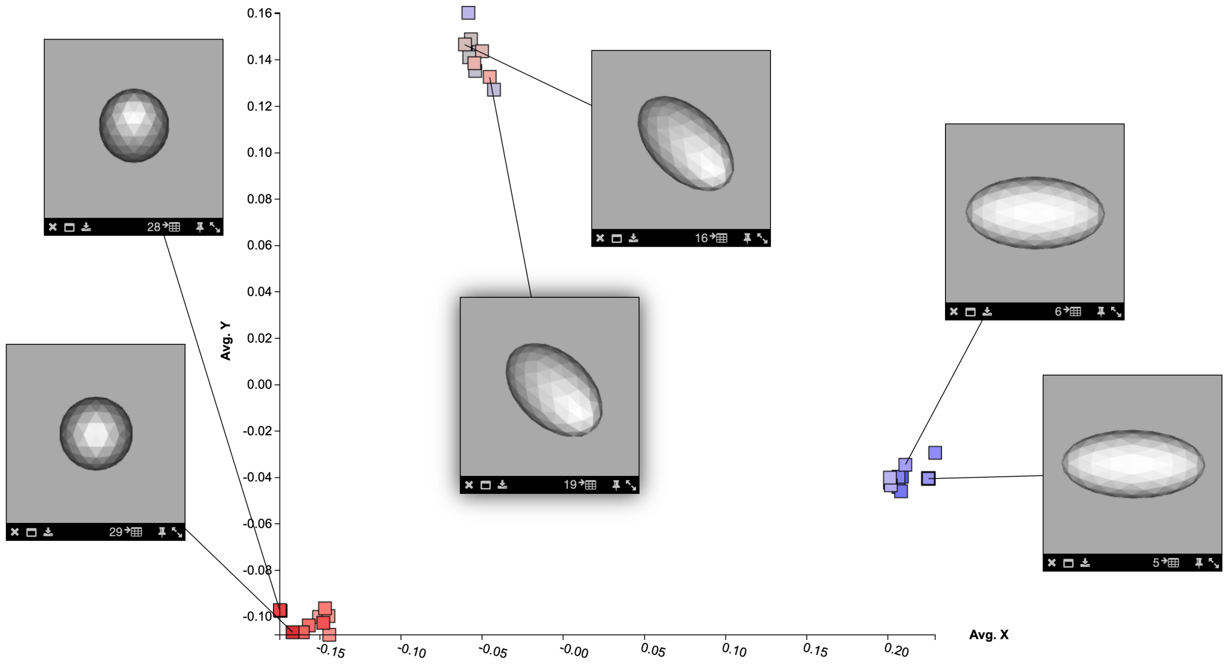


Figure 2. Ellipsoid Orientation Data. The results of the metric visualization using ellipsoids with the same geometry and mesh resolution, but with orientation generated to cluster into three groups. The three clusters were discovered by the metric/MDS calculation. The cluster on the bottom left is the ellipsoids oriented end-on; the cluster on the top corresponds to the ellipsoids oriented at an angle; and the cluster on the right corresponds to the ellipsoids oriented left to right.

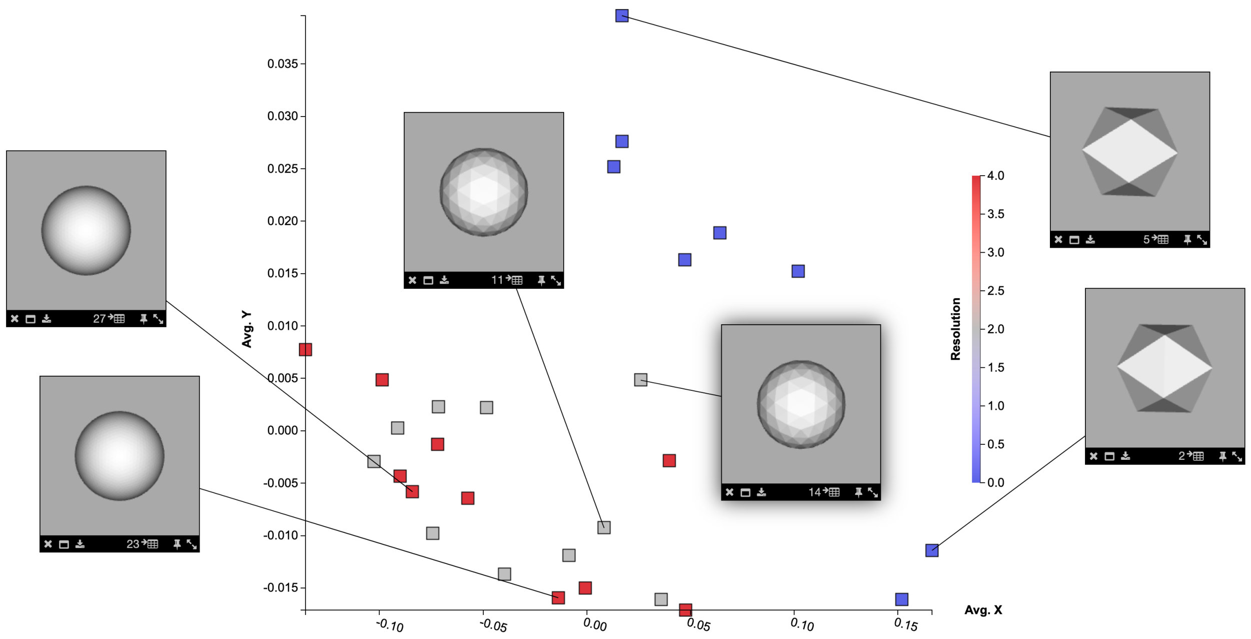


Figure 3. Sphere Resolution Data. The results of the metric visualization on spheres with three distinct resolutions. In this case, the metric separated the three clusters into two groups. On the left side of the scatter plot are the spheres with higher resolutions and and the right are the spheres with very low resolution.

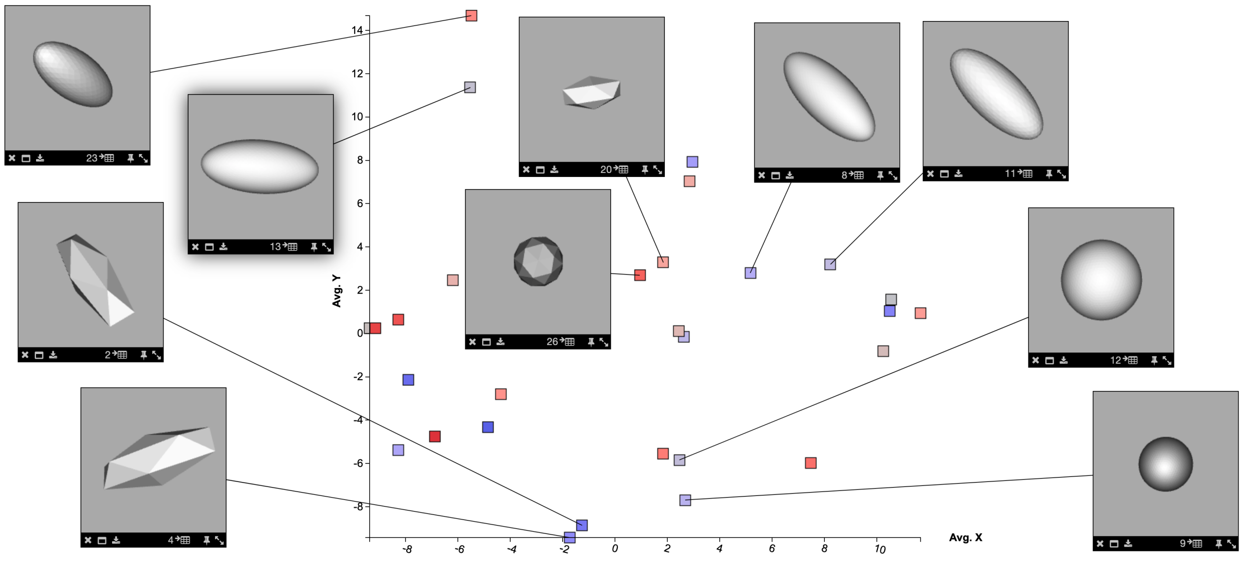


Figure 4. *Ellipsoid Data. The results of the metric visualization on a random combination of ellipsoids of different sizes, shapes, resolutions, locations, and orientations. Although there were many random variations, the metric nevertheless managed to co-locate similar shapes, orientations, and resolutions in the scatter plot. It should be noted that the two groups of ellipsoids (one on the upper left and one in the center) are in fact separated in the reduced dimensional layout because they are different sizes. They appear to be the same size because the mesh rendering software scales the objects automatically in the pinned windows.*

Sierra/SolidMechanics (Sierra/SM) [22], a Lagrangian, three-dimensional code for problems with large deformations and non-linear material behaviors. This ensemble was created to explore the effects of changes in simulation parameters on material fracturing. The modeled object is a punch impacting a metal plate under various conditions, such as different punch velocities, material properties, or plate thicknesses. For each run, the ensemble consists of 8 inputs and 38 outputs (12 scalar results and 16 variables changing over time). The full ensemble is 15K runs, with about a terabyte of data. For our parameter study we used a very small subset of 5 randomly selected runs with 42 time steps each, giving a dataset of 210 total time steps.

We first examined the effect of reduced sampling on the accuracy of the results. We measured accuracy by comparing against the full sampling strategy, consisting of all vertices, midpoints of edges, and a random selection of samples from the mesh faces. For each reduced sampling strategy, we computed the Frobenius norm of the difference between the pairwise distance matrices computed using the reduced sampling strategy and the full sampling strategy. The results are shown in Figure 5. For our reduced sample sets, we first used vertex sampling, followed by edge sampling, followed by face sampling. For a typical mesh in this dataset, there were 12k vertex samples, 150k edge samples, and remaining samples randomly taken from the mesh faces. Thus the 25k sample dataset would contain half vertex samples and half edge samples, and the 200k sample dataset would contain all the vertex and edge samples plus 37k face samples.

Next we compared different closest point approximations when computing the metric distance in Equation (1). We compared our approximations to the standard exact calculation, which computes for each sample $p \in S_1$ the closest point on the surface S_2, thus approximating the distance $e(p,S)$ in Equation (1). We tried three different approximations: first using the nearest vertex in S_2 to $p \in S_1$; next using reverse sampling, a variation using the nearest vertices in S_1 to a random sample on S_2 (including faces); and last using nearest faces, which determines nearest faces on S_2 as containing the nearest vertex on S_2 instead of the faces within a bounding box around $p \in S_1$. Note that the nearest faces method is much faster than the standard implementation, but is not guaranteed to identify all the nearest faces from S_2. If it does identify the correct faces, however, the results are identical. We show the results of our approximations in Figure 6(a).

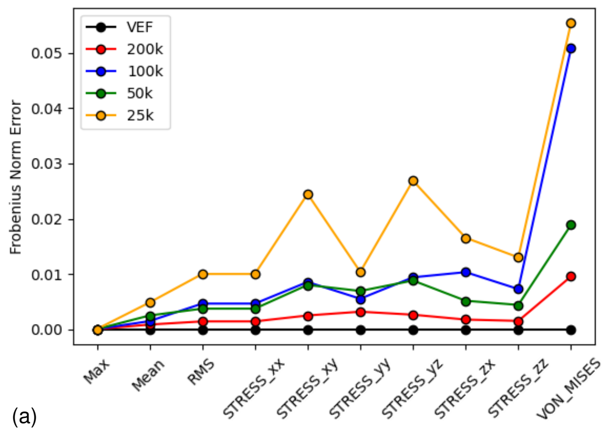
Our study indicates that the nearest faces approximation can use vertex sampling (the smallest possible sample size) and still obtain the most accurate results. It is also the fastest version of the algorithm. Using the full dataset, we can compute the mesh metric pairwise matrix in anywhere from 29 hours and 49 minutes using 8 processors to 2 hours and 28 minutes using 128 processors. This is done in an embarrassingly parallel manner by splitting the full pairwise distance matrix into submatrices. We show how the algorithm scales with number of processors in Figure 6(b).

Simulation Data

We provide examples from two simulation ensembles to demonstrate the use of our metric on real-world data. Both ensembles are parameter studies from impact simulations. The first ensemble is from the punch-plate simulation used previously to understand the speed and accuracy of our algorithm. The second is a canister-plate ensemble, where the effects of different initial position and angle are examined when a canister strikes a plate.

Punch-Plate

The punch-plate data consists of 10 simulations, each with 20 time samples (every 50th time step over 1000 steps). In Figure 7(a), we superimpose all of the time steps for the 10 runs of the punch ensemble. The points are color-coded by time step, transi-



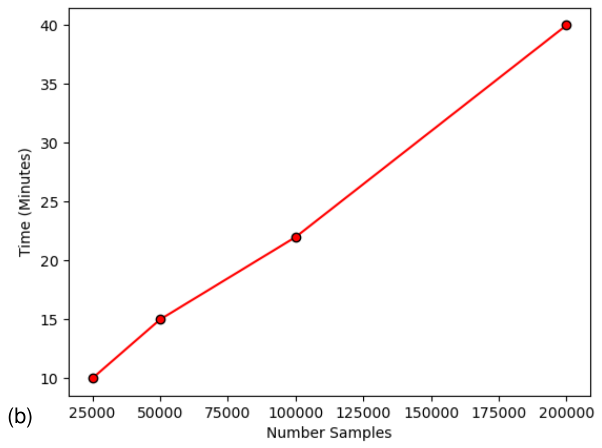
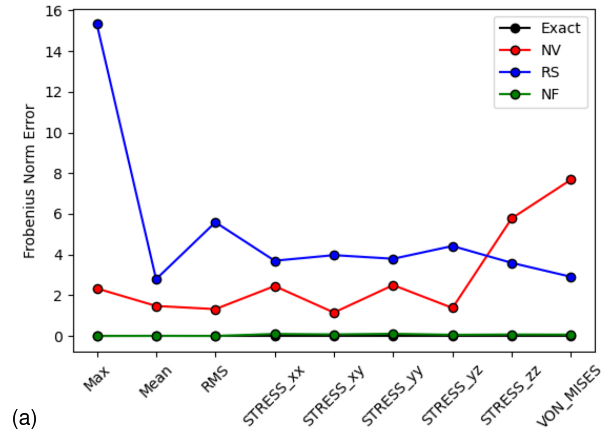


Figure 5. *Sampling Study. Here we compare the effect of varying the number of samples used to compute the mesh metric. On the left (a), we compare a baseline sampling strategy taking 250k samples (labeled VEF) with reduced sets of samples. For the reduced sample datasets, we computed the geometric mesh metrics max, mean, and RMS in Equations (2)-(4), as well as the geometric field variable metrics based on stress for the punch-plate dataset. On the right (b), we show that the algorithm's time requirement scales linearly with number of samples.*



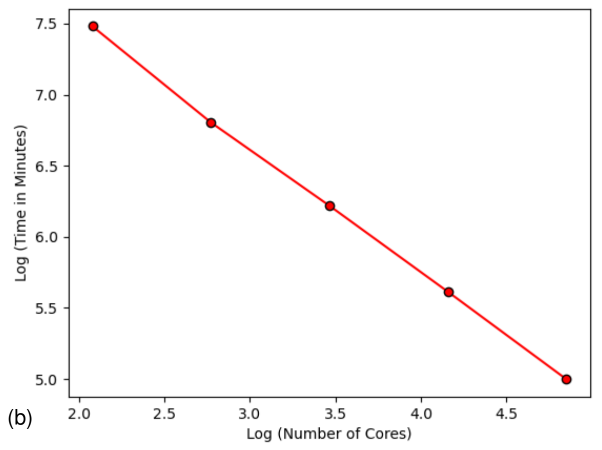


Figure 6. *Approximation Study. Here we compare the effect of different closet point approximations used to compute the mesh metric. On the left (a), we compare the nearest vertex, reverse sampling, and nearest faces approximations to the exact calculation. For each approximation, we computed the geometric mesh metrics max, mean, and RMS in Equations (2)-(4), as well as the geometric field variable metrics based on stress for the punch-plate dataset. (Note that the Exact and NF curves are overlapping.) On the right (b), we show how the embarrassingly parallel version of the algorithm scales with number of processors.*

tioning from blue to white to red as time increases. Highlighting the final time steps (larger dark red points), a shared pattern can be seen across all runs, in which they all begin at the origin, curve up to the right, peak at a sharp transition, dip down to the left, then end with a rise, also to the left. Most runs end along the upward slope on the left, though two stand out as being anomalous. To understand what distinguishes these two, we pull up the surface meshes for all of the final runs and compare them, as shown in Figure 7(b). The runs that finish together on the left are all the simulations where the punch has penetrated the plate. The two anomalous runs are those where the punch has failed to penetrate. (It should be noted that the visualization can also be colored and/or filtered by simulation so that the user can identify specific simulations.)

An additional point of interest occurs at a discontinuity in the timelines. In Figure 8, we examine the surface meshes for two runs, focusing on the time steps immediately before and after the reversal. We've overlaid arrows on the image to highlight the time sequence of the points. In both cases, the Von Mises stress increases until the point of transition (the red coloration of the post), then sharply drops off once a chunk of the plate has started to separate as a plug.

These two examples demonstrate that patterns can be used to reveal events and other transitions, reducing the number of surfaces that the user needs to visually inspect.

Canister-Plate

The canister data has 124 simulations, each with 25 consecutive time steps. In Figure 9(a), we show all of the time steps for the 124 runs of the simulation. In this example, we see very different patterns from those observed in the punch-plate example. Here the runs do not follow a single pattern, but instead represent differences in the angles of the canister's initial position and that of the plate. In Figure 9(b), we look at a subset of runs for a pitch angle of 45 degrees. We see that the clock angles of 90 and 180 degrees generate patterns that are almost mirror images of each other. Similar patterns can be seen with other combinations of pitch and clock angles.

Using the trajectory visualization technique for the punch-plate data (Figure 8), we show a typical trajectory for a canister-plate simulation in Figure 10. In the case of the canister-plate data, the trajectory proceeds from the upper left corner of the reduced dimensional space to the lower right, where the canister experiences maximum overall Von Mises stress as it bounces off the plate. The trajectory then curves back to the center of the scatter plot, where the canister experiences high local stress causing the lid to open. It is interesting to note that the canister experiences a similar state and stress both as it first traverses the center of the scatter plot and during the final moments of the simulation. This can be seen using the 3D viewer, as shown for both cases in Figure 10.

Metric Comparisons

The punch-plate data can also be used to illustrate some of the properties of our mesh metric and it's behavior using field variables. First, it is interesting to observe the difference between the *max*, *mean*, and *RMS* options using geometry alone, as shown in Figure 11.

Second, we look at the effect of using a field variable on the mesh metric. The field variables associated with the punch-plate data are primarily involved with measuring the stress in the plate. In Figure 12, we show the metric visualization considering both geometry and Von Mises stress.

Discussion

Comparing time steps using mesh data from numerical simulations is a very difficult problem for a variety of reasons. First, mesh data is not uniform in either space or underlying dimension. For example, objects modeled by a mesh in a numerical simulation will very often change in shape and orientation from one time step to the next, and hence the number of vertices, edges, and faces in the mesh will change as well. Second, mesh data encodes not only geometry but also field data described by both scalar and vector quantities of interest. Third, mesh data is "big," because all of the encoded 3D geometric and field information must often be recorded at a high resolution over many time steps to obtain good simulation results. Fourth, we desire to analyze ensembles of simulations rather than just one simulation at a time. Finally, the metric comparisons must be presented in a usable form for scientists and engineers for interpreting the results of their simulations.

Our proposed metric for mesh comparison attempts to address all of these difficulties and provide a visualization tool for ensembles of simulations. We demonstrated using the toy data that the underlying metric can differentiate between shape, orientation, and even mesh resolution. We demonstrated that the metric can be extended to include field variables as desired, and compared the results using data from a punch-plate simulation. A particular strength of this approach is that we can tailor our analysis by considering different field variables and geometric metrics, all of which can be computed simultaneously.

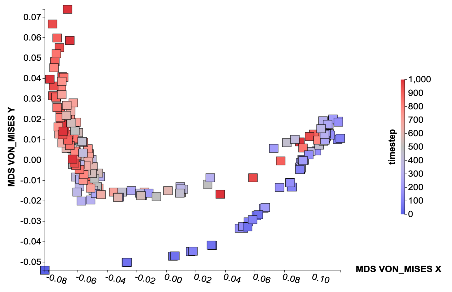
In future work, we will investigate the effect of different sets of field variables on the quality and information content of the visualizations. For example, we could obtain multiple layouts of simulation snapshots by allowing the user to select specific sets of variables and view side-by-side comparisons between the resulting visualizations.

An additional future avenue of investigation, especially given that we are studying time series data, would be the algorithmic consideration of periodic data, for example spinning objects or data with some rhythmic undercurrent (such as a heart beat). Although this could be addressed by incorporating time as an additional axis in the scatter plots (making them 3D), it might be better to use an additional time based term in the metric itself.

In practice, our biggest challenge was computational expense, which we addressed by comparing different approximations for speed and accuracy using the punch-plate data and using parallel implementations. In the future, we would like to further speed up the algorithm and process larger ensembles and greater numbers of time steps.

Our mesh metric can be used as a stand-alone tool, but it is far more useful to an actual analyst as a part of a visualization system. To that end we developed a 3D viewer for the time step data and incorporated the comparisons into Slycat as an end-user visualization system. This system supports rapid investigation of simulation ensembles.

(a)



(b)

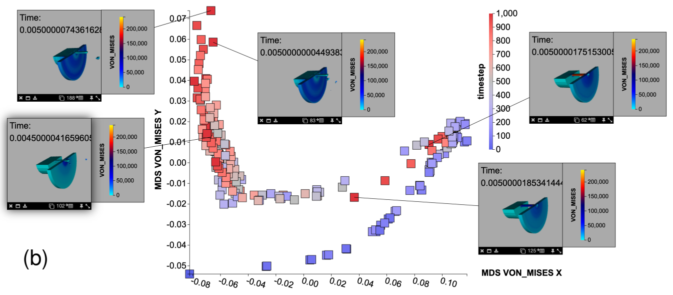
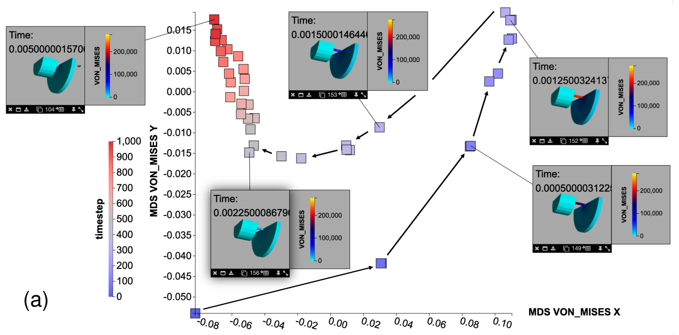


Figure 7. *Punch Timestep Evolution. On the left (a), we simultaneously view all time steps from all runs in the punch ensemble. The points are color-coded by the time step, with earliest time step in dark blue and the final time steps highlighted and colored dark red. The initial time steps are all superimposed at the origin. Following time steps curve up and to the right until there is a discontinuity, after which the sequences reverse direction and curve back and to the upper left. All but two simulations have their final time step along this final vertical section. (It should be noted that the visualization can also be colored and/or filtered by simulation so that the user can identify specific simulations.) On the right (b), we have retrieved the surface models for each of the final time steps. Note that the punch has failed to penetrate the plate in the two unusual runs.*

(a)



(b)

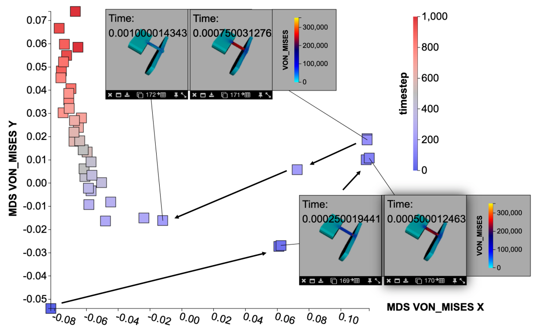
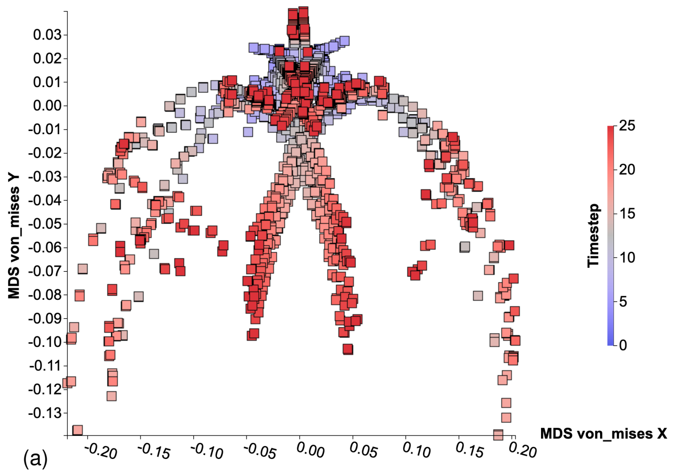


Figure 8. *Punch Trajectories. On the left (a), we show how the trajectory for a mid-range punch velocity appears using our mesh metric with the Von Mises feature. Each point represents a time step in a particular simulation selected using a given (mid-range) punch velocity. The points are colored according to timestep, where blue represents the initial time step and red is the final time step. Following the points from blue to red, the trajectory proceeds from the lower left to the upper right as the punch strikes then breaks through the plate. It is interesting to observe that while the MDS coordinates are computed without regard to the time steps, the trajectory is nevertheless preserved, since it is related to the Von Mises field values as the punch impacts then punctures the plate. A similar pattern is observed using a higher punch velocity on the right (b).*

(a)



(b)

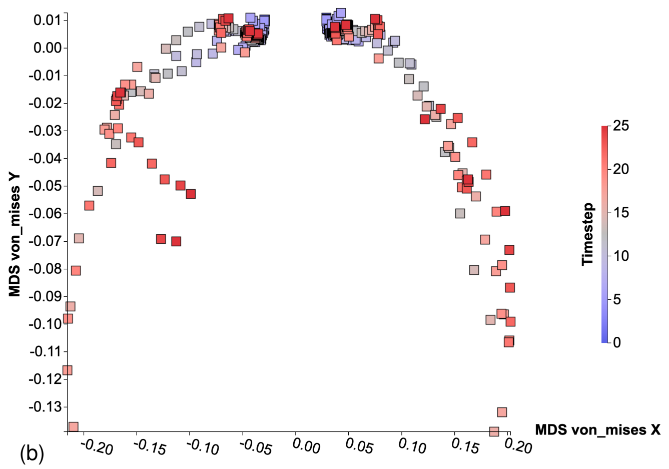


Figure 9. *Canister Timestep Evolution. On the left (a), we simultaneously view all time steps from all runs in the canister ensemble. Here the pattern reflects differences in the parameters around the angle of the can and the angle of the plate. On the right (b), we have filtered the angles, revealing that the 90 and 180 degree clock angles generate patterns that are mirror images of each other.*

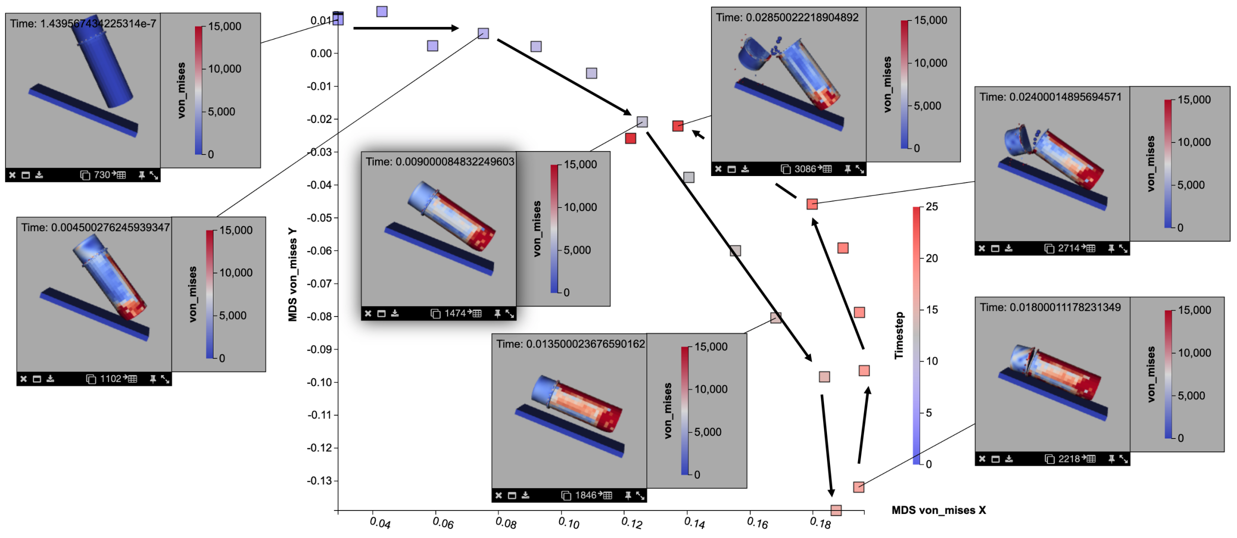


Figure 10. *Canister Trajectory. Here we show the trajectory of a single canister-plate trajectory in the reduced dimension scatter plot. The trajectory proceeds from the upper left of the scatter plot to the lower right, passing through the center as the end of the canister bounces off the plate. The Von Mises stress is highest on the lower right as the canister fully contacts the plate then lower as the trajectory continues back to the center of the scatter plot, where the canister again bounces off the plate, this time causing the lid to open.*

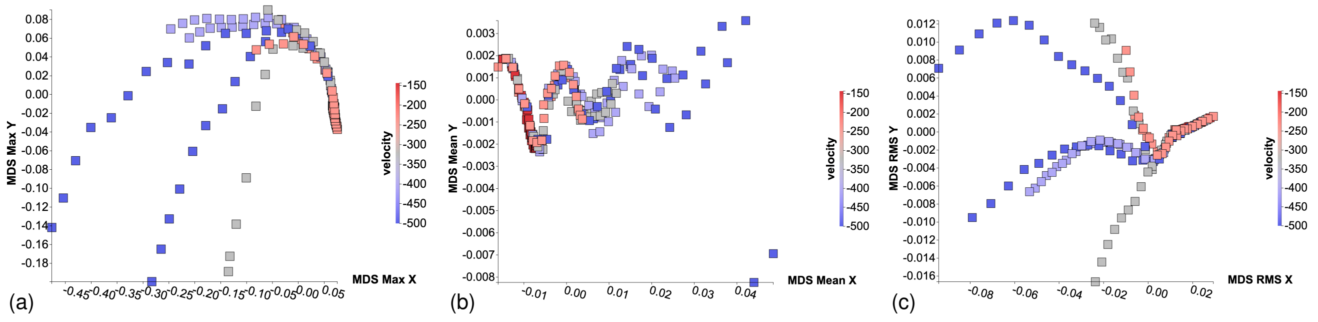


Figure 11. *Metric Geometric Behavior. Here we compare the effect of using max, mean, or RMS to compute the mesh metric, as defined in Equations (2)-(4). On the left (a) we show the max metric, in the middle (b) we show the mean metric, and on the right (c) we show the RMS metric. In all cases, the five simulation runs in the punch-plate data can be easily distinguished as trajectories in the scatter plot. They originate at the same point (same initial conditions), and diverge depending on punch velocity. For the max metric (a), the trajectories diverge in a waterfall pattern. For the mean metric (b), the points follow an oscillating pattern, most likely corresponding to the vibration of the plate when struck by the punch. For the RMS metric (c), the points diverge from the moment the punch strikes the plate.*

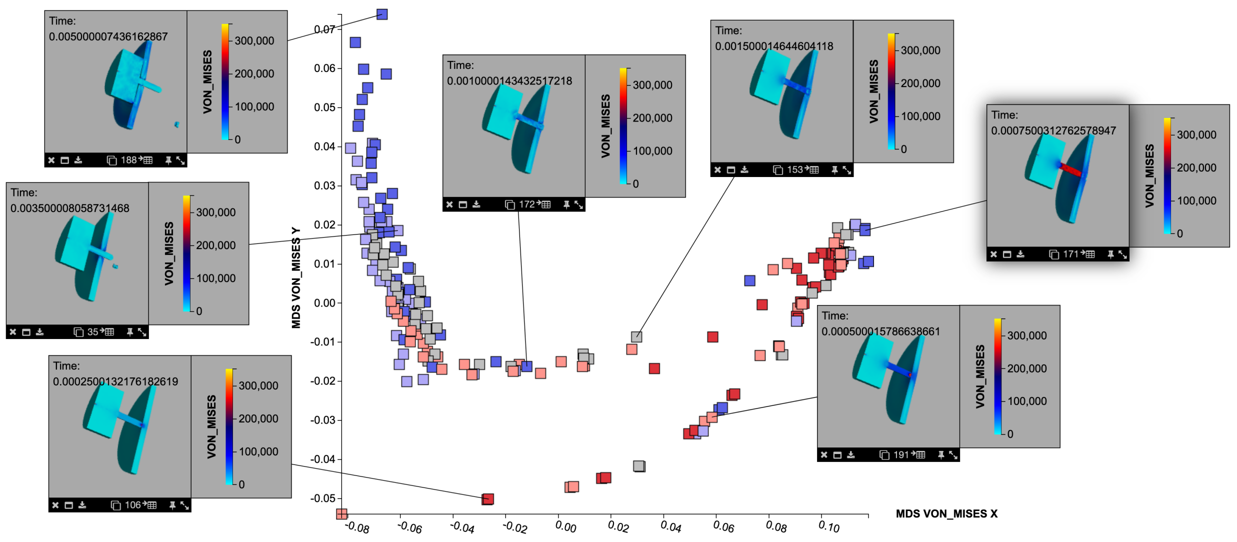


Figure 12. *Metric Field Variable Behavior. Here we show the effect of a field variable on the mesh metric, in particular the Von Mises stress on the plate in the punch-plate data. Although the five simulation trajectories are still discernible (see Figure 11), they are now grouped according to whether or not the punch actually makes it through the plate.*

Conclusion

Surface mesh output from numerical simulations is often of primary interest to scientists. As the number of time steps and simulations increase, however, scientists cannot reasonably examine each mesh individually. We have developed a mesh metric to compare time steps and provide an abstraction using dimension reduction to simultaneously visualize all the time steps in a simulation ensemble. Our representation allows scientists to analyze large collections of surface mesh time steps from multiple simulations simultaneously. Our metric provides not only comparison based on geometry, location, and orientation, but also scalar and vector field variables.

Further, the results of our calculations have been integrated into an interactive web application using the Slycat framework. This application lets scientists easily view and interact with both the abstract representation of the data as well as examine each time step in detail. Observations can be bookmarked and shared between collaborators.

Together our metric, dimension reduction, and interactive interface enable analysis of numerical simulation surface mesh data on a scale previously unavailable.

Acknowledgements

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

[1] I. Abouelaziz, A. Chetouani, M. El Hassouni, and H. Cherifi. Mesh visual quality assessment metrics: A comparison study. pp. 283–288, 12 2017. doi: 10.1109/SITIS.2017.55

[2] O. S. Alabi, X. Wu, J. M. Harter, M. Phadke, L. Pinto, H. Petersen, S. Bass, M. Keifer, S. Zhong, C. Healey, and R. M. T. II. Comparative visualization of ensembles using ensemble surface slicing. In P. C. Wong, D. L. Kao, M. C. Hao, C. Chen, R. Kosara, M. A. Livingston, J. Park, and I. Roberts, eds., *Visualization and Data Analysis 2012*, vol. 8294, pp. 318 – 329. International Society for Optics and Photonics, SPIE, 2012. doi: 10.1117/12.908288

[3] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[4] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998. doi: 10.1016/S0097-8493(97)00082-4

[5] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998. doi: 10.1111/1467-8659.00236

[6] P. Crossno. Challenges in visual analysis of ensembles. *IEEE Computer Graphics and Applications*, 38(2):122–131, 2018. doi: 10.1109/MCG.2018.021951640

[7] P. J. Crossno, T. M. Shead, M. A. Sielicki, W. L. Hunt, S. Martin, and M.-Y. Hsieh. Slycat ensemble analysis of electrical circuit simulations. In J. Bennett, F. Vivodtzev, and V. Pascucci, eds., *Topological and Statistical Methods for Complex Data*, pp. 279–294. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[8] M. Garland and P. S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. *Proceedings Visualization '98 (Cat. No.98CB36276)*, pp. 263–269, 1998.

[9] H. Hoppe. New quadric metric for simplifiying meshes with appearance attributes. In *Proceedings of the Conference on Visualization '99: Celebrating Ten Years*, VIS '99, p. 59–66. IEEE Computer Society Press, Washington, DC, USA, 1999.

[10] D. Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, 21(3):24–35, 2001. doi: 10.1109/38.920624

[11] F. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, 2003. doi: 10.1109/TVCG.2003.1196006

[12] H. Obermaier, K. Bensema, and K. I. Joy. Visual trends analysis in time-varying ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 22:2331–2342, 2016.

[13] H. Obermaier and K. Joy. Future challenges for ensemble visualization. *Computer Graphics and Applications, IEEE*, 34:8–11, 05 2014. doi: 10.1109/MCG.2014.52

[14] M. Otto, T. Germer, H.-C. Hege, and H. Theisel. Uncertain 2d vector field topology. *Computer Graphics Forum*, 29(2):347–356, 2010.

[15] M. Otto, T. Germer, and H. Theisel. Uncertain topology of 3d vector fields. In *Proceedings of the 2011 IEEE Pacific Visualization Symposium*, PACIFICVIS '11, p. 67–74. IEEE Computer Society, USA, 2011.

[16] H. Piringer, S. Pajer, W. Berger, and H. Teichmann. Comparative visual analysis of 2d function ensembles. *Computer Graphics Forum*, 31(3pt3):1195–1204, 2012. doi: 10.1111/j.1467-8659.2012.03112.x

[17] M. Reberol and B. Lévy. Computing the distance between two finite element solutions defined on different 3d meshes on a gpu. *SIAM Journal on Scientific Computing*, 40(1):C131–C155, 2018. doi: 10.1137/17M1115976

[18] M. Roy, S. Foufou, and F. Truchetet. Mesh comparison using attribute deviation metric. *Int. J. Image Graph.*, 4:127–, 2004.

[19] J. Schmidt, R. Preiner, T. Auzinger, M. Wimmer, M. E. Gröller, and S. Bruckner. Ymca — your mesh comparison application. In *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 153–162, 2014. doi: 10.1109/VAST.2014.7042491

[20] A. Taime, A. Saaidi, and K. Satori. *Comparative Study of Mesh Simplification Algorithms*, vol. 380, pp. 287–295. 01 2016. doi: 10.1007/978-3-319-30301-7_30

[21] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000. doi: 10.1126/science.290.5500.2319

[22] J. D. Thomas. Sierra/solid mechanics 4.22 user's guide. 10 2011. doi: 10.2172/1029807

[23] L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.