SANDIA REPORT
SAND2022-12293
Printed September 2022

Sandia National Laboratories

304L Can Crush Validation Studies

Xai Lao, Bonnie Antoun, Amanda Jones, Kimberley Mac Donald, Andrew Stershic, Brandon Talamini

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods/

ABSTRACT

Accurate prediction of ductile behavior of structural alloys up to and including failure is essential in component or system failure assessment, which is necessary for nuclear weapons alteration and life extensions programs of Sandia National Laboratories. Modeling such behavior requires computational capabilities to robustly capture strong nonlinearities (geometric and material), rate-dependent and temperature-dependent properties, and ductile failure mechanisms.

This study's objective is to validate numerical simulations of a high-deformation crush of a stainless steel can. The process consists of identifying a suitable can geometry and loading conditions, conducting the laboratory testing, developing a high-quality Sierra/SM simulation, and then drawing comparisons between model and measurement to assess the fitness of the simulation in regards to material model (plasticity), finite element model construction, and failure model. Following previous material model calibration, a J_2 plasticity model with a microstructural BCJ failure model is employed to model the test specimen made of 304L stainless steel.

Simulated results are verified and validated through mesh and mass-scaling convergence studies, parameter sensitivity studies, and a comparison to experimental data. The converged mesh and degree of mass-scaling are the mesh discretization with 140,372 elements, and a mass scaling with a target time increment of 1.0e-6 seconds and time step scale factor of 0.5, respectively. Results from the coupled thermal-mechanical explicit dynamic analysis are comparable to the experimental data. Simulated global force vs displacement (F/D) response predicts key points such as yield, ultimate, and kinks of the experimental F/D response. Furthermore, the final deformed shape of the can and field data predicted from the analysis are similar to that of the deformed can, as measured by 3D optical CMM scans and DIC data from the experiment.

ACKNOWLEDGEMENTS

The authors acknowledge Kevin Connelly for assistance with can validation specimen and fixture drawings and Gwyn Diaz for completing 3D scanning of the deformed cans and converting to step files.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page left blank

ACRONYMS AND DEFINITIONS

Abbreviation	Definition
F/D	Force vs displacement
DIC	Digital Image Correlation
CMM	Coordinate measuring machine

1. INTRODUCTION

Accurate prediction of ductile behavior of structural alloys up to and including failure is essential in component or system failure assessment, which is necessary for nuclear weapons alteration and life extensions programs at Sandia National Laboratories. Modeling such behavior requires computational capabilities to robustly capture strong nonlinearities (geometric and material), rate-dependence, temperature-dependence, and ductile failure mechanisms of the material of interest.

The objective of this study is to validate numerical simulations of a high-deformation crush of a stainless steel can geometry. This process consists of identifying a suitable can geometry and loading conditions, conducting the laboratory testing independent of any simulation effort, developing a high-quality Sierra/SM simulation, and then drawing comparisons between model and measurement. Through this process we intend to assess the fitness of the following components of the Sierra/SM simulation: material model (plasticity), finite element model construction, and failure model.

The material of interest for this study is AISI 304L-VAR, an austenitic stainless steel, with very low carbon content and high corrosion resistance. This is a highly ductile alloy that can withstand severe deformation before tearing (e.g. equivalent plastic strains of 0.9).

The boundary conditions are designed in order to provide a rigorous crushing load to the can geometry, with the intent to promote material tearing. This fosters validation comparisons for Sierra/SM's regularized failure models, such as phase field fracture and nonlocal damage regularization. The crushing-type loading, broadly in compression with elements of buckling, stands in contrast to puncture loadings with localized shear character that have formed recent validation studies [2] and better represents a different class of abnormal mechanical situations. Preliminary simulations were employed to propose the orientation of the loading, finding that concentrating the impact on one "corner" of the can was most likely to cause tearing; nonetheless, the 304L alloy is so ductile that these predictions could not guarantee that tearing would occur.

The validation can crush experiments were performed on a 220 kip capacity MTS 810 servo-hydraulic test frame in the Mechanics of Materials Laboratory at Sandia National Laboratories, California. Experiments were conducted in actuator displacement (stroke) control at one of four monotonic loading rates of 0.01, 0.1, 1, and 5 inches per second. The stroke and test frame 220 kip load cell were recorded during each experiment, but the complex loading angle required an additional three-axis load cell to capture the response of each can to the axial loading and resulting deformation. Stereo optical cameras were used to capture DIC images and Vic3D 9 Correlated Solutions software was used to post-process the images to determine full-field displacement and strain fields. A FLIR 6903sc infrared camera was used to measure full-field temperature in a region of the cans that were tested at the three highest loading rates, where temperature increase was expected.

Despite that none of the cans exhibited sufficient damage to constitute a tear or through-crack, this dataset nevertheless provides a valuable resource for model validation through a challenging loading. This report seeks to document this validation process, including model-measurement comparisons, identification of model successes and failures, and discussion of means of improvement.

This work is the product of a multi-disciplinary team across the laboratories. Xai Lao performed the bulk of the simulations and developed the validation comparisons. Bonnie Antoun provided guidance for the test geometry, designed the test fixtures, and developed and supervised the laboratory testing. Kimberley Mac Donald set up and collected all DIC and FLIR images and performed much of the processing of that data. Amanda Jones completed additional post-processing and merging of the DIC and FLIR data. Gwyneth Diaz completed the 3D optical scans of the

deformed can crush specimens and converted the files to *.stl and *.stp formats. Andrew Stershic and Brandon Talamini conducted the preliminary design simulations and provided guidance for the production simulations and validation analysis.

2. MODEL VALIDATION APPROACH

Complementary to the experiment, a finite element model is built to simulate the physical can crushing process. Simulation results are post-processed for verification and validation. The verification and validation process involves mesh and mass-scaling convergence studies, model parameter sensitivity studies, and comparison of the simulated response to experimental data.

2.1. Develop simulation to compare to experiment

In the experimental configuration, a 304L stainless steel can is fastened at its six perimeter screw tabs to a wedge fixture, as shown in Figure 2-1 (a). This fixture is designed with a 45° angle in one direction, and a 30° in a second direction; this orientation allows the loading cell plate to impact the can on a small corner. The can is fitted around the protrusion of an aluminum can insert (not visible) that is partially seated in the recess of and bolted to the wedge fixture. This provides additional constraint, partially distributing the load from the bolts.

The complementary finite element model consists of key geometric bodies such as the loading top plate, base-plate, can, can insert, and screw tabs, as shown in Figure 2-1 (b); note that only the protruding portion of the can insert is modeled. The following boundary and/or contact conditions are enforced:

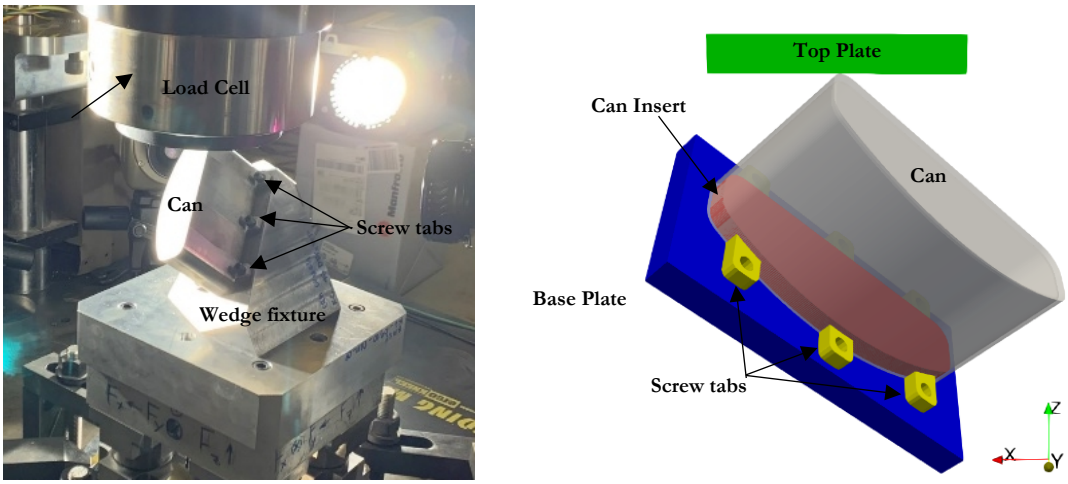
- top plate is constrained to only displace in the vertical direction (i.e. along Z-axis)
- bottom surface of the angled base-plate is held fixed in all directions
- can insert and screw tabs are perfectly connected to the top surface of the angled base-plate through tied contact
- all other contact interactions between each geometric body are modeled as Coulomb frictional contact with a constant frictional coefficient of 0.4

From previous studies, the mechanical behavior of this alloy may be adequately modeled using a rate- and temperature-dependent yield stress, temperature-dependent elastic moduli, and J_2 plasticity with Voce hardening [3] (Sierra/LAMÉ j2_plasticity). To model failure, a scalar damage model based on micro-mechanistic void nucleation and growth is incorporated into the constitutive model [3] (Sierra/LAMÉ bcj_failure). Element death capability is used to remove associated finite elements whenever any integration point reaches the critical damage threshold, set to be 15%. The damage and failure employed is purely local; no regularization is included that could provide a material length scale. For more detailed information on the calibration and selection of the constitutive parameters, please see [2].

This constitutive model is assigned to all geometric bodies, which were meshed with eight-noded hexes with Total Lagrange section, full-integration formulation of degree 3, and a volume-averaged Jacobian. It has been verified that the simulated responses are insensitive to the choice in modeling the can insert as aluminum (as in the experiment), or 304L stainless steel. This is so because in either choice, the can insert experienced very little yielding throughout the loading protocol.

The loading protocol consists of the displacing the top plate downward at the prescribed loading rate (e.g. 5 in/s) in the -Z direction until the can is crushed and a displacement of 1.0 in. is reached, then the loading is reversed at the same rate. This crushing process is simulated through a coupled thermo-mechanical explicit dynamic analysis with appropriate mass scaling.

In the following analyses, a baseline loading rate of 5 in/s is assumed for the simulation, unless otherwise noted. Similarly, simulations shall be assumed to use adiabatic heating (non-conductive) for thermal effects and mass scaling in order reduce run-times.



(a) (b)

Figure 2-1. Comparison between experimental and finite element model set-up: (a) 304L can specimen bolted to angled base plate, and (b) complementary finite element model

2.2. Verification

The verification process consisted of mesh and mass-scaling convergence studies to substantiate the appropriate and consistent simulated response.

2.2.1. Mesh Convergence Study

Four mesh densities, see Table 2-1, each being successively uniformly refined by a factor between 1.5 and 2.0, are used for the mesh convergence study. We note that a constant target timestep was specified for mass scaling for all mesh densities tested. As this results in different degrees of mass scaling for each mesh, this affects the interpretation of overall system response with respect to the influence mesh density alone. Recognizing this subtlety, additional sets of analysis with different target time step for mass scaling were conducted in an attempt to isolate the pure mesh effect; these results were consistent and concluded to the same converged mesh density.

Figure 2-2 shows the global force versus displacement (F/D) response of each analysis, in which the global force is defined as the sum of all nodal reaction forces on the bottom surface of the base-plate and the global Z-displacement is absolute value of the averaged of all nodal Z-displacements on the bottom surface of the top plate (see Figure 2-1 (b)). Although the mesh dependency of the structural response when using a local damage model is well-documented, this dependency was negligible in the simulated responses because the damage growth was minimal, and only a few elements died. Hence, the simulated F/D responses converged with mesh refinement, as shown in Figure 2-2. The converged response (i.e. simulation with a mesh density of 140,372 elements) is comparable to the experimental data; whereas with coarser mesh densities, the simulated responses were much softer due to an inadequacy in resolving the local deformation; this leads to a relatively more dispersed deformation field, and consequently different instances of bulging. To substantiate this claim, we note that deviations between the responses occurred in the loading time-span during which the can was undergoing large deformation and geometry change, and not in the initial loading and unloading periods when there was little-to-no deformation and change in geometry.

Table 2-1. Mesh element densities

Mesh	Element densities
1	204,932
2	140,372
3	65475
4	34794

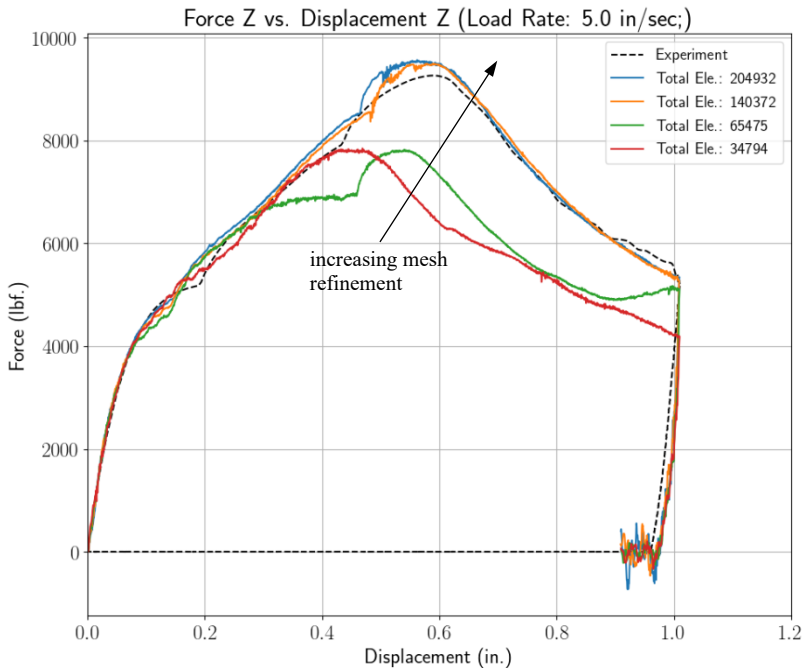


Figure 2-2. Mesh convergence study results: simulated global F/D responses plotted against experimental data

While global responses converged, it does not always guarantee converging local responses. To be more rigorous in this study, local features such as displacement and temperature field, and deformation shape at final loading stage were also compared. At the final deformed and unloaded state of the can, the displacement and temperature fields converged with mesh refinement, as shown in Figure 2-3 (a) and (b), respectively. Additionally, the final deformed shape of the can converged.

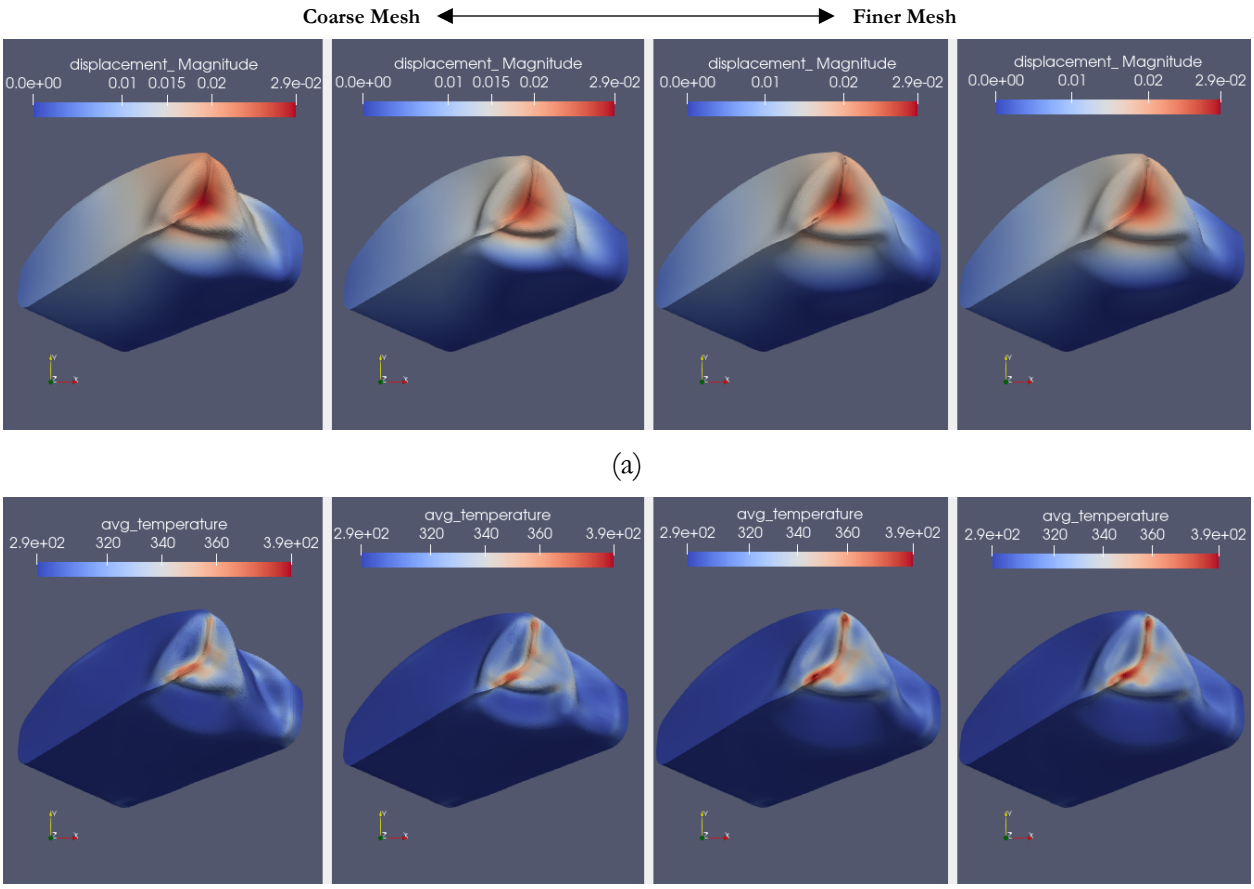


Figure 2-3. Mesh convergence study results: simulated local features (a) displacement field, and (b) average temperature field of final deformed shape of can

2.2.2. Mass Scaling Convergence Study

When using explicit dynamic analysis to model quasi-static or less rapid physical processes, artificial mass scaling is often used to reduce analysis time. In this study, significant mass scaling is required to allow for a tractable simulation time. However, this approach often implies a reduction in simulation prediction quality because the addition of artificial mass may significantly alter the inertial effects of the physical processes. Recognizing this, we seek to quantify the effects that this artificial mass scaling has on the quality of the simulated response through a mass scaling study. Hence, five analyses, each with a different mass scaling applied to all geometric bodies, were completed. The mesh discretization of 140,372 elements were used along with five different degrees of mass scaling that are indirectly specified through the target time increment, see Table 2-2. An additional time-step scale factor of 0.5 is used to ensure stability and no spurious early element death, addressing previously observed stability issues with the Total Lagrange Hex8 element.[1]

Overall, the F/D responses of the analyses showed a convergent trend as mass scaling is decreased, as seen in Figure 2-4, although some sensitivity remains in the baseline (TS = 1.0e-6s) simulation. The simulated response with more mass scaling was stiffer and less comparable to the

[1] Recent Sierra/SM development work has focused on stability of the Total Lagrange Hex8 element, particularly the calculation of the critical timestep in explicit dynamic simulations. Through preliminary testing, the authors believe that these improvements likely address this present instability. This element stability hardening is available in Sierra/SM 5.8.1.

experimental result because of stronger non-physical inertial effects. Complementing the convergent trend in the global responses, the displacement and temperature fields also converged, as shown in Figure 2-5 (a) and (b), respectively. Also, at the last load step of the analysis with the least mass scaling, the deformed shape of the can converged with decreasing mass scaling. Despite best efforts to balance between reduced computation time and prediction accuracy, it is important to note that the use of mass scaling may still affect the overall response, and may contribute to the differences between model and experiment. Considering analysis's duration and accuracy, the analysis with a target time increment of 1.0e-6 seconds is taken as the converged response.

Table 2-2. Target time step increments used in mass scaling convergence study

Analysis	Target Time Step Increment [s]	Total Mass Changed of Can [%]
1	1.0e-5	7.90e5
2	5.0e-6	1.97e5
3	2.0e-6	1.69e4
4	1.0e-6	7.80e3
5	5.0e-7	1.87e3

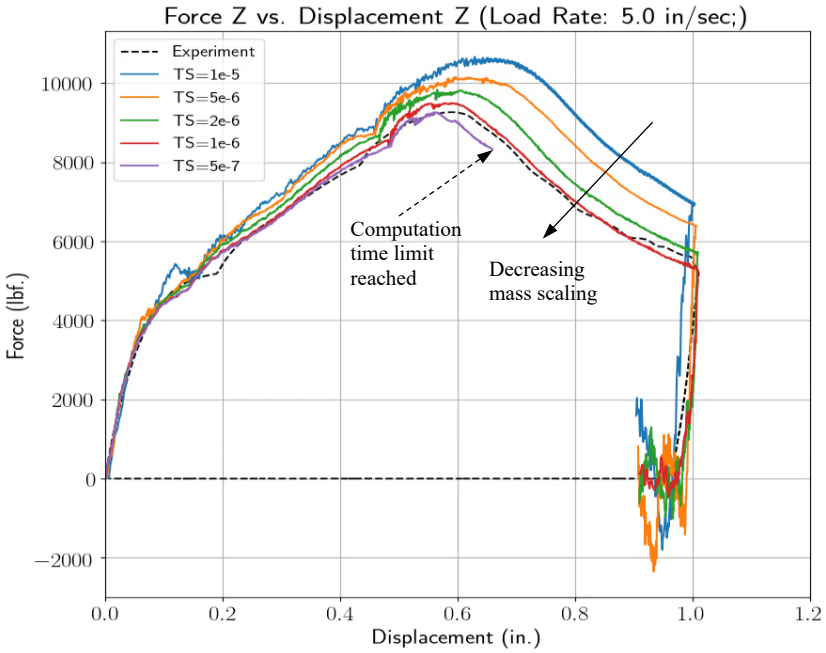


Figure 2-4. Mass scaling study results: simulated F/D responses plotted against experimental data

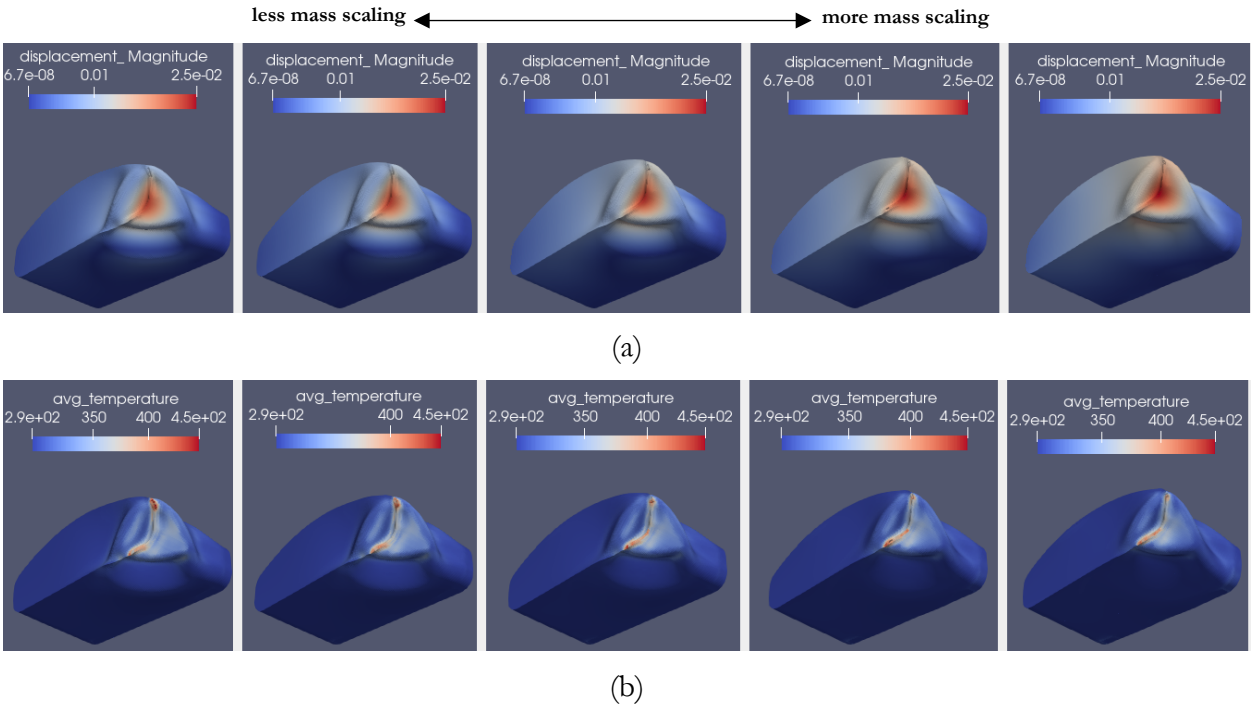


Figure 2-5. Mass scaling study results: simulated local features (a) displacement field, and (b) average temperature field of the deformed shape of can at the last converged load step of the simulation with the least mass scaling

2.3. Comparison of F/D responses

All subsequent coupled thermal-mechanical explicit dynamic analyses use the converged mesh discretization of 140,372 elements and a mass scaling with a target time increment of 1.0e-6 seconds and time step scale factor of 0.5.

The simulated global force vs displacement response in each global X, Y, and Z direction is plotted against the experimental data, as shown in Figure 2-6 (a), (b), and (c) respectively. Despite under-predicting the reaction forces, the simulated F/D response in the X and Y-direction was able to pick up the general trend of the experimental data. The instantaneous jump in reaction force and its oscillation in the onset and tail end of the unloading process was due to dynamic effects that are magnified the mass scaling.

While responses in the X and Y direction moderately captured the experimental data, response in the Z-direction predicted the experimental F/D quite well. The loading, softening, and unloading branches and their respective average slopes of both the simulated and experimental response curves were comparable. Similar to the experiment, the system is being unloaded from a stiffer geometric configuration; hence, for both response curves, the unloading branch was steeper than the initial loading branch. Also, after the system was unloaded, free vibration manifests itself in the oscillation of the simulated response. Furthermore, despite the timing offset, the simulation was able to pick up the first two kinks, yield, and the ultimate point. But, it failed to capture the third kink; this third kink may be an outlier because it did not show up in the response curves of all other experimental replicates loaded at lower rates (see Figure 2-7).

Although the majority of the report focuses primarily on the can crushing process at a loading rate of 5.0 inch per second, other loading rates such as 1.0, 0.1 and 0.01 inch per second were also investigated. The general findings and observations of the simulated results at these loading rates were similar to those concluded from the results at 5.0 in per second loading rate; simulated F/D responses in the Z direction were comparable to the experimental data, as shown in Figure 2-7, and the simulated F/D responses in the other global directions exhibited greater differences.

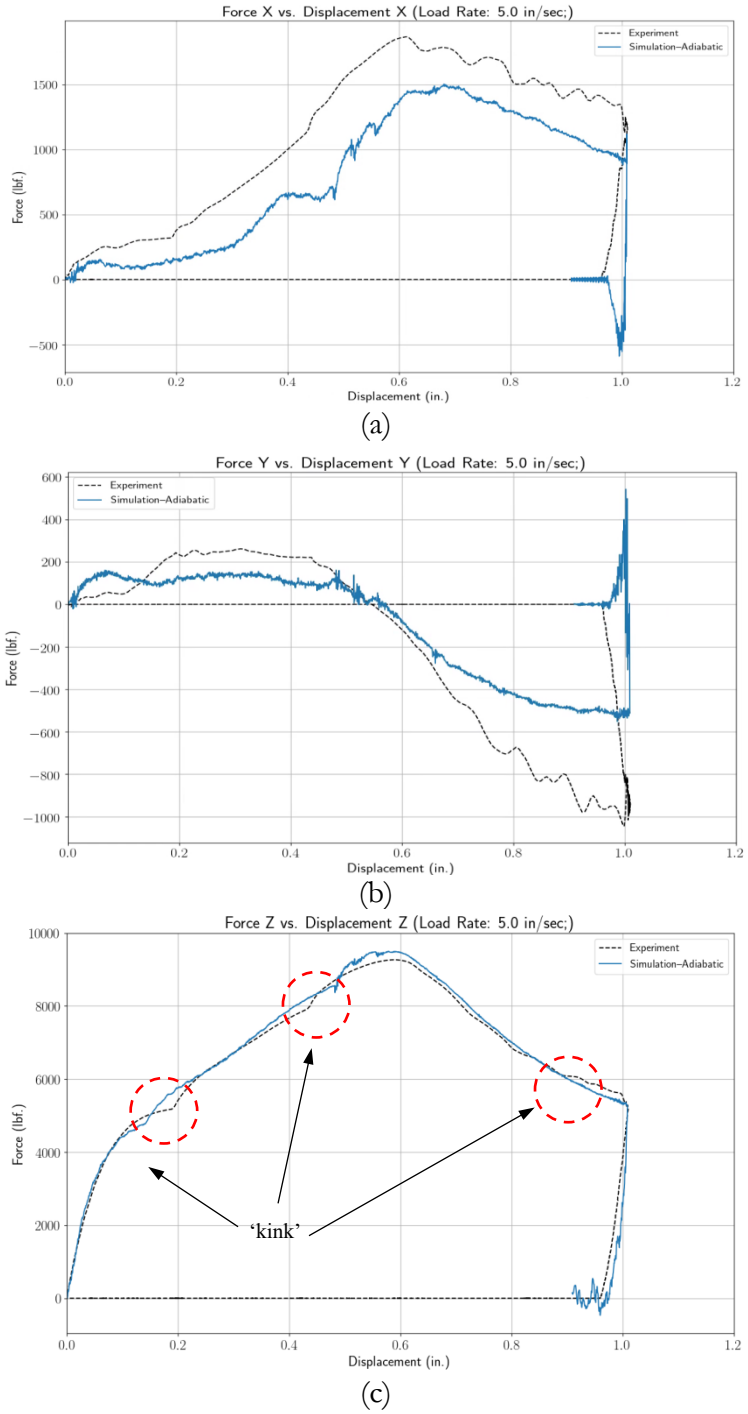


Figure 2-6. 5.0 inch/sec Loading Rate's Simulated F/D response plotted against experimental data in: (a) global X-direction, (b) Y-direction, and (c) Z-direction

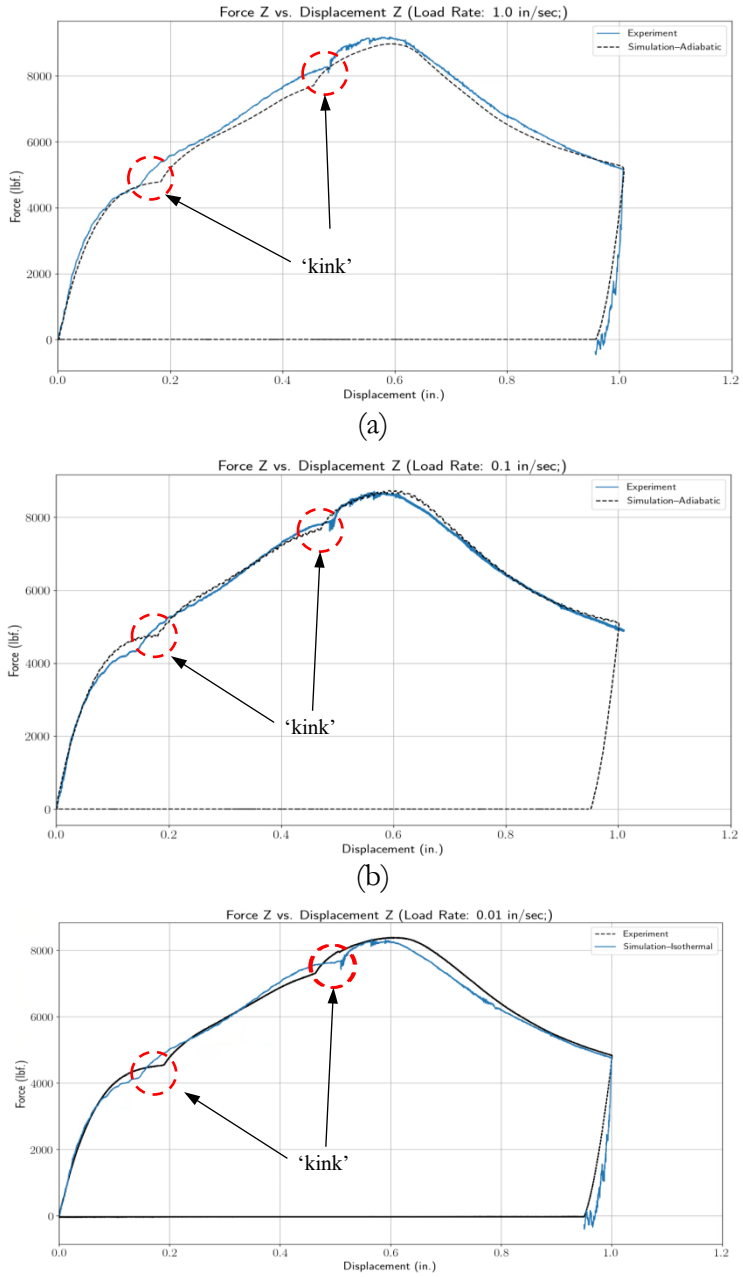


Figure 2-7. Simulated Global Z-direction F/D response plotted against experimental data for loading rate (a) 1 inch/sec, (b) 0.1 inch/sec, and 0.01 inch/sec (isothermal)

2.4. Map F/D features to simulation events

To further elucidate the can crushing process, key features of the F/D response are mapped to simulation events. As illustrated in Figure 2-8:

- (1) → (2): During this process, the can is loaded to yielding and minimally deforms locally in the vicinity of the contact area; thus, the reaction forces gradually increase.

- (2) → (4): As the can continues to yield and deform locally, the stiffness decreases while both the contact area and consequently, the reaction force increases. At (4), the can begins to bulge (local buckling) in the vicinity of the contact area.
- (4) → (7): With similar deformed geometry between state (4) and (5), the relatively sudden increase of contact area results in a sudden increase of reaction force as evident in the first kink point. Then, as the bulge grows with the loading, the stiffness of the current deformed geometry is less than before (i.e. at (1)-(4)). Also, there is a balance between the effects in lessening the stiffness caused by localized deformation, and in increasing the stiffness caused by the increase of contact area; the local buckling stabilizes, resulting in a steady effective positive stiffness.
- (7) → (9): Similar to the process (4) to (5), from (7) to (8), the reaction force suddenly increases due to a sudden increased of contact area. The reaction force gradually increases with contact area, whereas the stiffness slowly decreases. At (9), the can begins to bulge globally at an area remote from the vicinity of the contact area.
- (9) → (11): With continued loading, the can predominantly deforms globally, while there is no net increased in contact area. Consequently, there is not enough 'stiffening' induced by contact area to offset the diminishing stiffness effect caused by the global bulging. Hence, the ultimate point at (9) and a softening branch.
- (11) → (13): With relatively constant deformed current geometry, as the system is unloading, the contact area decreases and consequently, so does the reaction. After the system is fully unloaded during which there was no contact area, the reaction force oscillates due to dynamic effects from free vibration.

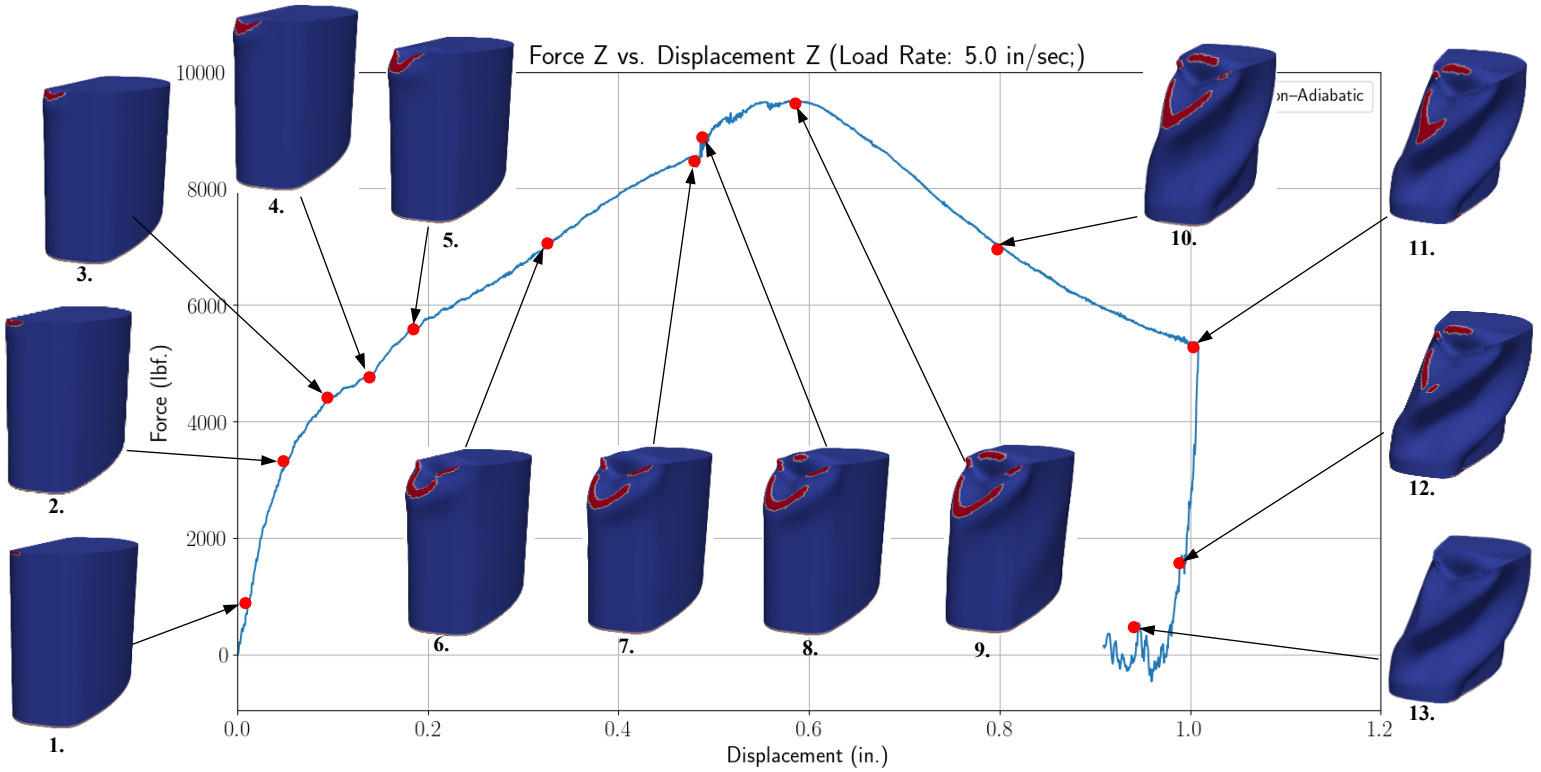


Figure 2-8. Map of F/D features to simulation events, with color indicating contact status (red = in-contact)

2.5. Establish sensitivities to model parameters

Despite comparable F/D responses between the simulation and experiment, the kinks are not aligned. This discrepancy may be caused by the simulated response's sensitivity to modeling efforts. To this end, efforts were made to validate simplifications and assumptions made in modeling the physical process, and to establish sensitivities to model parameters such as initial geometric configuration, treatment of boundary conditions, and material parameters.

The simulated results from the sensitivity study are compared against the baseline (converged simulation) simulated results, and experimental data. Reasonable bounds for model parameters were determined through analyst experience, rather than tangible physical evidence; for example, geometric tolerances tend to be rather small, as the test specimen must remain in alignment, whereas material calibration procedures can often lead to a much larger spread for yield strength and hardening parameters; a large range was considered for friction coefficient, as this is rarely measured directly for the test materials, and wide ranges exist in the literature.

One key factor unexplored in this sensitivity analysis is geometric tolerances in the manufacturing of the can test specimen and differences between idealized geometry and manufactured specimens. The CMM comparisons later in this study address this concern in part, but not perhaps to the level of detail needed to verify geometric tolerances.

2.5.1. Specimen Orientation

By considering slightly different initial specimen orientations with respect to the top loading plate, the simulated response's sensitivity to specimen orientation may be found. Specifically, four different initial specimen orientations are established by rotating the baseline specimen orientation by +/- 1 degree about each axis of rotation, as shown in Figure 2-9.

The simulated global F/D responses are insensitive to initial specimen orientation as evident in Figure 2-10. While the kinks are invariant, the difference in contact area between analyses results in small deviations in the F/D responses between 0.3 and 0.8 inch global displacement interval. Furthermore, the final deformed shapes are comparable.

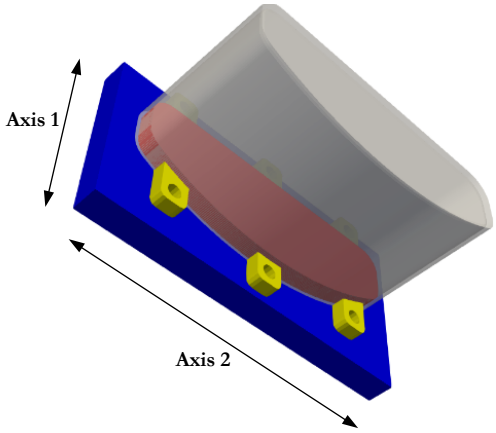
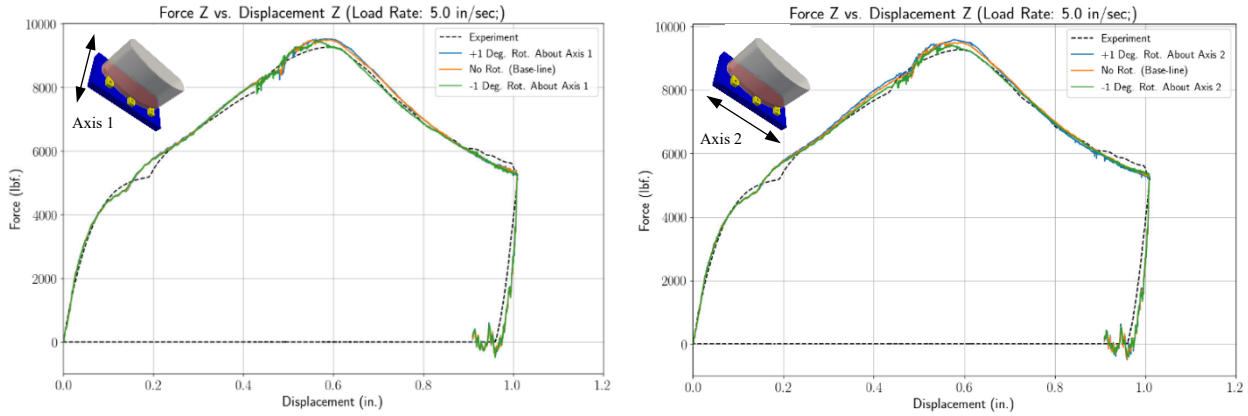


Figure 2-9. Original geometry configuration with illustration of the axes of rotation



(a) (b)

Figure 2-10. F/D response plotted against baseline simulation and experimental data for: (a) specimen rotated about Axis 1, and (b) specimen rotated about Axis 2

2.5.2. Yield Stress

Two initial yield stresses, with +/- 5% difference from the base-line initial yield stress, are considered. As shown in Figure 2-11, the simulated global F/D responses are sensitive to the initial yield stress of the material model. This sensitivity is noticeable in the loading period during which the can experienced significant yielding. On the contrary, since there is minimal-to-no yielding in the initial linear loading and unloading branches, there was no deviation in the response curves.

Furthermore, increasing the initial yield stress shifts both the yield point, and consequently, the response curve upward. For the case with larger initial yield stress, the localized bulge is larger, so it contacts the top loading plate earlier; hence, the second kink occurs sooner.

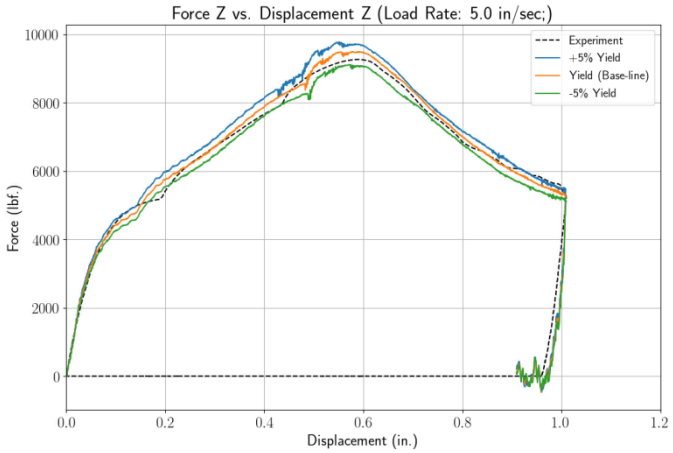


Figure 2-11. Simulated F/D response for model with varied initial yield stress plotted against base-line simulation and experimental data

2.5.3. Voce Hardening Parameters

All analyses use the Voce hardening model, which is represented with the following equation:

$$\bar{\sigma} = \sigma_y + A(1 - \exp(-n\bar{\varepsilon}^p),$$

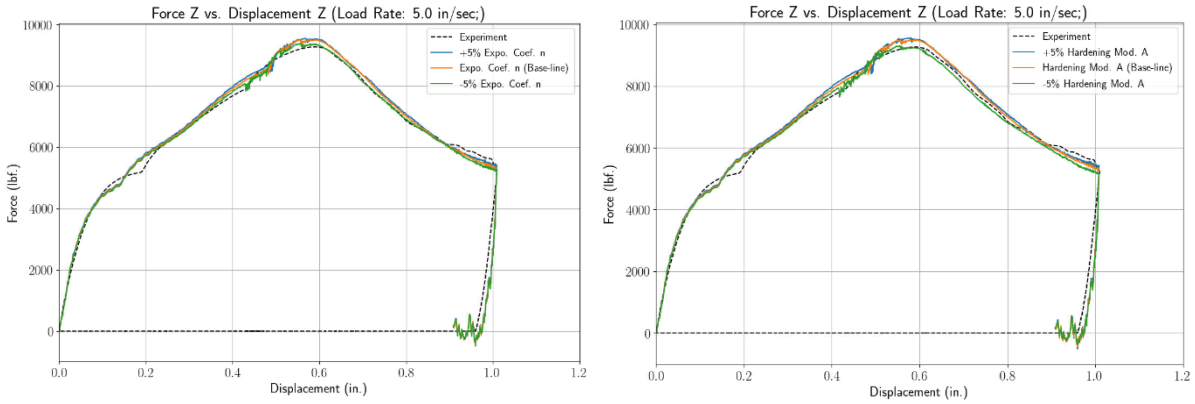
where σ_y is the initial yield stress, A is the hardening modulus, and n is the exponent coefficient. The hardening slope is:

$$\frac{d\bar{\sigma}}{d\bar{\varepsilon}^p} = nA\exp(-n\bar{\varepsilon}^p)$$

Using the base-line n and A, the initial hardening slope is calculated by setting $\bar{\varepsilon}^p$ to be 0. To test the model sensitivity on the initial hardening slope, two other initial hardening slopes with +/- 5% difference from the base-line initial hardening slope are considered. Since the Voce hardening parameters are coupled in the slope expression, two sets of {n, A} are considered: one set consists of a varied n, paired with the base-line A; the other consists of a varied A paired with base-line n.

At this level of variation, the simulated global F/D responses are insensitive to either Voce hardening parameters, but were relatively more sensitive to A than n, shown in Figure 2-12. Aside from the unloading branch, deviations in F/D responses occurred after a global displacement of 0.25 inch. Note that the kinks and yield point are relatively unchanged; whereas, there is a slight sensitivity

to the peak force. Increasing either n or A results in stiffer response due to a larger effective hardening slope.



(a) (b)

Figure 2-12. F/D response plotted against baseline simulation and experimental data for: (a) +/- 5% of Voce exponent coefficient n, and (b) +/- 5% of Voce hardening modulus A

2.5.4. Yield Surface

Besides the baseline simulation with J_2 von Mises yield surface, we explored model-form sensitivity by testing the model with a Hosford yield surface. For only isotropic hardening, the yield surface can be expressed by the same parameters:

$$f(\sigma_{ij},\bar{\varepsilon}^p) = \phi(\sigma_{ij}) - \bar{\sigma}(\bar{\varepsilon}^p) = 0,$$

where ϕ is effective stress and $\bar{\sigma}$ is the current yield stress. The Hosford effective stress is given as:

$$\phi(\sigma_{ij}) = \left[\frac{|\sigma_1 - \sigma_2|^a + |\sigma_2 - \sigma_3|^a + |\sigma_1 - \sigma_3|^a}{2}\right]^{1/a}$$

where a is the yield surface exponent. Hosford yield surfaces reduces to J_2 von Mises yield surface when the yield surface exponent is 2 or 4. For this study, the yield surface exponent is set to 15 resulting in a yield surface position between the Tresca and J_2 von Mises yield surfaces. No other material re-calibration is performed; the properties from the original J_2 von Mises model are used.

The simulated global F/D responses are sensitive to the choice of yield surface used. As shown in Figure 2-13, the analysis with Hosford yield surface results in a 'softer' response. Unlike the first kink, analysis with the Hosford yield surface does a better job at picking up the second kink.

Furthermore, response curve of the analysis with the Hosford yield surface experiences the 2nd kink sooner compared to the base-line analysis, because there is more local bulging at the vicinity of the contact area, and consequently increased of contact area with the loading plate.

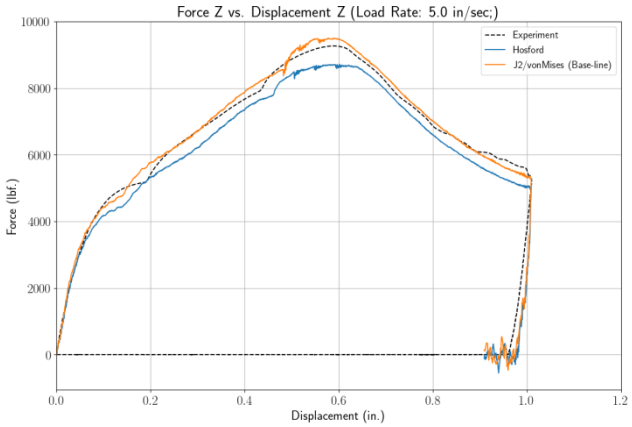


Figure 2-13. F/D response for models with J2 von Mises and Hosford yield surface

2.5.5. Contact Friction Coefficient

Since the model has many contacts between different geometric bodies, it is prudent to assess its sensitivity to contact friction coefficient. To this end, two separate analyses are performed for comparison, each with a different contact friction coefficient (i.e. 0.3 and 0.5).

The simulated global F/D responses of these analyses are compared to the base-line analysis (i.e. simulation with contact friction of 0.4) and the experimental data, as shown in Figure 2-14. Evidently, the global F/D responses are relatively insensitive to the contact friction coefficient. In fact, deviation between the responses only occurs during the 'softening' branch (i.e. between the ultimate and unloading point). Prior to this point, the can predominantly deforms locally in the vicinity of loading contact, resulting in little to no slippage between the geometric bodies. In contrast, during the 'softening' branch, the can exhibits broad buckling, with deformation distant from the loading contact area, resulting in some slippage. This slippage permits sensitivity to the friction coefficient, with a larger friction coefficient resulting in a larger global force. Despite this difference in global response, the final deformed geometry of the analyses are very similar.

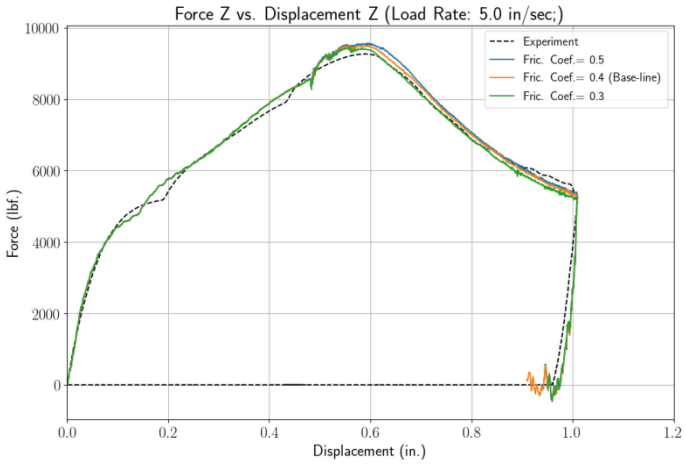


Figure 2-14. F/D response for models with different contact friction coefficients

2.5.6. Boundary condition assumptions (treatment of bolted connections)

The model's sensitivity to the treatment of the bolted connections are also assessed. Figure 2-15 shows the node-sets and surfaces that are used to define different tied boundary conditions of the bolted connections to the base-plate. Three alternative combinations of the treatment of the bolted connections are considered:

- (Baseline) Both can insert bolt and screw tab surfaces tied to base-plate
- Can insert bolt node set and screw tab surfaces tied to base-plate
- Both can insert and screw tabs bolt node-sets tied to base-plate
- Can insert surface and screw tabs bolt node-set tied to base-plate

Clearly, as shown in Figure 2-16, the simulated global F/D responses are sensitive to the treatment of the bolted connections. Comparing the two cluster of the analyses (i.e. screw tabs bolt node-set tied versus screw tab surfaces tied), treatment of the screw tabs boundary condition significantly affects the global response, resulting in a large offset of the curves; whereas, the effects of the can insert boundary conditions to the global response are relatively minimal. In fact, the offset of response curves between the sets are caused by the different boundary conditions of the screw tabs.

Similar to the observation made in the prior section regarding the contact friction coefficient, the deviation in the global response occurs predominantly in the 'softening' branch. In the analyses with screw tabs bolt node-sets tied, the oscillatory responses at the end of the 'softening' branch are believed to be a numerical artifact due element stability[2]. Again, despite the difference in global response, the final deformed shape of the can of these analyses were still comparable.

[2] The authors believe that the recent element stability hardening, as previously mentioned in Section 2.2.2, addresses this present instability.

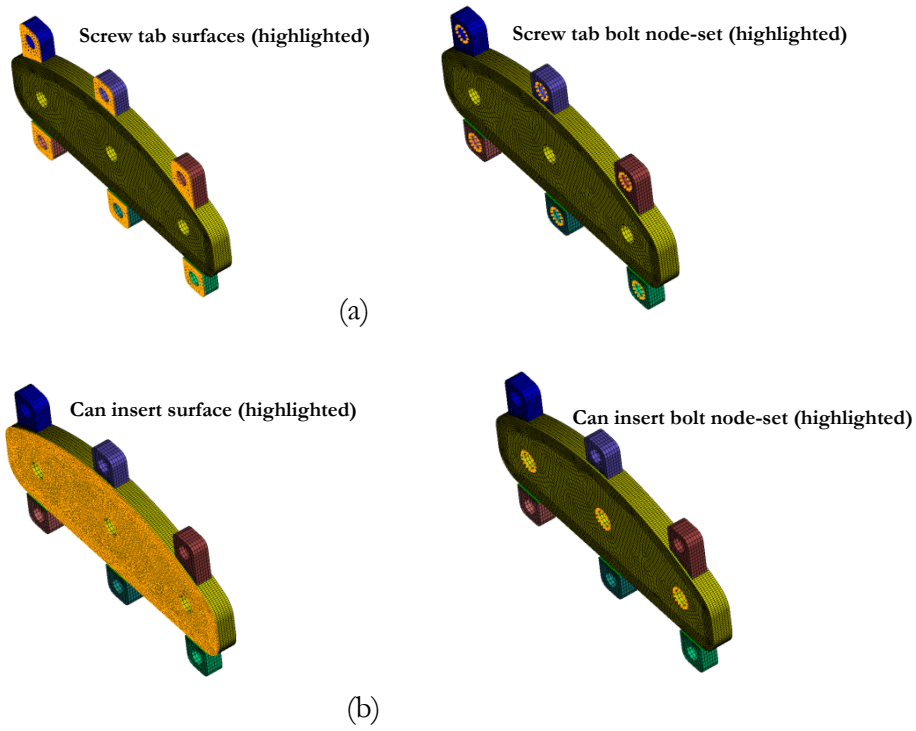


Figure 2-15. Illustration of the node-sets and surfaces that will be tied to base-plate for (a) screw tabs, and (b) can insert

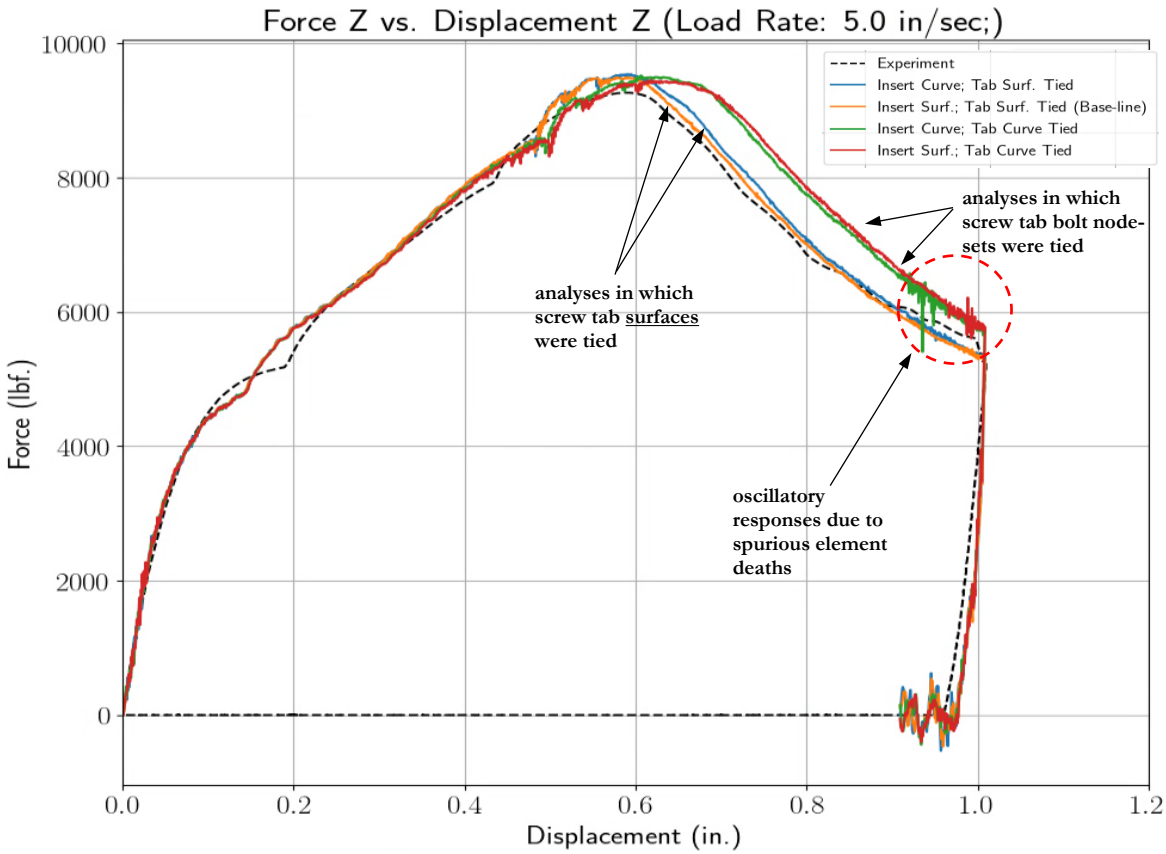


Figure 2-16. F/D response for models with different can insert and screw tab boundary conditions

2.5.7. Bolt Preload Tension

The study of the model's sensitivity to boundary condition treatment of the bolted connections assessed the two extreme cases in the treatment of the screw tabs boundary condition (i.e. one with only the screw tabs bolt node-sets tied and the other with their surfaces tied). An intermediate case with a more realistic treatment of the boundary condition of the screw tabs bolted connections is considered via preload tension capability in Sierra. Note that for this case, the bolted connections at the can inserts are modeled as tied interfaces, since these boundary conditions have minimal effects on the response, as previously shown. Additionally, only isothermal mechanical explicit dynamic analysis is considered for this study since the can's mechanical responses (F/D curve and deformed shape) are comparable to coupled thermal-mechanical analysis.

Three separate isothermal mechanical explicit dynamic analysis, each with a different contact friction coefficient, are performed. In these analyses, the bolted connections at the screw tabs are modeled explicitly with a bolt head and shaft with the interface conditions, as shown in Figure 2-17, and their respective preload tension. During the preload period, the preload tension along each bolt's longitudinal axis is established to a target value which is determined based on actual torque applied in the experiment (40 in-lb). A relaxation period immediately follows the preload period to damp out any inertial effects from the preload process prior to the main loading.

The preload force is determined from a classical relation:

$$T_{[in-lbf]} = KP_{[lbf]}d_{[in]}$$

where T is torque, P is preload force, d is bolt diameter, and K is the nut factor, typically taken as 0.2. This relation is known to be approximate, with error as 35% [4]. The imprecision of this relation is accounted for, in part, by considering a wide range of friction coefficients in the preload study: 0.3-0.5 .

The simulated global F/D responses of these analyses is plotted against the experimental F/D response in Figure 2-18. These responses are able to capture key features (i.e. kinks, ultimate point, softening branch, etc.) and shape of the experimental F/D response curve. Additionally, the trends are consistent with the conclusions drawn from the studies with different screw tabs boundary condition and contact friction coefficient.

Since this bolt preload boundary condition is more representative of the real bolted connection, we expected that the corresponding F/D response should be closer to the experimental data in than previous analyses with simplified bolt treatment. However, this is not the case; the response is stiffer, with a delayed unloading branch, compared to the baseline model. This response is similar to the alternative boundary condition cases using tied nodesets for the screw tabs.

A potential explanation for this discrepancy may be that the preload simulation is in fact more accurate, but combined with the stiffer response imparted by mass scaling (refer to Figure 2-4), it overshoots the experimental data. Therefore, it is possible that the softer response of a simulation without mass scaling combined with the stiffer response of the preloaded bolt connection may counteract each other, providing a response closer to the experimental data. Unfortunately, testing this hypothesis exceeds the limits of computational tractability on Sandia's Capacity Clusters (HPC), so it was not pursued at this time.

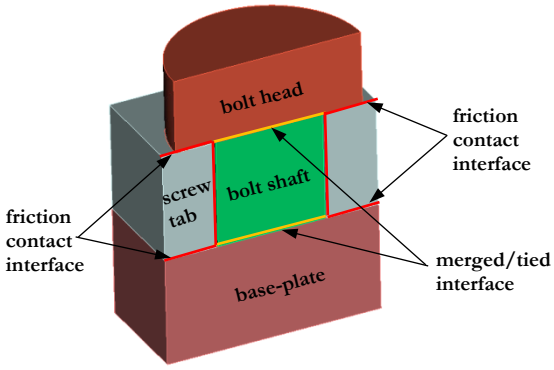


Figure 2-17. Schematic of bolted connection at the screw tabs

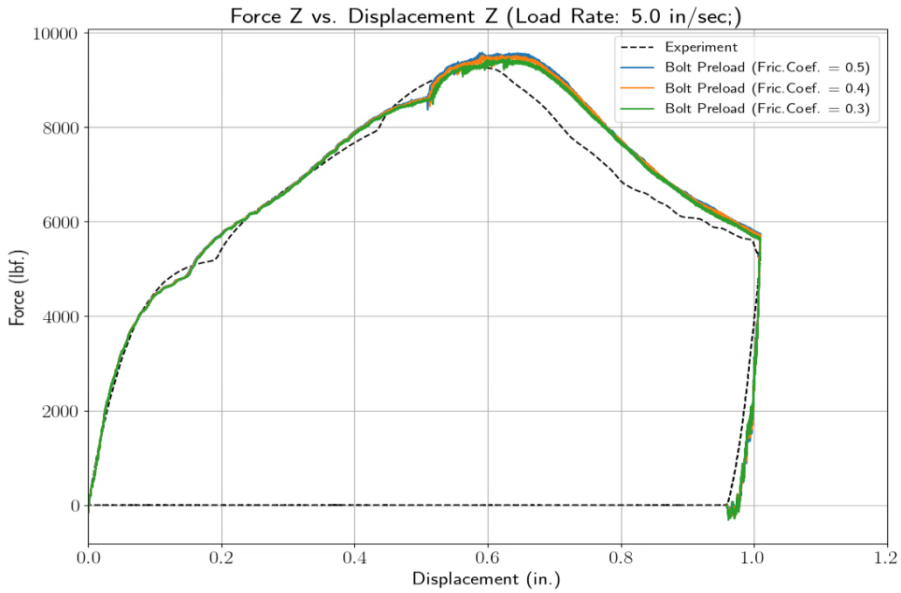


Figure 2-18. F/D response for bolt preload models with different contact friction coefficients

2.6. Comparison of deformed geometry from CMM scans and DIC results

Although the global F/D responses, especially in the global Z-direction, between simulation and experiment were comparable, additional data is available for model validation, namely digital image correlation (DIC) strain/displacement data, infrared temperature field data, and 3D or coordinate measure machine (CMM)-scanned deformed cans. Developing comparisons against these field measurements complements the previous F/D comparisons to provide a more rigorous validation approach. Successful efforts and future endeavors in comparing the deformed shapes are presented this section.

2.6.1. 3D CMM Scan

As a preliminary base-line assessment, the simulated final deformed shape of the can is compared to an image of the experimental final deformed shape of the can. Figure 2-19 illustrates key features common between the experimental and simulated final deformed shape of the can. These key features include:

- the relatively flat triangular 'load bearing' area
- the 'Y-shaped' bulging ridge and top bulging dome, which were brought about by the localized buckling at the vicinity of the 'bearing' area
- the bulging ridge of the can's lateral sides, and the valleys between the this ridge the 'Y-shaped' ridge, which were brought about by the buckling remote from the 'bearing' area
- the 'uplift' of the front unconstrained bottom portion of the can

In contrast, a key difference between the two final deformed shapes of the can is that in the experiment, there was no fracture observed; whereas, in the simulation, full damage develops in a few elements, resulting in their removal by element death. These areas of simulated damage correspond to areas of abrasions, as seen in the experiment. Specifically, higher damage concentrates along the

deformed edge of the can between the top bulge and 'bearing area', at which there are significant deep abrasions along this contour.

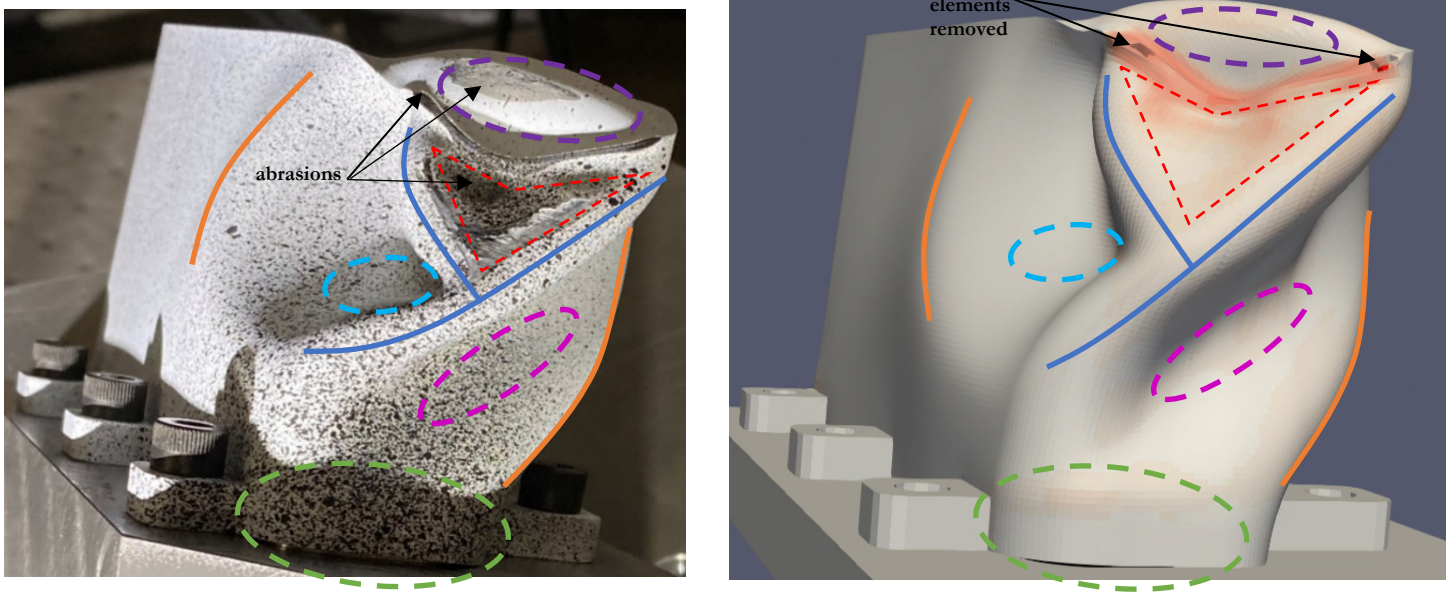
Complementary to comparisons via photos, a more thorough comparison may be achieved by comparing 3D scans of the final deformed shape to that of the simulation. Before actual comparison can be made, both experimental and simulation data are post-processed as illustrated in the flowchart found in Appendix A.1 resulting in a common and consistent representation of the deformed can. This process includes:

1. Raw scan data of many images of the deformed can to allow a full 360 degree representation are stitched together to form a coherent geometry using the scanner software
2. The experimental deformed can geometry is further processed to produce an ACIS solid model (stp format) so that it can be use in Cubit.
3. Exodus file of the simulated deformed geometry of the can is also processed to produce an ACIS solid model (stp format).
4. Both deformed can ACIS solid models are imported into Cubit.
5. Establish a common reference point to facilitate alignment and overlay of both ACIS models since each model references different global coordinate systems.
    a. Ideal reference points are spatial points either on or associated with region of the deformed geometry at which has undergone negligible deformation and displacement that were common to both the experiment and simulation.
    b. Since the farthest screw tab opposite to the loading top plate negligibly deformed and displaced in both the experiment and simulation, the common reference point was taken to be the center of this tab.
6. Use 'Transform' operations in Cubit to align and overlay both ACIS models to the best of an analyst's judgment.
7. Mesh both deformed ACIS solid models and export each ACIS solid model separately as Exodus file so that the final deformed shaped may be compared in Paraview.

As an estimate to provide a 360° comparison, Figure 2-20 (a)-(c) shows the deformed shape of both the simulation and experiment and their overlaid geometry, as viewed from the global ±X, ±Y, and ±Z-axis, respectively. Overall, the simulated final deformed geometry is consistent with the deformed geometry of the experiment.

Despite the successful implementation of this workflow, it is equally important to recognize its drawbacks, specifically in terms of accuracy and computational cost. Intrinsic error in mesh representation of the deformed geometries, especially at severely deformed regions, should be minimized as practical by using higher mesh density at the expense of computational cost. Additionally, errors associated with human subjectivity when aligning and overlaying the deformed geometries should also be considered.

Note: Photo of the deformed shape of the can may be from the experiment with a different loading rate (0.01 inch/sec) and maximum Z-displacement excursion (1.23 inch)



(a) (b)

Figure 2-19. Comparison key features of can's final deformed shape between (a) experiment and (b) simulation

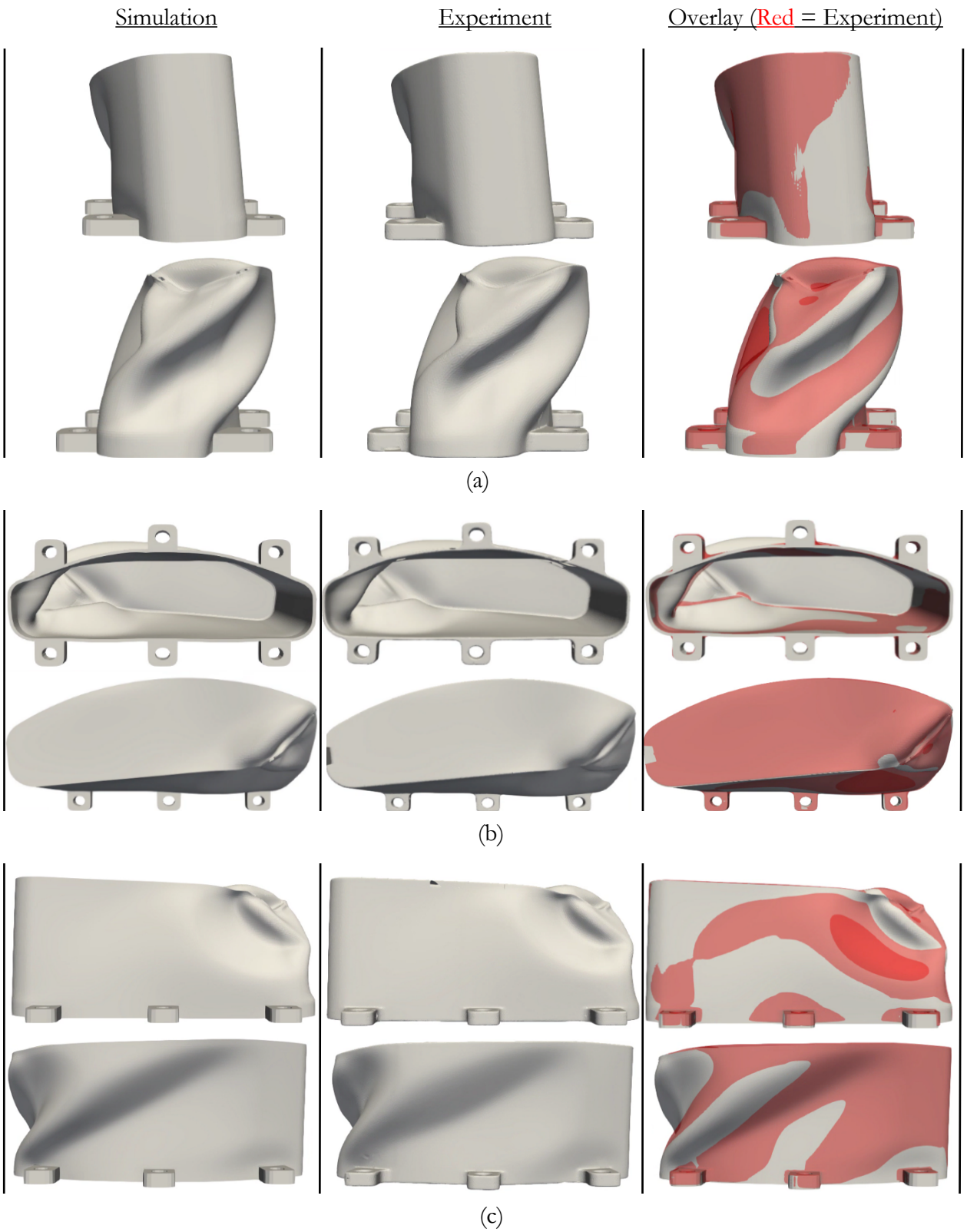


Figure 2-20. Comparison of can's final deformed shape between simulation and experiment as view from (a) ±X-axis, (b) ±Y-axis, and (c) ±Z-axis

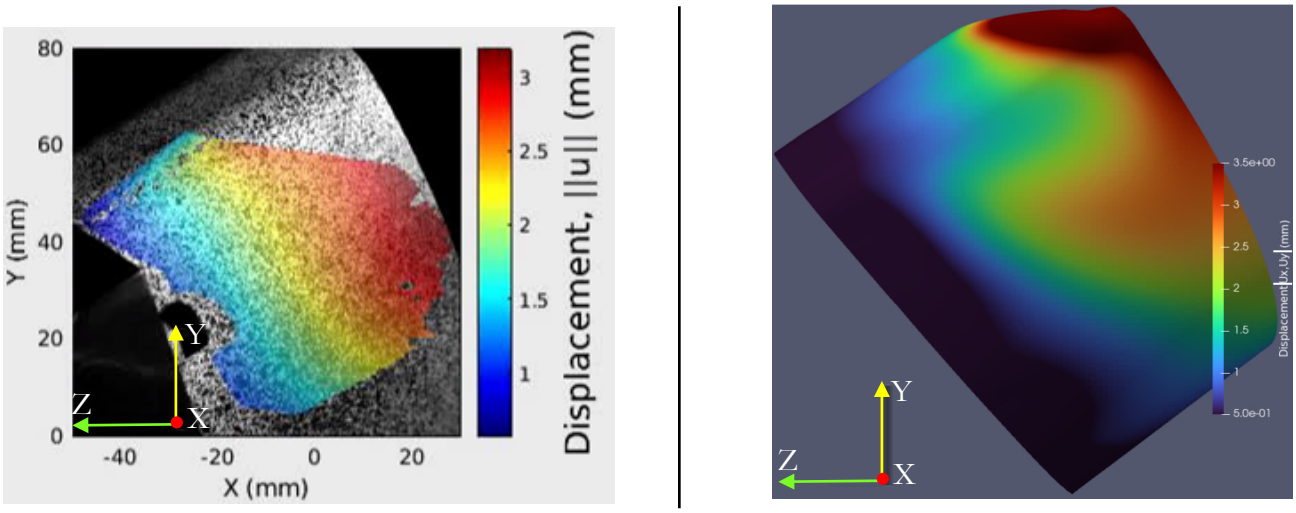
2.6.2. Digital Image Correlation Results

Due to its versatility and relative ease of implementation, digital image correlation (DIC) techniques are often use in mechanical experiments to measure displacement and strain fields. This technique is implemented for the can crushing experiments in this study, with the results shown in Figure 2-21. The DIC results (i.e., displacement and strain fields) are compared to the corresponding field variables from the simulation. Only the X (lateral) and Y (vertical) fields are presented here, but a more comprehensive data comparison is provided in Appendix A.2. It is important to note that the DIC results and the FE model are not perfectly aligned to the same coordinate system, despite our best efforts to do so.

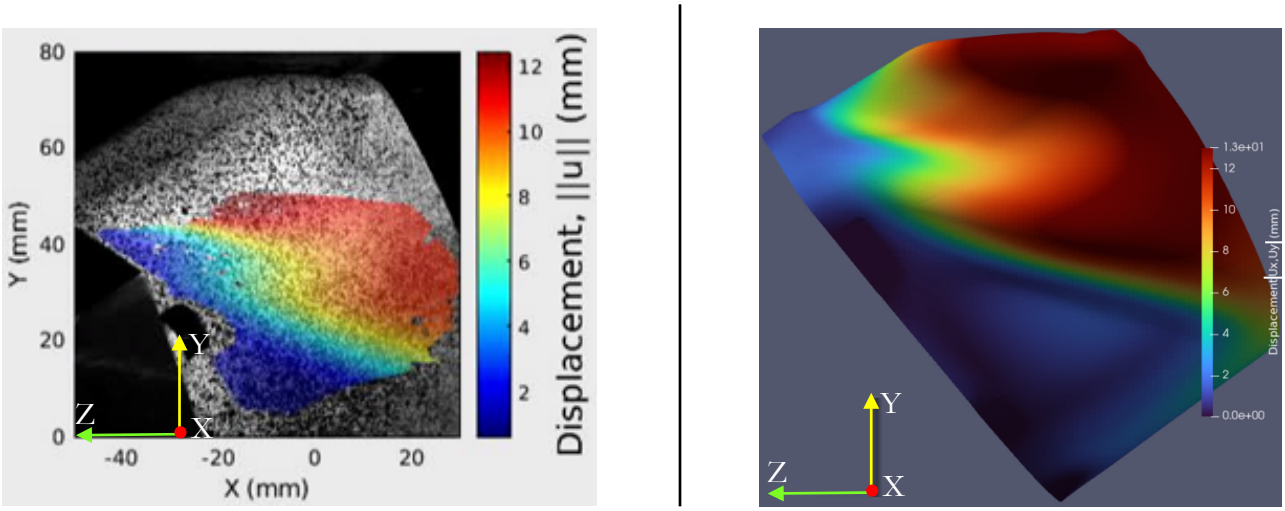
Figure 2-21 shows a comparison between the simulated and DIC displacement magnitude field variable at two different load steps (i.e., at ultimate point and largest displacement excursion of global F/D response). As illustrated in Figure 2-21, the displacement magnitude field variable of the simulation and DIC are qualitatively comparable. Specifically, the contour striations of both the simulation and DIC are similar and occur over a relatively common region of the deformed can. Furthermore, the wavy contours are present in both the DIC and simulated displacement magnitude field. Despite the similarity in contour striations, the size of each contour band and their transition (e.g., gradient) are different between the simulated and DIC result. These differences are more pronounced at the later loading step, as evident in the smaller width and rapid transition of the contours as illustrated in Figure 2-21 (b). We believe that the primary cause of this difference is that the simulation predicted a larger bulging ridge than was captured by DIC; however, the slight differences of the coordinate systems may also contribute.

DIC

Simulation



(a)



(b)

Note: The colored scale bars are the same for experimental DIC and simulated data.

Figure 2-21. Comparison between simulation (right) displacement (magnitude) field variable and experimental (left) DIC displacement (magnitude) response at (a) ultimate point, and (b) largest applied displacement of global F/D response

2.6.3. *Future work*

Thus far, only qualitative comparisons between simulated and experimental results have been established. It will be valuable to also establish some quantitative metrics to measure 'errors' in F/D response and in the final deformed geometry. In either case, these error metrics may be use in optimization scheme for model parametrization study.

Since error metrics for F/D response (i.e. an averaged error sense in term of energy of F/D curve and/or mean squared difference of the F/D data points) are well established, the primarily focus here is to establish an error measure in the final deformed geometry. Particularly for quantifying 'error' in the final deformed geometry, a potential error metric can be established in terms of the 'intersected' volume between the simulated and experimental deformed shapes as demonstrated in Figure 2-21 (a). Following naturally from the "Alignment and Overlaying" process, the total 'intersected' volumes, along with the average volume of the two deformed shapes, may be readily calculated using the Geometry Boolean functions in Cubit, provided that the deformed geometries are in favorable format. Then, the 'intersected' volume is normalized by the average volume of the two deformed shapes; this quantity represented on average, how well the predicted deformed shape fits the true experimental deformed shape. Finally, the percent error is computed by subtracting this quantity from one.

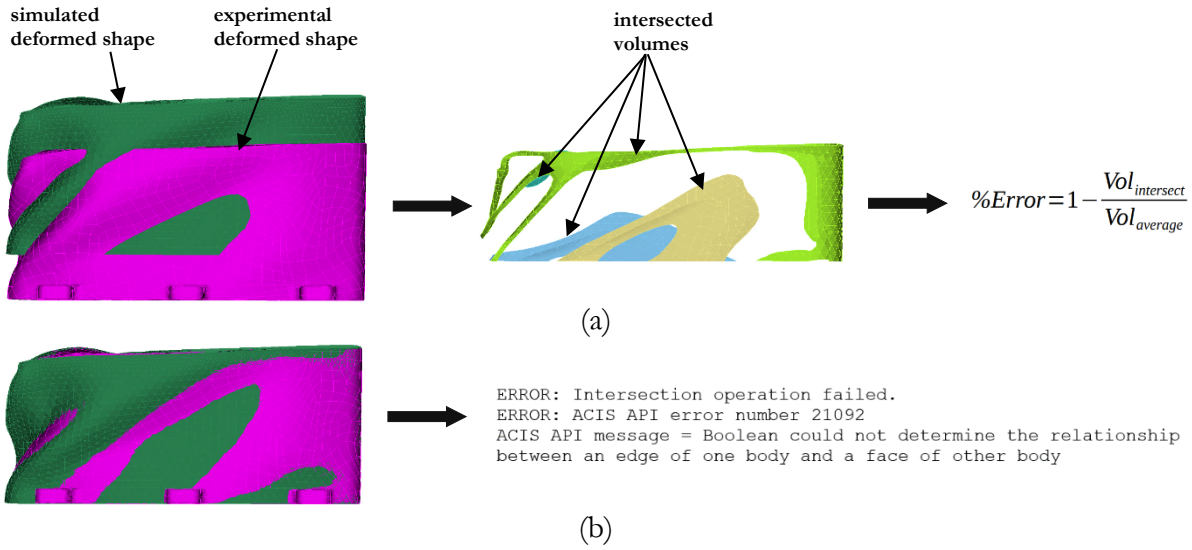


Figure 2-22. Illustration of the error metric calculation process for (a) misaligned, but successful intersected volumes calculation and (b) aligned, but unsuccessful intersected volumes calculation deformed geometries

This proposed process has been prototyped and executed with mixed success. In some cases, the intersection operation failed to get the intersected volume between the experimental and simulated deformed geometries that were aligned because Cubit could not determine the relationship between the geometric entities, as see in Figure 2-21(b). After much exploration and help from the Cubit team, we believe that this failure may be attributed to Cubit's inability to operate on a large number of surfaces in close proximity. Ongoing efforts and other options (such as possibly using Paraview filters to calculate intersected volume) are being explored.

Similarly, in future endeavors, we would like to develop a scheme which enables quantitative comparison between the simulated and DIC field data. This often requires that DIC data and the FE model are aligned with the same coordinate system. Doing so will provide a more direct one-to-one comparison of the data and minimize errors in transforming one data set to a consistent coordinate system of another. Additionally, interpretation of field data comparisons over time is not trivial. To address this challenge, the field data may be reduced to a 1-D field over time by only sampling data at key DIC locations (e.g., at a specific point or along a specific line within the DIC data) and comparing this field data to the interpolated FE model results at corresponding points. Alternately, a 'leveling approach', in which the FE model is used to simulate the DIC experiment, provides a more comprehensive means to compare the field data by enabling the capability to compute the difference between the two field data sets. The suggested approaches as discussed here are not exhaustive. Hence, there is ongoing discussion and collaboration with R&D engineers to determine the most effective and feasible methods for performing comprehensive DIC-FE-model comparisons.

3. CONCLUSION

State-of-the-art computational tools and capabilities are often used to accurately model physical processes and predict ductile behavior up to failure in the component or system failure assessment. In this study, Sierra/SM is used to model the physical process of the crushing of a 304L stainless steel can. The simulated results are verified and validated through mesh and mass-scaling convergence studies, several parameter sensitivity studies, and a comparison between simulated and experimental results.

From the convergence studies, the converged mesh and degree of mass scaling are established to be the mesh discretization with 140,372 elements, and a mass scaling with a target time increment of 1.0e-6 seconds and time step scale factor of 0.5, respectively. Using these parameters, the simulated results from the coupled thermal-mechanical explicit dynamic analysis are comparable to the experimental data. In addition to reproducing the different branches of the response curve (i.e. loading, softening, and unloading), the simulated F/D also predicts key points such as yield, ultimate, and kinks. It is established that the kinks are caused by stiffness increases brought about by the additional contact area; whereas the ultimate point (peak force) is limited by the can's inability to support further loading due to the development of a significant buckling mode. Furthermore, the final deformed geometry of the simulation and experiment is very similar, as verified by comparisons to photography and 3D or CMM scans. They both consist of common features such as bulges caused by buckling, a triangular bearing area, and uplift at the can's base. Favorable comparison of deformation fields between the finite element model and DIC data further supports this validation.

Despite comparable F/D response, final deformed shape between the simulation and experiment, and comparable DIC fields, the kinks are still misaligned; the simulation under -and over-predicted the timing of the first and second kinks, respectively. This discrepancy may be because of the response's sensitivity to modeling parameters and assumptions. To this end, a study was done to establish the response's sensitivity to model parameters such as initial geometric misalignment, treatment of boundary conditions of screw tabs and can insert, and material yielding and hardening parameters. Out of all the model parameters, the F/D response was most sensitive to the initial yield strength, yield surface, and treatment of boundary conditions for the can insert and screw tabs. Similar to decreasing the initial yield strength, changing the yield surface shape (i.e. using Hosford vs von Mises) shifted the response curve downward; hence, resulting in a more compliant response. This compliance may have allowed the simulation with the Hosford yield surface to better pick up the second kink. On the contrary, the kinks' positions were invariant to initial yield stress and the particular contact boundary conditions of the can insert and screw tabs. Furthermore, it was found that the response is more sensitive to the contact boundary condition of the screw tabs than the can insert. Even so, changing the contact boundary condition of the screw tabs from circular bolt node-sets to having the whole surface tied to the base-plate resulted in little change to the response curve. Considering all these findings, it is concluded that the simulation's inability to better capture the kinks is likely affected more by the material constitutive modeling than the model construction choices.

REFERENCES

[1] SIERRA Solid Mechanics Team. Sierra/SolidMechanics 5.4 User's Guide

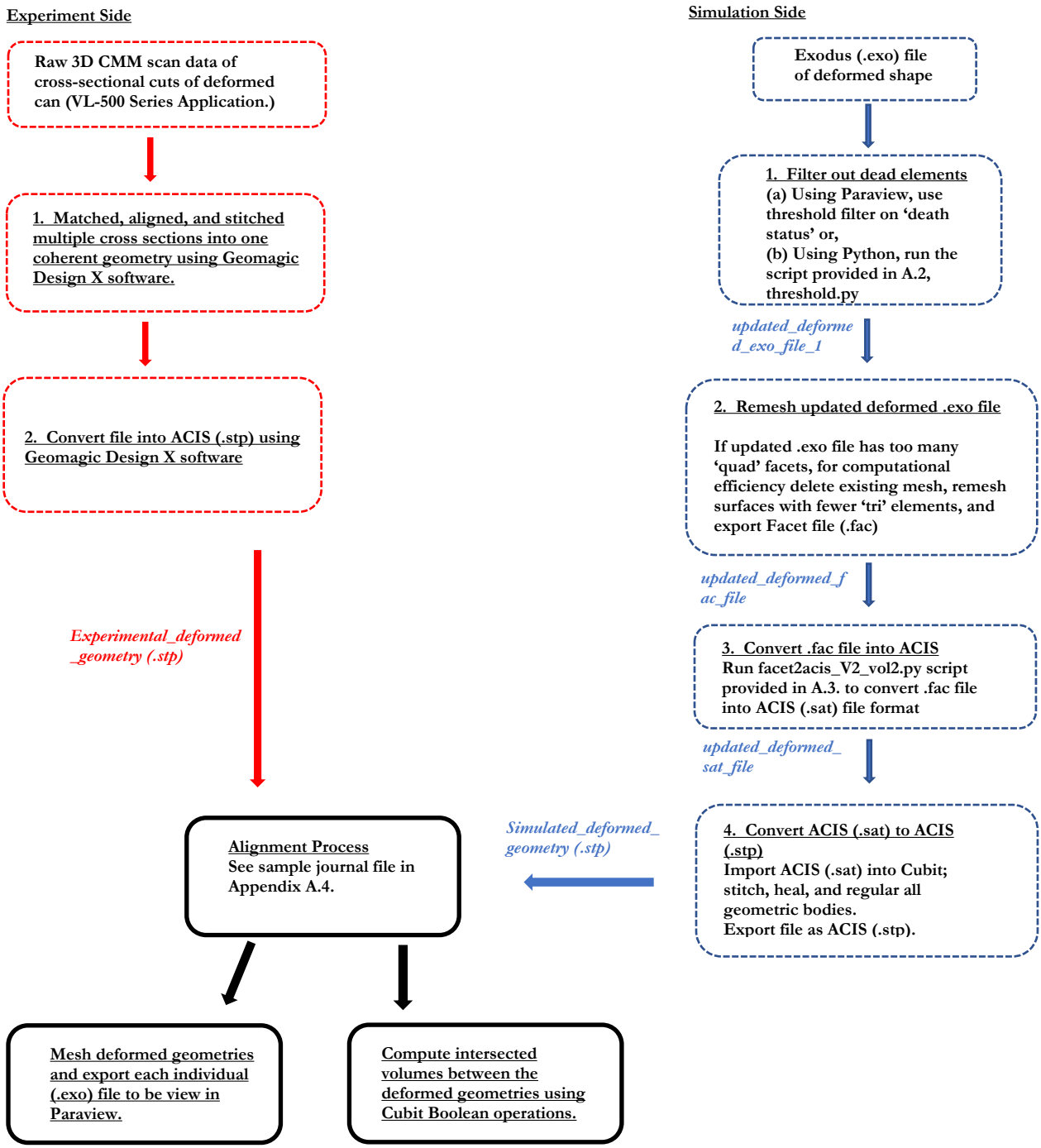
[2] Sharlotte L.B. Kramer, Thomas A. Ivanoff, Amanda R. Jones, Edmundo Corona, Kyle N. Karlson, Andrew J. Stershic, Brandon L. Talamini, Peter W. Grimmer, Brian T. Lester, John M. Emery, Kyle Johnson, Michael K. Neilsen, William M. Scherzinger, Neal Hubbard, and Brett Sanborn. Sandia Fracture Challenge: Puncture. SAND Report, Sandia National Laboratories, 2022 (draft).

[3] Andrew J. Stershic, Brandon L. Talamini, Kyle N. Karlson, Bonnie R. Antoun, Sharlotte L.B. Kramer, Jakob T. Ostien, Vincente Pericoli, and Brian Phung. FY20 ASC P&EM L2 Milestone #7182: Ductile Failure Capability Demonstration for Nuclear Safety Assessment.

[4] Kevin H. Brown, Charles Morrow, Samuel Durbin, and Allen Baca. Guideline for Bolted Joint Design Analysis: Version 1.0. SAND Report, Sandia National Laboratories, 2022. SAND2008-0371.

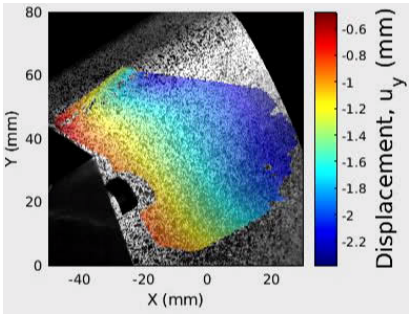
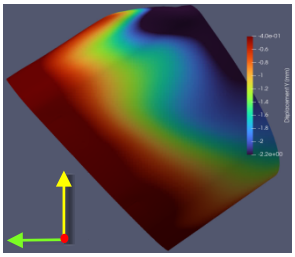
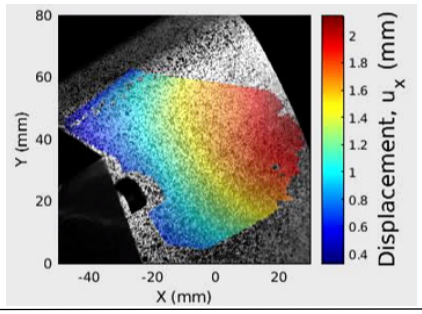
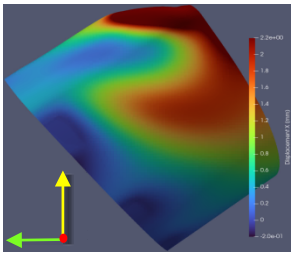
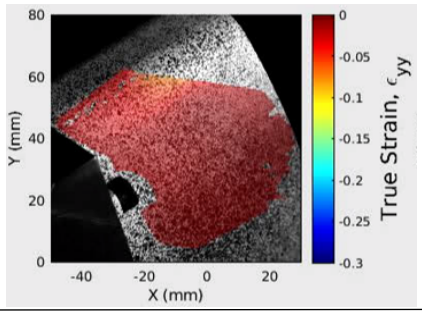
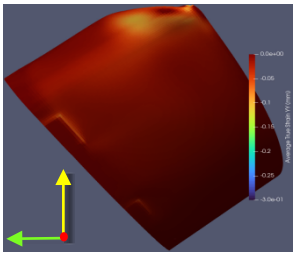
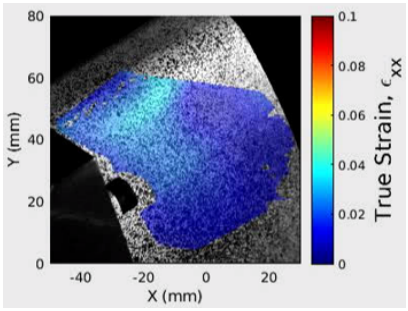
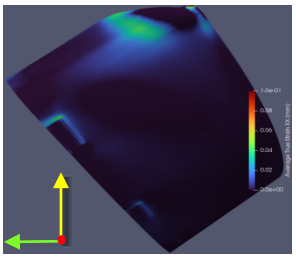
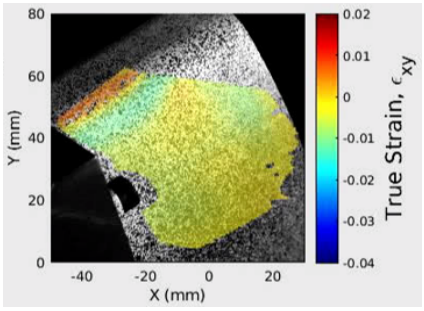
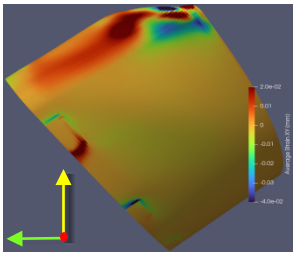
APPENDIX A.

A.1. Flowchart for simulated and experimental can deformed geometries alignment process



A.2. Comparison between simulated response and DIC result

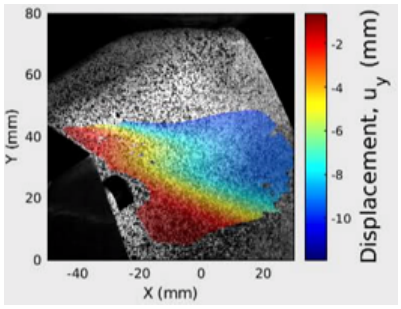
Results are presented for the load step at the ultimate point of F/D response. Note that the colored scale bars are the same for experimental DIC and simulation data.

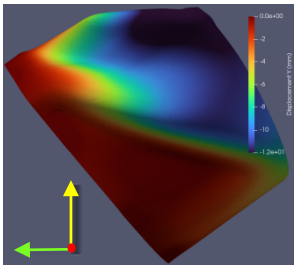
DIC	Simulation
	
	
	
	
	

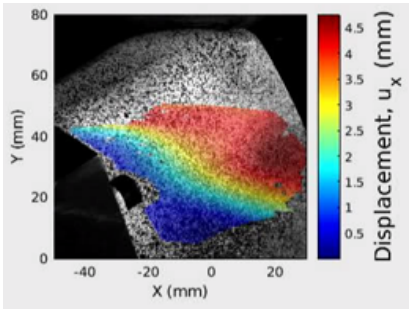
Results are presented for the load step at the largest applied displacement during the test. Note that the colored scale bars are the same for experimental DIC and simulation data.

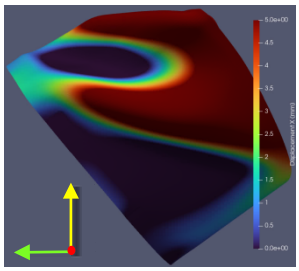
DIC

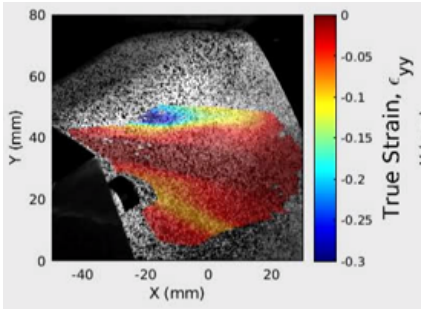
Simulation

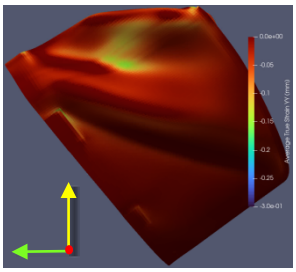


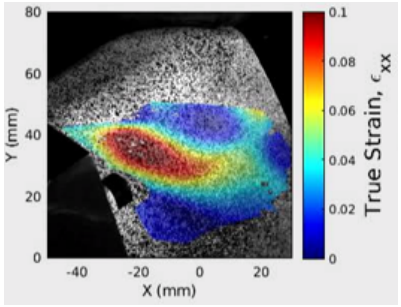


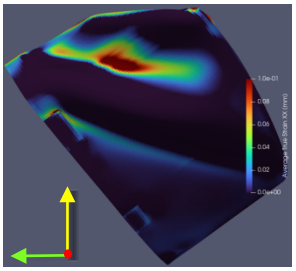


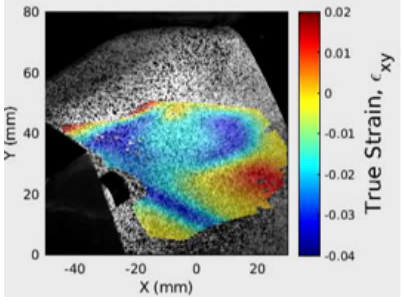


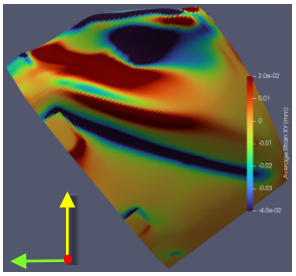












A.3. Python script used to filter out dead elements, threshold.py (Shared by Michael Veilleux, 1542)

```
import sys
import os
from optparse import OptionParser

usage = \
'''
Filter out the elements of specified blocks that are not within the specified bounds
of an element or nodal variable.


Example:
  % python threshold.py --input in.e --output out.e --blocks 2 --type element --
variable death_status --min 0.99 --max 1.01 --step -1

This example:
1. Reads the pre-existing exodus file in.e
2. Applies the filter to block_2 only.  All other blocks do not get any elements
removed.
3. Filters on the element variable death_status; all elements in block_2 with
death_status outside of 0.99 to 1.01 are removed.
4. Applies this filter for the death_status field values at the last time step (--step
-1) in in.e
5. Writes the filtered results to a new exodus file out.e
'''

#
# find and import exodus.py
#
exoPath = os.environ.get('SIERRA_SNTOOLS_PATH')
if not isinstance(exoPath, str):
  raise Exception, \
      "SIERRA_SNTOOLS_PATH not set. Is the sierra module loaded?"
exoPath += "/contrib/testTools/adagio/modules"
if not os.path.isdir(exoPath):
  raise Exception, \
      "Directory searched for exodus.py, %s, does not exist" % exoPath
exoFile = "%s/exodus.py" % exoPath
if not os.path.isfile(exoFile):
  raise Exception, \
      "Expected full path of exodus.py, %s, does not exist" % exoPath
sys.path.append(exoPath)

from exodus import exodus

#
# parser helper functions
#
def isInt(var):
  """
    Determine if variable is an int.
    Works for strings, floats, and int.
  """
```

```
  ok = True
  try:
    num = int(var)
  except ValueError:
    ok = False
  return ok

def isFloat(var):
  """
    Determine if variable can be a float.
    Works for strings, floats, and int.
  """
  ok = True
  try:
    num = float(var)
  except ValueError:
    ok = False
  return ok

def isOptionFlag(arg):
  if arg[:2] == "--" and len(arg) > 2:
    return True
  if isInt(arg) or isFloat(arg):
    return False
  if arg[:1] == "-" and len(arg) > 1 and arg[1] != "-":
    return True
  return False

def varArgCallback(option,opt_str,value,parser,*args,**kwargs):
  """
  handler for an option flag that follows with a variable number of arguments,
  all arguments following the flag must be of the same type
  """
  done = 0
  value = []
  rargs = parser.rargs
  # iterate through the remaining unparsed arguments
  while rargs:
    arg = rargs[0]
    # break when we get to the next option flag
    if isOptionFlag(arg):
      break
    # process an integer argument
    if 'type' in kwargs and kwargs['type'] == 'int':
      value.append(int(arg))
      del rargs[0]
    # process a float argument
    elif 'type' in kwargs and kwargs['type'] == 'float':
      value.append(float(arg))
      del rargs[0]
    # otherwise keep the argument as a string
    else:
      value.append(arg)
      del rargs[0]
  setattr(parser.values,option.dest,value)
```

```
#
# parsing
#
parser = OptionParser(usage=usage)
parser.add_option('-i', '--input', dest='input', type='str', default=None,
                  help='Provide name of input exodus file')
parser.add_option('-o', '--output', dest='output', type='str', default=None,
                  help='Provide name of output exodus file')
parser.add_option('-b', '--blocks', dest='blocks', action='callback', default=None,
                  callback=varArgCallback, callback_kwargs={'type':'int'},
                  help='Provide the blocks on which to apply filter (default is all
blocks)')
parser.add_option('-t', '--type', dest='type', type='str', default=None,
                  help='Provide the variable type to filter (nodal or element)')
parser.add_option('-v', '--variable', dest='variable', type='str', default=None,
                  help='Provide the name of the variable to filter')
parser.add_option('-n', '--min', dest='min', type='float', default=None,
                  help='Provide the minimum bound of the filter')
parser.add_option('-x', '--max', dest='max', type='float', default=None,
                  help='Provide the maximum bound of the filter')
parser.add_option('-s', '--step', dest='step', type='int', default=None,
                  help='Provide the time step of when to apply the filter')
parser.add_option('-k', '--keep-all-steps', dest='keep_all_steps',
                  action='store_true', default=False,
                  help='Copy variables at all time steps (default copies only filter
step)')
(options,args) = parser.parse_args()

#
# check for required command line inputs
#
if options.input == None:
  parser.error("Name of input exodus file required by -i or --input options")
if options.output == None:
  parser.error("Name of output exodus file required by -o or --output options")
if options.type == None:
  parser.error("Filter variable type (nodal or element) required by -t or --type
options")
if options.variable == None:
  parser.error("Filter variable name required by -v or --variable options")
if options.min == None:
  parser.error("Minimum bound of filter required by -n or --min options")
if options.max == None:
  parser.error("Maximum bound of filter required by -x or --max options")
if options.step == None:
  parser.error("Time step to apply the filter required by -s or --step options")

#
# open the input exodus file in read-only mode
#
exo_in = exodus(options.input, "r")

#
# handle the unique request for filtering on the last step
```

```
#
num_steps = exo_in.num_times()
if options.step == -1:
  options.step = num_steps

#
# containers for mesh entities that pass threshold filtering
#
threshold_blk_ids  = []
threshold_elem_ids = set()
threshold_node_ids = set()
threshold_nset_ids = []
threshold_sset_ids = []

#
# fork based on threshold variable type
#
if options.type == 'element':

  # check that the threshold element variable exists
  num_evars = exo_in.get_element_variable_number()
  if num_evars <= 0:
    raise Exception, "Cannot perform threshold operation. " + \
        "No element variables exist in file '%s'" % options.input

  evar_names = exo_in.get_element_variable_names()
  try:
    ithreshold = evar_names.index(options.variable)
  except:
    raise Exception, "Cannot perform threshold operation. Element variable" + \
        "'%s' not found in file '%s'" % (options.variable, options.input)

  # determine which blocks, elements, and nodes meet the threshold
  truth_table = exo_in.get_element_variable_truth_table()
  blk_ids     = exo_in.get_elem_blk_ids()

  if options.blocks == None:
    options.blocks = blk_ids

  elem_id = 1

  for iblk in xrange(len(blk_ids)):
    blk_id = blk_ids[iblk]
    (conn, num_blk_elems, \
     num_elem_nodes) = exo_in.get_elem_connectivity(blk_id)

    if blk_id not in options.blocks:
      for ielem in xrange(num_blk_elems):
        threshold_elem_ids.add(elem_id+ielem)
        threshold_node_ids.update( \
            conn[ielem*num_elem_nodes:(ielem+1)*num_elem_nodes])

      threshold_blk_ids.append(blk_id)
      elem_id += num_blk_elems
      continue
```

```
    blockIsAdded = False

    if truth_table[iblk*num_evars + ithreshold] == False:
      elem_id += num_blk_elems
      continue

    evar_vals = \
        exo_in.get_element_variable_values(blk_id, \
        options.variable, options.step)
    for ielem in xrange(num_blk_elems):
      if evar_vals[ielem] >= options.min and \
         evar_vals[ielem] <= options.max:
        blockIsAdded = True
        threshold_elem_ids.add(elem_id+ielem)
        threshold_node_ids.update( \
            conn[ielem*num_elem_nodes:(ielem+1)*num_elem_nodes])

    if blockIsAdded == True:
      threshold_blk_ids.append(blk_id)

    elem_id += num_blk_elems

elif options.type == 'nodal':

  # check that the threshold nodal variable exists
  num_nvars = exo_in.get_node_variable_number()
  if num_nvars <= 0:
    raise Exception, "Cannot perform threshold operation. " + \
        "No nodal variables exist in file '%s'" % options.input

  nvar_names = exo_in.get_node_variable_names()
  try:
    ithreshold = nvar_names.index(options.variable)
  except:
    raise Exception, "Cannot perform threshold operation. Nodal variable" + \
        "'%s' not found in file '%s'" % (options.variable, options.input)

  # determine which nodes meet the threshold
  nvar_vals = \
      exo_in.get_node_variable_values(options.variable, options.step)
  for inode in xrange(exo_in.num_nodes()):
    if nvar_vals[inode] >= options.min and \
       nvar_vals[inode] <= options.max:
      threshold_node_ids.add(inode+1)

  blk_ids = exo_in.get_elem_blk_ids()

  if options.blocks != None:
    for blk_id in blk_ids:
      if blk_id in options.blocks:
        continue
      (conn, num_blk_elems, \
       num_elem_nodes) = exo_in.get_elem_connectivity(blk_id)
      for node_id in conn:
```

```
        threshold_node_ids.add(node_id)

  # determine which blocks and elements meet the threshold
  elem_id = 1

  for iblk in xrange(len(blk_ids)):
    blk_id = blk_ids[iblk]
    (conn, num_blk_elems, \
     num_elem_nodes) = exo_in.get_elem_connectivity(blk_id)
    blockIsAdded = False

    for ielem in xrange(num_blk_elems):
      if len(threshold_node_ids.intersection(set( \
          conn[ielem*num_elem_nodes:(ielem+1)*num_elem_nodes]))) == \
          num_elem_nodes:
        blockIsAdded = True
        threshold_elem_ids.add(elem_id+ielem)

    if blockIsAdded == True:
      threshold_blk_ids.append(blk_id)

    elem_id += num_blk_elems

else:
  raise Exception, "Invalid threshold variable type '%s'" % options.type

#
# determine which nodesets to keep
#
if exo_in.num_node_sets() > 0:
  nset_ids = exo_in.get_node_set_ids()
  for ns_id in nset_ids:
    if threshold_node_ids.intersection( \
        set(exo_in.get_node_set_nodes(ns_id))):
      threshold_nset_ids.append(ns_id)

#
# determine which sidesets to keep
#
if exo_in.num_side_sets() > 0:
  sset_ids = exo_in.get_side_set_ids()
  for ss_id in sset_ids:
    (elem_list, side_list) = exo_in.get_side_set(ss_id)
    if threshold_elem_ids.intersection(set(list(elem_list))):
      threshold_sset_ids.append(ss_id)

#
# open the output exodus file in write mode
#
if os.path.isfile(options.output):
  os.remove(options.output)

exo_out = exodus( options.output,          \
    mode        = "w",                     \
    array_type  = 'ctype',                 \
```

```
    title       = exo_in.title(),          \
    numDims     = exo_in.num_dimensions(), \
    numNodes    = len(threshold_node_ids), \
    numElems    = len(threshold_elem_ids), \
    numBlocks   = len(threshold_blk_ids),  \
    numNodeSets = len(threshold_nset_ids), \
    numSideSets = len(threshold_sset_ids) )

#
# transfer QA and information records
#
exo_out.put_qa_records( exo_in.get_qa_records() )
exo_out.put_info_records( exo_in.get_info_records() )

#
# transfer nodal coordinate names
#
exo_out.put_coord_names( exo_in.get_coord_names() )

#
# transfer nodal coordinates
#
threshold_node_ids = list(threshold_node_ids)
threshold_node_ids.sort()

(x, y, z) = exo_in.get_coords()

threshold_x = []
threshold_y = []
threshold_z = []

old_to_new_node_ids = {}
new_node_id = 1

for node_id in threshold_node_ids:
  threshold_x.append(x[node_id-1])
  threshold_y.append(y[node_id-1])
  threshold_z.append(z[node_id-1])

  old_to_new_node_ids[node_id] = new_node_id

  new_node_id += 1

exo_out.put_coords(threshold_x, threshold_y, threshold_z)

#
# transfer node ID map
#
id_map = exo_in.get_node_id_map()

threshold_id_map = []
for node_id in threshold_node_ids:
  threshold_id_map.append(id_map[node_id-1])

exo_out.put_node_id_map(threshold_id_map)
```

```
#
# transfer element blocks
#
elem_id = 1
blk_ids = exo_in.get_elem_blk_ids()

for blk_id in blk_ids:
  (elem_type, num_blk_elems, \
   num_elem_nodes, num_elem_attrs) = exo_in.elem_blk_info(blk_id)

  blk_elem_ids = set()
  for ielem in xrange(num_blk_elems):
    blk_elem_ids.add(elem_id)
    elem_id += 1

  if blk_id not in threshold_blk_ids:
    continue

  threshold_blk_elem_ids = threshold_elem_ids.intersection(blk_elem_ids)
  threshold_num_blk_elems = len(threshold_blk_elem_ids)

  if threshold_num_blk_elems == 0:
    continue

  exo_out.put_elem_blk_info(blk_id, elem_type, \
      threshold_num_blk_elems, num_elem_nodes, num_elem_attrs)

  blk_elem_ids = list(blk_elem_ids)
  blk_elem_ids.sort()

  threshold_blk_elem_ids = list(threshold_blk_elem_ids)
  threshold_blk_elem_ids.sort()

  blk_elem_id_map = []
  jelem = 0
  for ielem in xrange(num_blk_elems):
    if jelem < threshold_num_blk_elems and \
       blk_elem_ids[ielem] == threshold_blk_elem_ids[jelem]:
      blk_elem_id_map.append(threshold_blk_elem_ids[jelem])
      jelem += 1
    else:
      blk_elem_id_map.append(-1)

  (conn, num_blk_elems, \
   num_elem_nodes) = exo_in.get_elem_connectivity(blk_id)
  threshold_conn = []
  for ielem in xrange(num_blk_elems):
    if blk_elem_id_map[ielem] > 0:
      for iconn in xrange(ielem*num_elem_nodes, (ielem+1)*num_elem_nodes):
        threshold_conn.append(old_to_new_node_ids[conn[iconn]])
  exo_out.put_elem_connectivity(blk_id, threshold_conn)

  if num_elem_attrs > 0:
    exo_out.put_element_attribute_names(blk_id, \
```

```
        exo_in.get_element_attribute_names(blk_id))
    attr = exo_in.get_elem_attr(blk_id)
    threshold_attr = []
    for ielem in xrange(num_blk_elems):
      if blk_elem_id_map[ielem] > 0:
        threshold_attr += attr[ielem*num_elem_attrs:(ielem+1)*num_elem_attrs]
    exo_out.put_elem_attr(blk_id, threshold_attr)

  elem_props = exo_in.get_element_property_names()
  for elem_prop in elem_props:
    prop_val = exo_in.get_element_property_value(blk_id, elem_prop)
    if elem_prop == "ID" and prop_val == blk_id:
      continue
    else:
      exo_out.put_element_property_value(blk_id, elem_prop, prop_val)

  exo_out.put_elem_blk_name(blk_id, exo_in.get_elem_blk_name(blk_id))

#
# transfer element ID map
#
threshold_elem_ids = list(threshold_elem_ids)
threshold_elem_ids.sort()

id_map = exo_in.get_elem_id_map()

old_to_new_elem_ids = {}
new_elem_id = 1

threshold_id_map = []
for elem_id in threshold_elem_ids:
  threshold_id_map.append(id_map[elem_id-1])

  old_to_new_elem_ids[elem_id] = new_elem_id

  new_elem_id += 1

exo_out.put_elem_id_map(threshold_id_map)

#
# transfer node sets
#
threshold_node_ids = set(threshold_node_ids)

if len(threshold_nset_ids) > 0:

  threshold_nset_ids.sort()

  for ns_id in threshold_nset_ids:
    (num_set_nodes, num_set_dfs) = exo_in.get_node_set_params(ns_id)
    node_set_nodes               = exo_in.get_node_set_nodes(ns_id)

    truths = []
    new_set_nodes = []
    set_size = 0
```

```
    for old_node_id in node_set_nodes:
      if old_node_id in threshold_node_ids:
        new_set_nodes.append(old_to_new_node_ids[old_node_id])
        truths.append(True)
        set_size += 1
      else:
        truths.append(False)

    exo_out.put_node_set_params( ns_id, set_size, min(num_set_dfs, set_size) )
    exo_out.put_node_set( ns_id, new_set_nodes )

    if num_set_dfs > 0:
      set_dfs = exo_in.get_node_set_dist_facts(ns_id)
      new_set_dfs = []
      for idf in xrange(num_set_nodes):
        if truths[idf]:
          new_set_dfs.append(set_dfs[idf])
      exo_out.put_node_set_dist_fact( ns_id, new_set_dfs )

    exo_out.put_node_set_name( ns_id, exo_in.get_node_set_name(ns_id) )

    ns_props = exo_in.get_node_set_property_names()
    for ns_prop in ns_props:
      prop_val = exo_in.get_node_set_property_value( ns_id, ns_prop )
      if ns_prop == "ID" and prop_val == ns_id:
        continue
      else:
        exo_out.put_node_set_property_value( ns_id, ns_prop, prop_val )

#
# transfer side sets
#
threshold_elem_ids = set(threshold_elem_ids)

next_ss_id = 1

ss_names = set()

if len(threshold_sset_ids) > 0:

  threshold_sset_ids.sort()

  next_ss_id = threshold_sset_ids[-1] + 1

  for ss_id in threshold_sset_ids:
    (num_set_sides, num_set_dfs) = exo_in.get_side_set_params(ss_id)
    (elem_list, side_list)       = exo_in.get_side_set(ss_id)

    new_elem_list = []
    new_side_list = []
    new_num_set_dfs = 0
    set_size = 0
    for iside in xrange(num_set_sides):
      old_elem_id = elem_list[iside]
      if old_elem_id in threshold_elem_ids:
```

```
        new_elem_list.append(old_to_new_elem_ids[old_elem_id])
        new_side_list.append(side_list[iside])
        set_size += 1

    new_num_set_dfs = 0
    new_set_dfs = []
    if num_set_dfs > 0:
      set_dfs = exo_in.get_side_set_dist_fact()
      node_list = exo_in.get_side_set_node_list(ss_id)
      for inode in xrange(len(node_list)):
        if node_list[inode] in threshold_node_ids:
          new_set_dfs.append(set_dfs[inode])
          new_num_set_dfs += 1

    exo_out.put_side_set_params( ss_id, set_size, new_num_set_dfs )
    exo_out.put_side_set( ss_id, new_elem_list, new_side_list )

    if new_num_set_dfs > 0:
      exo_out.put_side_set_dist_fact( ss_id, new_set_dfs )

    ss_name = exo_in.get_side_set_name(ss_id)
    ss_names.add(ss_name)
    exo_out.put_side_set_name( ss_id, ss_name )

    ss_props = exo_in.get_side_set_property_names()
    for ss_prop in ss_props:
      prop_val = exo_in.get_side_set_property_value( ss_id, ss_prop )
      if ss_prop == "ID" and prop_val == ss_id:
        continue
      else:
        exo_out.put_side_set_property_value( ss_id, ss_prop, prop_val )

#
# transfer time steps
#
times = exo_in.get_times()
copy_steps = []
num_copy_steps = 0
if options.keep_all_steps:
  for istep in xrange(num_steps):
    exo_out.put_time( istep+1, times[istep] )
    copy_steps.append(istep+1)
    num_copy_steps += 1
else:
  exo_out.put_time( 1, times[options.step-1])
  copy_steps = [options.step]
  num_copy_steps = 1

#
# transfer global variables
#
num_gvars = exo_in.get_global_variable_number()
if num_gvars > 0:
  exo_out.set_global_variable_number(num_gvars)
```

```
  gvar_names = exo_in.get_global_variable_names()
  for iname in xrange(num_gvars):
    exo_out.put_global_variable_name( gvar_names[iname], iname+1 )

  for istep in xrange(num_copy_steps):
    exo_out.put_all_global_variable_values( istep+1, \
        exo_in.get_all_global_variable_values(copy_steps[istep]) )

#
# transfer nodal variables
#
threshold_node_ids = list(threshold_node_ids)
threshold_node_ids.sort()

threshold_inodes = [node_id-1 for node_id in threshold_node_ids]

num_nvars = exo_in.get_node_variable_number()
if num_nvars > 0:
  exo_out.set_node_variable_number( num_nvars )

  nvar_names = exo_in.get_node_variable_names()
  for ivar in xrange(num_nvars):
    exo_out.put_node_variable_name(nvar_names[ivar], ivar+1)

    for istep in xrange(num_copy_steps):
      vals = exo_in.get_node_variable_values( nvar_names[ivar], copy_steps[istep] )
      new_vals = [vals[inode] for inode in threshold_inodes]
      exo_out.put_node_variable_values( nvar_names[ivar], istep+1, new_vals )


#
# transfer element variables
#
num_evars = exo_in.get_element_variable_number()
if num_evars > 0:
  exo_out.set_element_variable_number( num_evars )

  truth_table = exo_in.get_element_variable_truth_table()
  new_truth_table = []
  for iblk in xrange(len(blk_ids)):
    blk_id = blk_ids[iblk]
    if blk_id in threshold_blk_ids:
      for ivar in xrange(num_evars):
         new_truth_table.append(truth_table[iblk*num_evars + ivar])
  exo_out.set_element_variable_truth_table(new_truth_table)

  evar_names = exo_in.get_element_variable_names()
  for ivar in xrange(num_evars):
    exo_out.put_element_variable_name(evar_names[ivar], ivar+1)

  elem_id = 1
  for iblk in xrange(len(blk_ids)):
    blk_id = blk_ids[iblk]

    (elem_type, num_blk_elems, \
```

```
     num_elem_nodes, num_elem_attrs) = exo_in.elem_blk_info(blk_id)

    blk_elem_ids = set()
    for ielem in xrange(num_blk_elems):
      blk_elem_ids.add(elem_id)
      elem_id += 1

    if blk_id not in threshold_blk_ids:
      continue

    threshold_blk_elem_ids = threshold_elem_ids.intersection(blk_elem_ids)

    blk_elem_ids = list(blk_elem_ids)
    blk_elem_ids.sort()

    threshold_blk_elem_ids = list(threshold_blk_elem_ids)
    threshold_blk_elem_ids.sort()

    threshold_num_blk_elems = len(threshold_blk_elem_ids)

    do_copy = []
    jelem = 0
    for ielem in xrange(num_blk_elems):
      if jelem < threshold_num_blk_elems and \
         blk_elem_ids[ielem] == threshold_blk_elem_ids[jelem]:
        do_copy.append(True)
        jelem += 1
      else:
        do_copy.append(False)

    for ivar in xrange(num_evars):

      if truth_table[iblk*num_evars + ivar] == False:
        continue

      evar_name = evar_names[ivar]

      for istep in xrange(num_copy_steps):
        vals     =     exo_in.get_element_variable_values(     blk_id,     evar_name,
copy_steps[istep] )

        new_vals = []
        for ielem in xrange(num_blk_elems):
          if do_copy[ielem]:
            new_vals.append(vals[ielem])

        exo_out.put_element_variable_values( blk_id, evar_name, istep+1, new_vals )

#
# close out the exodus databases
#
exo_in.close()
exo_out.close()
```

A.4. Python script used to convert facet-based geometry to ACIS geometry

```
#!python

#This script uses an ASCII-based facet file to create ACIS geometry 'from the ground
up' in Cubit.
#This script expects the facet file to define a single enclosed volume.
#This script does *not* perform well as the size of the facet file increases.
#Larger files may take significant time and cause Cubit to crash when memory fills.
#The format of facet or stl files varies a bit, you can use this as an example.
#As with all examples, your mileage may vary.
#Originally written by Kevin Pendley March, 2009 based on earlier scripts dating from
June, 2006.
#Newer version uploaded by cubit-help - Feb 2017

#This script expects the facet file to be in the following format:
# <Vertex_count>
# <Vertex_ID> <tab> <X_coord> <tab> <Y_coord> <tab> <Z_coord>
# ...
# <Facet_count>
#   <Facet_ID>   <tab>   <Vertex_ID_1>   <tab>   <Vertex_ID_2>   <tab>   <Vertex_ID_3>
(Optional:<tab> <Vertex_ID_4>)
# ...

cluster_ids = ["chama", "uno", "serrano", "ser", "ghost", "gho",
                   "eclipse", "ec", "skybridge", "sb", "cayenne",
                   "jemez", "pecos"]
import os
import socket
import sys
host_name = socket.gethostname()

if [cluster_id for cluster_id in cluster_ids if cluster_id in host_name]:
    location = "cluster"

elif "cee" in host_name:
    location = "cee"

else:
    # Assume sierra remesh is running on the amech LAN
    location = "amech"

if location == "cluster" or location == "cee":
    path = "/projects/cubit/claro.Lin64.beta/bin"
elif location == "amech":
    path = "/apps/Cubit/claro.Lin64.beta/bin"
if path:
    try:
        sys.path.insert(1, path)
        os.environ["PYTHONPATH"] = path + os.pathsep + os.environ["PYTHONPATH"]
    except KeyError:
        os.environ["PYTHONPATH"] = path
print os.environ["PYTHONPATH"]
```

```
import cubit
cubit.init(['cubit','-nojournal'])

#Update below with the correct path/filename
infacetfile = "v1.fac"
infile = open(infacetfile,"r")
line = infile.readline()
words = line.split()
vertex_count = int(words[0])
print "Number Vertex = ", vertex_count

d_ids = {}
for n in range(vertex_count):
  line = infile.readline()
  words = line.split()
  v_id = int(words[0])
  x_val = float(words[1])
  y_val = float(words[2])
  z_val = float(words[3])
  command = "create vertex " + str(x_val) + " " + str(y_val) + " " + str(z_val)
  print command
  cubit.cmd(command)
  print 'a'
  d_ids[v_id] = cubit.get_last_id("vertex")
  print 'b'

line = infile.readline()
face_count = int(line)
print "Number Faces = ", face_count
for n in range(face_count):
  line = infile.readline()
  words = line.split()
  nw = len(words)
  if nw == 4:
    facetID = int(words[0])
    coord1 = int(words[1])
    coord2 = int(words[2])
    coord3 = int(words[3])
    command = "create surface vertex " + str(d_ids[coord1]) + " " + str(d_ids[coord2])
+ " " + str(d_ids[coord3])
    cubit.cmd(command)
  elif nw == 5:
    facetID = int(words[0])
    coord1 = int(words[1])
    coord2 = int(words[2])
    coord3 = int(words[3])
    coord4 = int(words[4])
    command = "create surface vertex " + str(d_ids[coord1]) + " " + str(d_ids[coord2])
+ " " + str(d_ids[coord3]) + " " + str(d_ids[coord4])
    cubit.cmd(command)
  else:
    print "Bad facet read."

#cubit.cmd('create volume surface all heal ')
#cubit.cmd('delete vertex all ')
```

```
cubit.cmd('export acis "surface1.sat" overwrite')
exit
```

A.5. Sample Journal file for Alignment of deformed can geometries

```
# Start Fresh
reset
reset aprepro


########### Define variables Begin ###########

#{dim_scale = .0254} #inch into meter
#{scale_exp2sim_dim = 0.001} #scale exp. to consistent unit with sim scale
########### Define variables End ###########

########### Import ACIS Deformed Shape (.stp) Files Begin  ###########

# Simulated Result File
import step "/home/xlao/Downloads/cansim_v2.stp" heal

# Experimental Result File
import step "/home/xlao/Downloads/Unit 6-2.stp" heal


# Reference undeformed base-line geometry for simulation
#import step "/home/xlao/ductile_failure/PeEM/can_crush_model/Can_One Piece.STEP" heal

########### Import ACIS Deformed Shape (.stp) Files End  ###########

########### Scale Exp. Result to Consistent Dim. of Sim. Result Begin  ###########

vol 8 scale {scale_exp2sim_dim}

########### Scale Exp. Result to Consistent Dim. of Sim. Result End  ###########


########### Alignment Process Begin  ###########

# Get correct consistent orientation in space
# Note: Exp. Result oriented such that bottom flat surface of the can is on the global
XY plane; +X is along the length (long side) of the can
# +Y is point from the 'straight' edge length towards the 'curve' edge length; global
+Z is towards the top of the can

# Reorient the Sims. Results to orientation of the Exp. Results
rotate Volume 1 to 7 angle 30  about X include_merged preview
rotate Volume 1 to 7 angle 30  about X include_merged
rotate Volume 1 to 7 angle 45  about Y include_merged preview
rotate Volume 1 to 7 angle 45  about Y include_merged
```

```
# since the exp. results consisted of 1 volume, it is best to align it with the sim
results (which consist of 7 volumes)

# We will start the alignment process in the +XY plane projection
# Create TEMP. geometries to help facilitate the alignment process
    # Create temp. center vertices of the screw tabs
    # create temp. plane to assist in drawing pin hole of exp.
    create surface rectangle width .25 zplane
#    move Surface 3209  x 0 y 0 z -.02 include_merged
    # Since there are curves surface on the screw tabs, create 'circle' of the screw
tab hole by projection of the curves
    # closed perimeter of the screw tab hole shalft

    #As few from Top, Farthest screw tabs (opposite contact point), CCW to other screw
tabs
    project curve 35000 34861 34863 34864 35003 35145  onto surface 25610
    project curve 36065 35681 36079 36282 36064 36061  onto surface 25610
    project curve 38075 38270 38463 38284 38092 37897  onto surface 25610
    project curve 39808 39807 39913 39665 39524 39812  onto surface 25610
    project curve 36643 36429 36435 36204 35986 35797 35989 36208  onto surface 25610
    project curve 37584 37585 37780 37774 37572 37365  onto surface 25610

    #Create corresponding circle sheet and its center vertex at the screw tabs (same
ref. as above)
    create surface circle vertex 14399  14392  14395
    create vertex center curve 40043

    create surface circle vertex 14402 14410 14403
    create vertex center curve 40044

    create surface circle vertex 14418 14412 14422
    create vertex center curve 40045

    create surface circle vertex 14426 14435 14434
    create vertex center curve 40046

    create surface circle vertex 14453 14458 14463
    create vertex center curve 40047

    create surface circle vertex 14441 14444 14438
    create vertex center curve 40048

    #Delete no longer needed temp geometric entities
    delete curve 40005 to 40042
    delete body 9 to 15



  #Create corresponding circle sheet and its center vertex at the screw tabs (same
ref. as above)
  # Sim Results
   create surface circle vertex 2226  2140  2137
   create surface circle vertex 842  1030  998
   create surface circle vertex 679  607  703
   create surface circle vertex 165  341  249
```

```
   create surface circle vertex 1828  1713  1770
   create surface circle vertex 1305  1414  1303

   # Create ref. center vertex of screw tabs (sims results)
    create vertex center curve 40049
    create vertex center curve 40050
    create vertex center curve 40051
    create vertex center curve 40052
    create vertex center curve 40053
    create vertex center curve 40054

   # Delete not needed temp. geo. entities
   delete body 16 to 21

   compress all

    #Rename nodes to diff
    # Exp. can geo
    vertex 14384 rename "EC_N1"
    vertex 14385 rename "EC_N2"
    vertex 14386 rename "EC_N3"
    vertex 14387 rename "EC_N4"
    vertex 14388 rename "EC_N5"
    vertex 14389 rename "EC_N6"

    # Sim. can geo
    vertex 14390 rename "SC_N1"
    vertex 14391 rename "SC_N2"
    vertex 14392 rename "SC_N3"
    vertex 14393 rename "SC_N4"
    vertex 14394 rename "SC_N5"
    vertex 14395 rename "SC_N6"


# Move exp results to align with sim results
move Volume 8  x 0.0846175858524 y 0.00980401743392  z 0.06959668920402  include_merged
move  Vertex EC_N1 EC_N2 EC_N3 EC_N4 EC_N5 EC_N6 x 0.0846175858524 y 0.00980401743392
z 0.06959668920402  include_merged

#More fine tune alignment via eye-ball test
# Z-X plane
rotate Volume 8 angle .25  about vertex 14386  14395  include_merged

#Y-Z plane
rotate Volume 8 angle .25  about X include_merged

#After 'good' enough alignment delete not needed temp geo. entities
delete vertex EC_N1 EC_N2 EC_N3 EC_N4 EC_N5 EC_N6 SC_N1 SC_N2 SC_N3 SC_N4 SC_N5 SC_N6
############ Alignment Process End  ############
```

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Bonnie Antoun	8363	brantou@sandia.gov
Brett Collins	8363	bccolli@sandia.gov
Edmundo Corona	1558	ecorona@sandia.gov
Jay Dike	8752	jjdike@sandia.gov
Amanda Jones	1528	ajones1@sandia.gov
Kyle Karlson	8752	knkarls@sandia.gov
Brian Lester	1558	btleste@sandia.gov
Kimberley Mac Donald	8363	kamacdo@sandia.gov
Stacy Nelson	1558	smnelso@sandia.gov
Yuki Ohashi	8752	yohash@sandia.gov
Vincente Pericoli	8755	vperico@sandia.gov
CA Technical Library	8551	cateclib@sandia.gov

This page left blank

This page left blank

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.