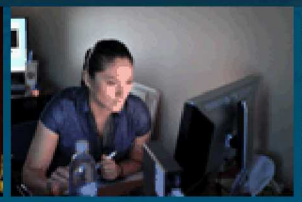
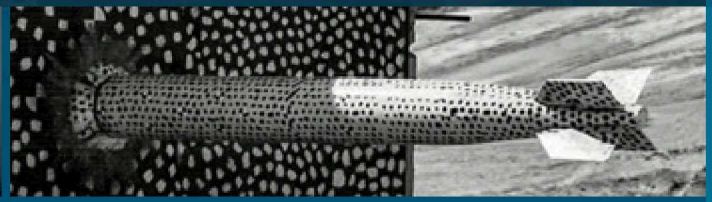
SAND2019-0195PE

DIC Simulator using Blender and Python





PRESENTED BY

Dan Rohe

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.

Motivation

Real life is hard

- The DIC simulator can give a noise-free, infinite depth of field, completely customizable environment to test DIC, Photogrammetry, and other optical techniques without getting bogged down in experimental details.

Test planning

- It can be time consuming to set up a test; once the part model and deflections have been determined and loaded into the simulator, the test setup can be quickly investigated. Feasibility of optical techniques can be investigated prior to the acquisition of hardware and lab space.

Drawbacks:

- In real life, taking 10,000 photos can take a fraction of a second with high-speed cameras, rendering each one takes a bit more time.
- The renderer, Blender, which is designed for 3D modeling and animation, has rudimentary antialiasing capabilities and can capture some subpixel motions; however, better results are obtained if the images are rendered at a higher resolution and then downsampled with interpolation. This drastically increases render times (factor of 10x or more)

Framework

The current framework utilizes the Blender modeling tool to create images from a 3D model.

Python preprocessors are available to convert e.g. an exodus file to a set of modes and modal qs.

Blender's Python engine is used to automate the deformation of the mesh and the rendering of the images at some multiple of the desired image resolution.

A subprocess is called to downsample the images to the desired image resolution after they have been rendered.

The images can then be loaded into DICe or another software or code to perform analysis.

Blender Website: https://www.blender.org/

DIC Simulator Examples

This presentation will cover two examples:

- To show the basic capabilities, we will examine the simple case of a plate exposed to an impact.
- To show additional capabilities we will examine a test-planning case where we look at real bomb test data.

Plate Example

The workflow for the plate example problem looks like this:

- Create a finite element model and run an eigen solution to compute modes
- Apply a virtual impact to the plate and generate modal responses
- Export the plate mesh to blender
- Generate a speckle pattern on the plate
- Set up scene including camera and lighting parameters
- Load in a calibration target and render a few calibration images
- Run a Python script that deforms the mesh and renders the image.
- Load calibration images and deformation images into DICe to run the analysis.
- Examine the response data and fit modes to it.

Plate Example: Finite Element Model

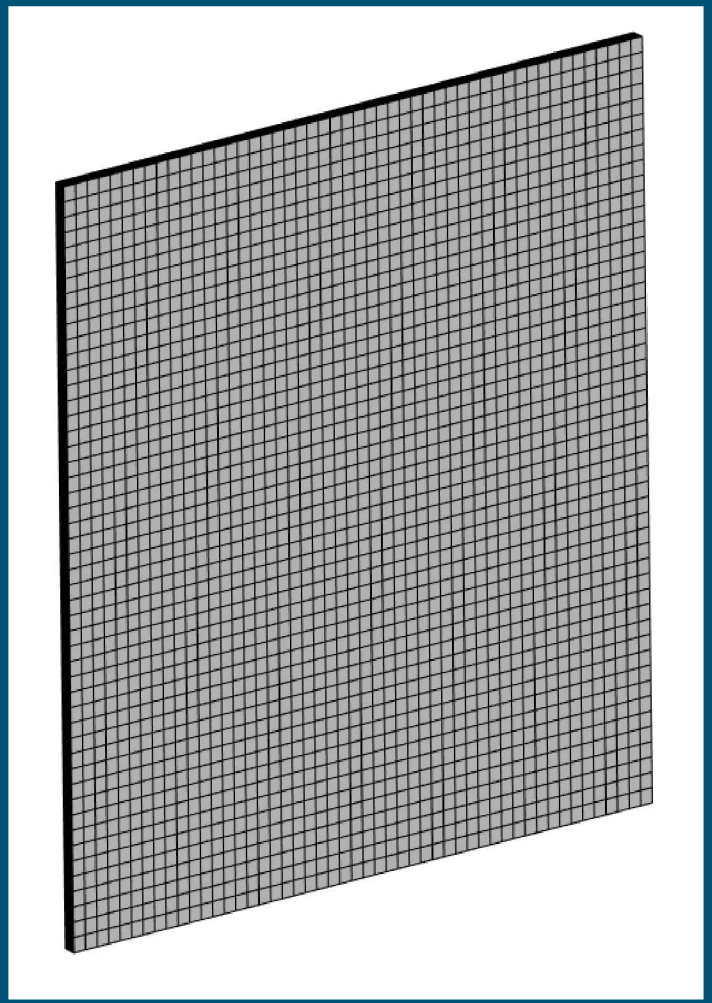
Finite element model parameters:

- Dimensions: 6” x 7” x 0.125”
- Material: Steel
- 18207 nodes, 15000 hex elements

After solving for the eigenvalues, the model was “skinned”, turning the volume elements into surface elements that could be used to construct a Blender surface mesh.

The skinned model was imported into Python so an impact could be applied.

The connectivity matrices and nodal coordinates were saved to external files that could be read by Blender.



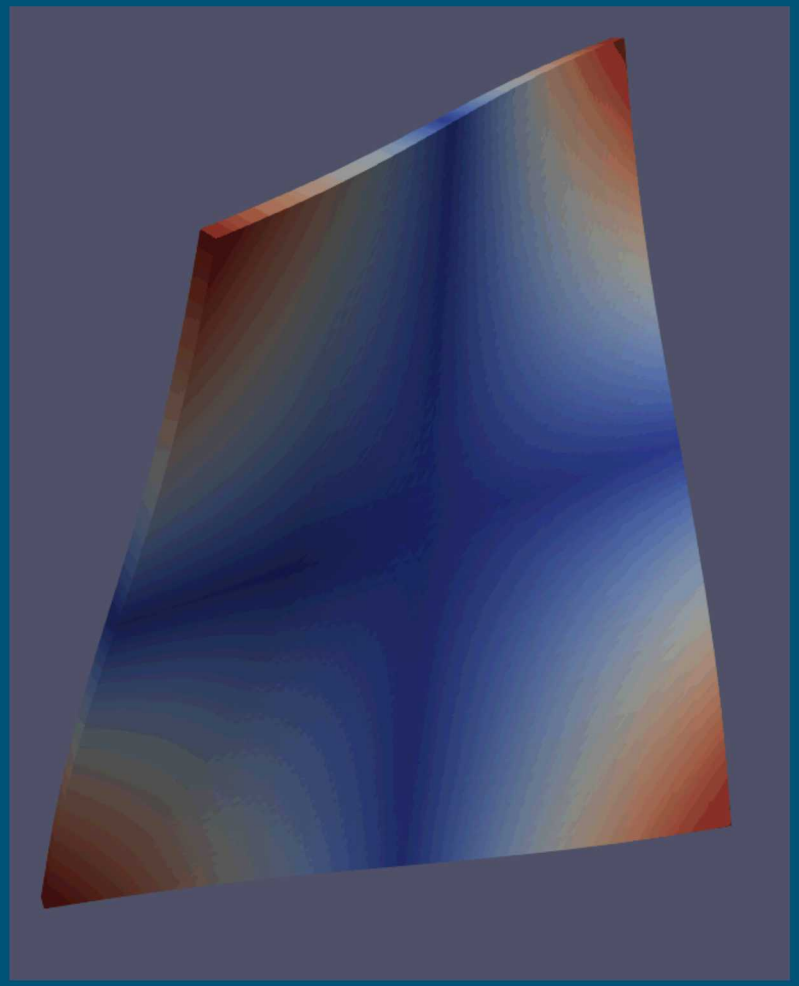


Plate Example: Applying the impact

An impulse signal was generated with a sine-squared pulse.

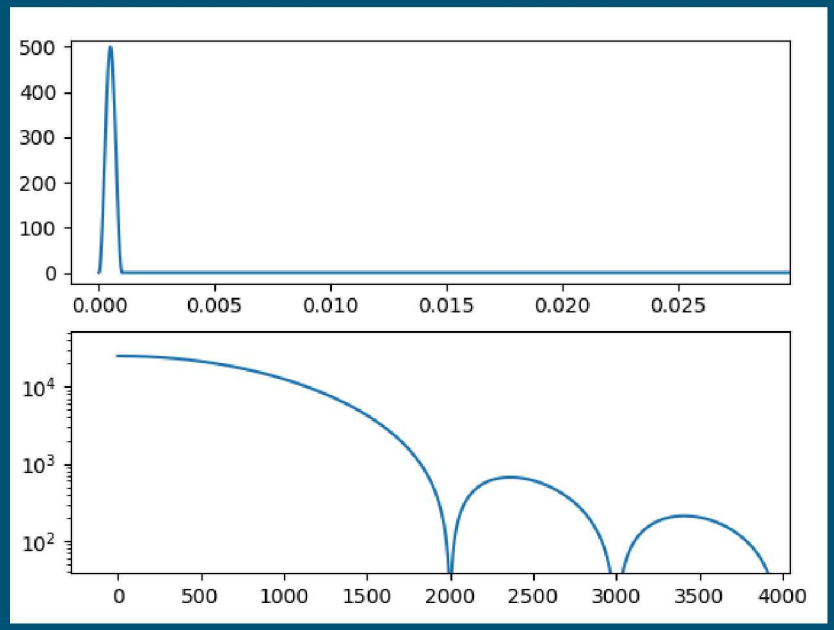
The force was applied perpendicular to the plate at the corner in the modal domain

0.2% modal damping was assumed.

The modal displacements over time were saved so they could be loaded into Blender.

- An alternative would be to save the displacement at every point, but it is more efficient to load in mode shapes and qs and multiply them in the code.

The maximum displacement was 0.06 inches.



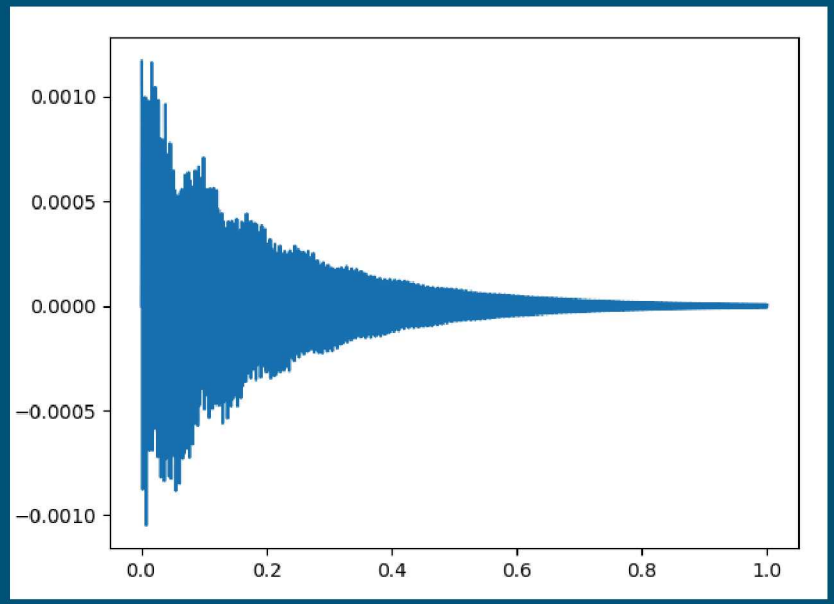


Plate Example: Setting up the Blender Scene: Loading in the Mesh

The mesh is loaded using a Blender Python script, creating the mesh using the connectivity matrix and the vertex coordinates from the finite element model.

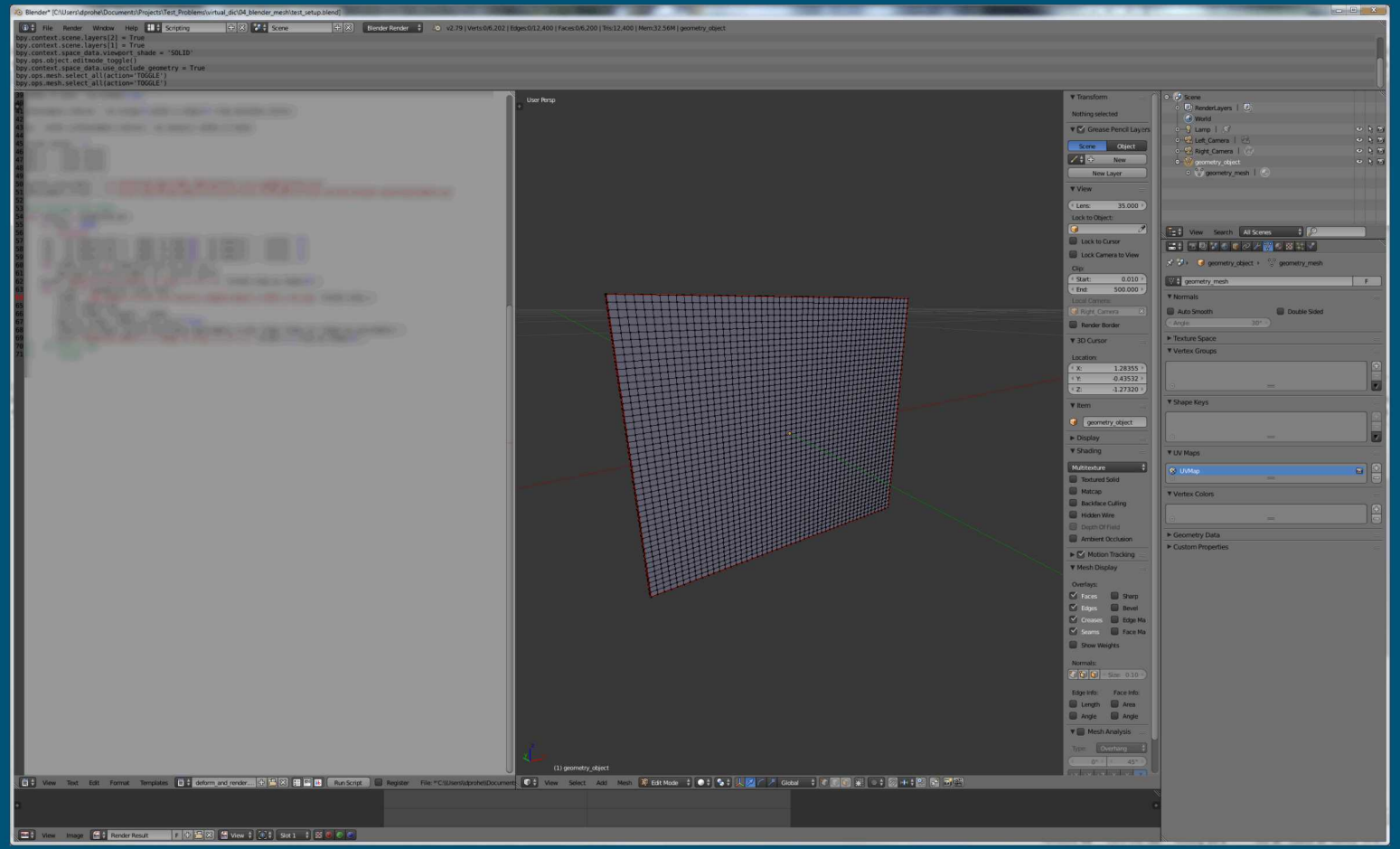
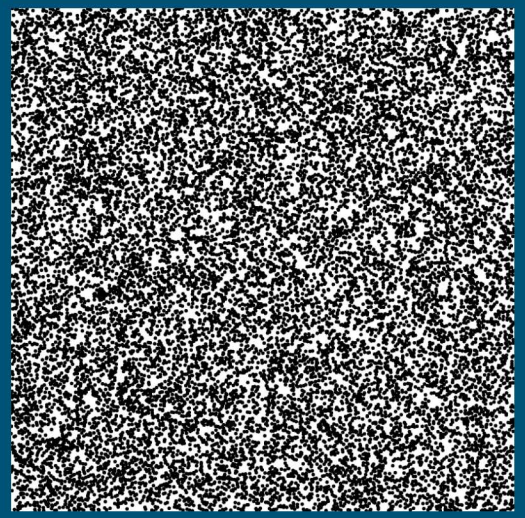


Plate Example: Creating a Speckle Pattern

A speckle pattern image was created using a Python script that randomly draws black dots of a specified size on a white background.

The Blender mesh was unwrapped and the image was applied to the surface as a texture using the UV mapping capabilities in Blender.



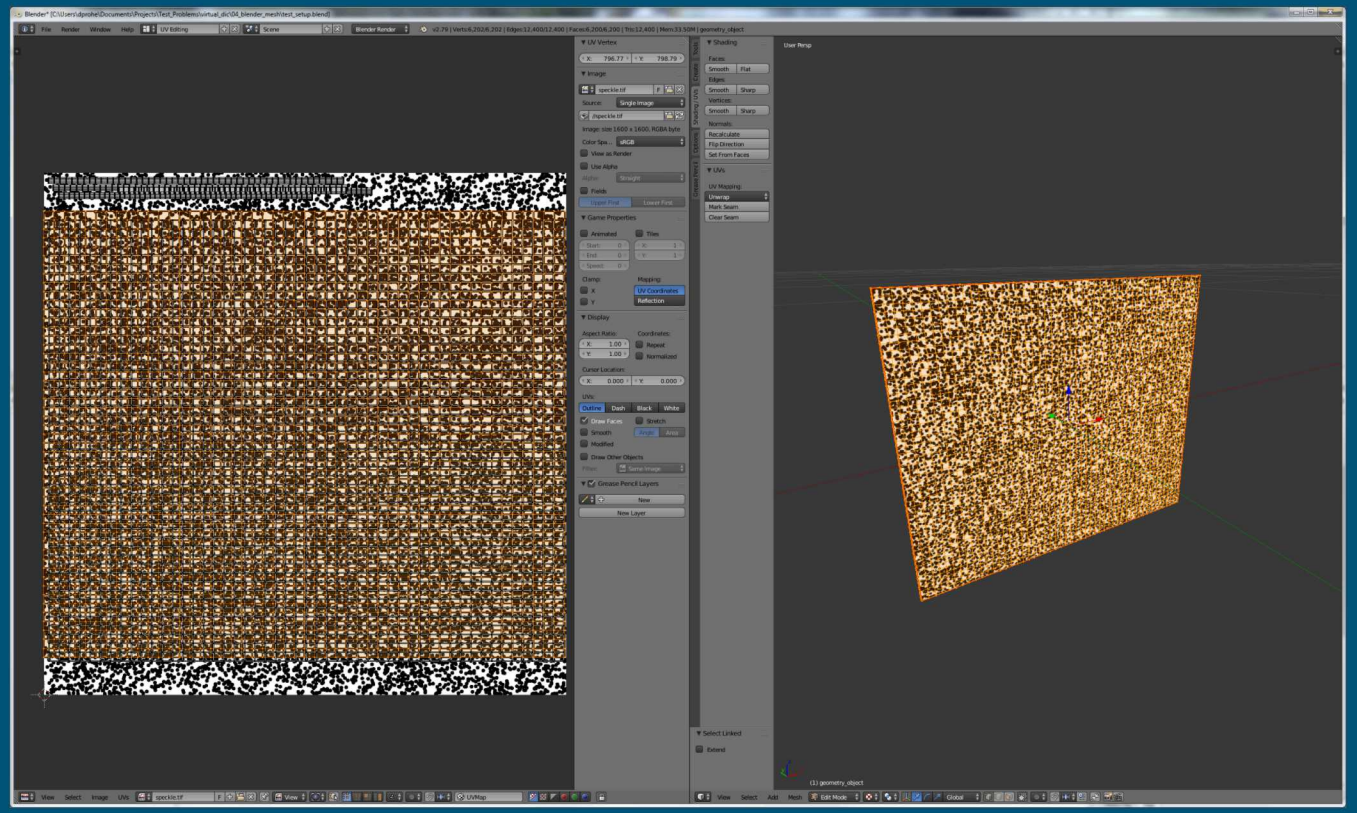
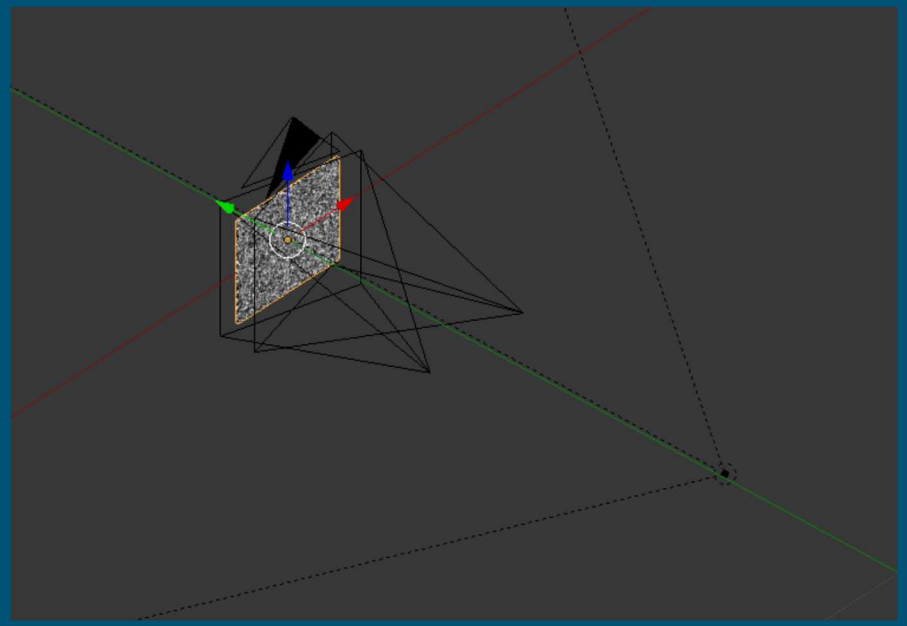


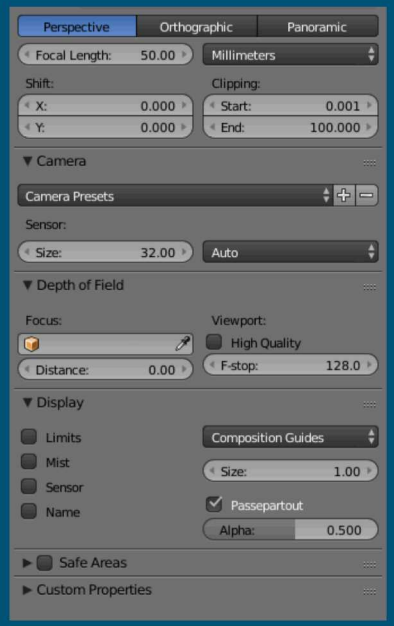
Plate Example: Scene Setup

To render an image, Blender needs a Camera and a Light source. These need to be added to the scene.

In addition to the camera positions and rotations, focal length and depth of field can also be specified.

Several different types of lights are available, and the intensity of each light can be set.





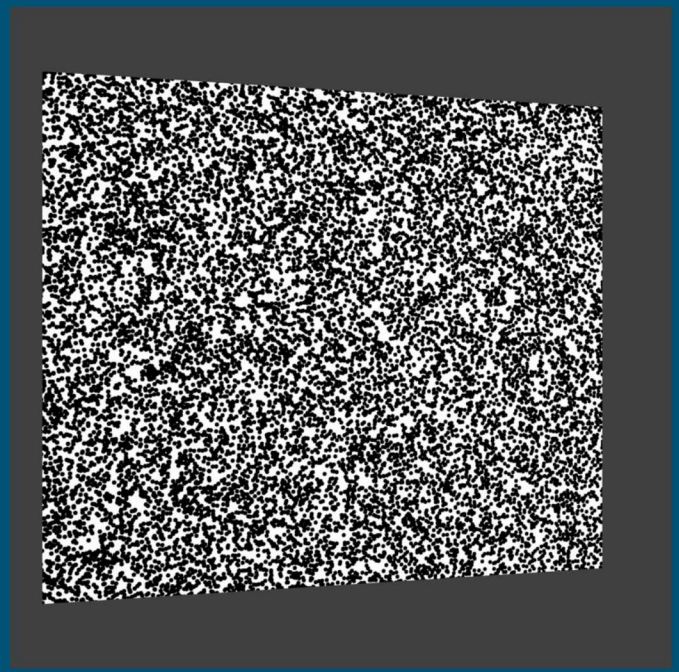


Plate Example: Calibrate Cameras

While we could theoretically create a perfect calibration because we have exact knowledge of camera positions, focal lengths, etc., a more realistic simulator would include camera calibration.

Created an image using Correlated Solutions' printable calibration targets, and applied the image to a mesh in the same way as was done for the plate mesh.

Calibration target can be programmatically loaded and positioned; a 6D space of calibration target positions and orientations can be sampled to generate calibration images.

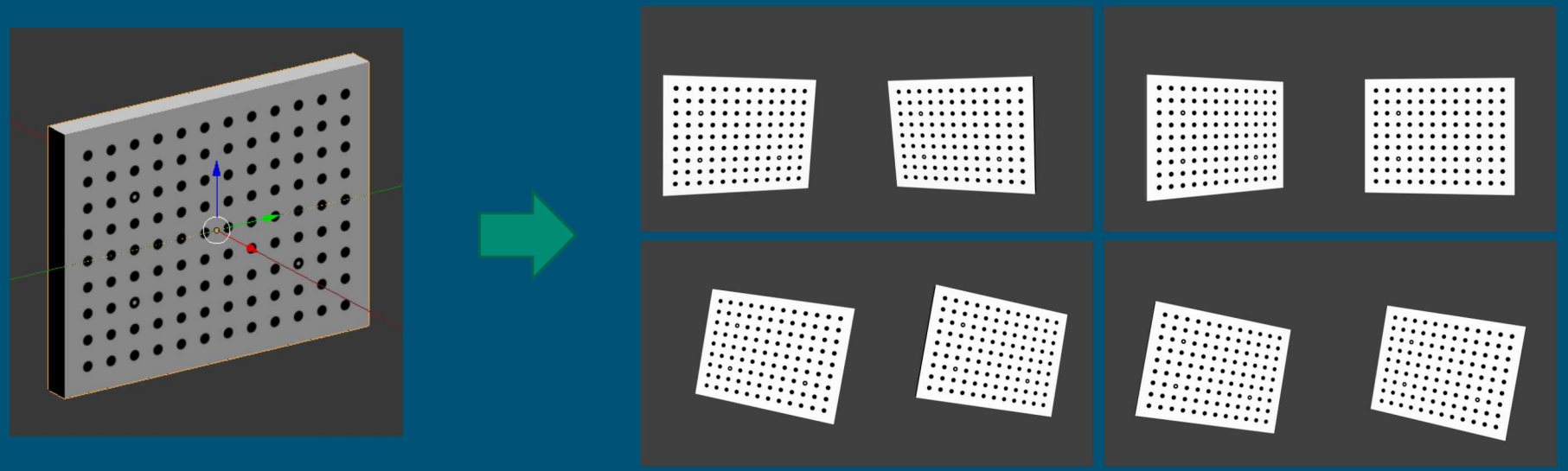


Plate Example: Deform Mesh and Render

A Blender Python script was written that loads in the mesh data, mode shapes, and modal qs at each time step and deforms the mesh appropriately.

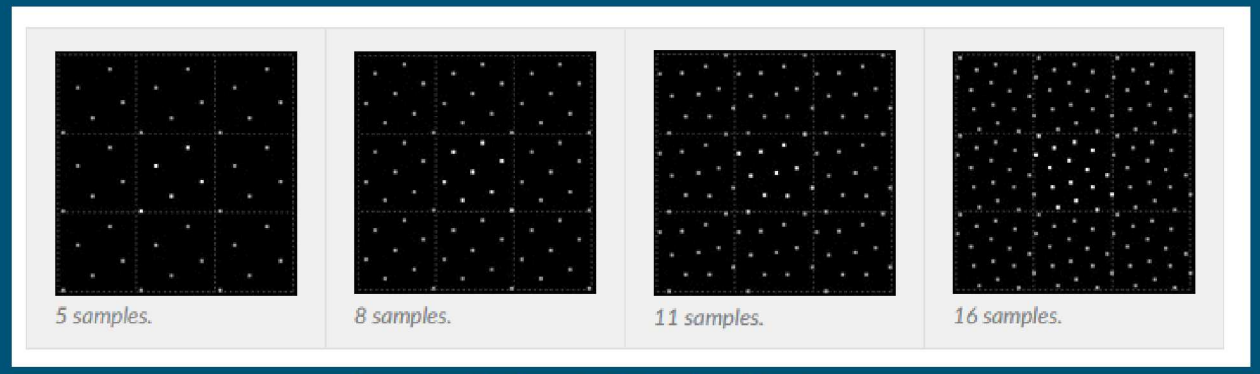
The Blender Render has a maximum of 16x oversampling for antialiasing. While it produces visually acceptable images, it was found to be insufficient to accurately render subpixel motions.

Instead the images are rendered at some factor of oversampling (6-10x) in addition to the native antialiasing.

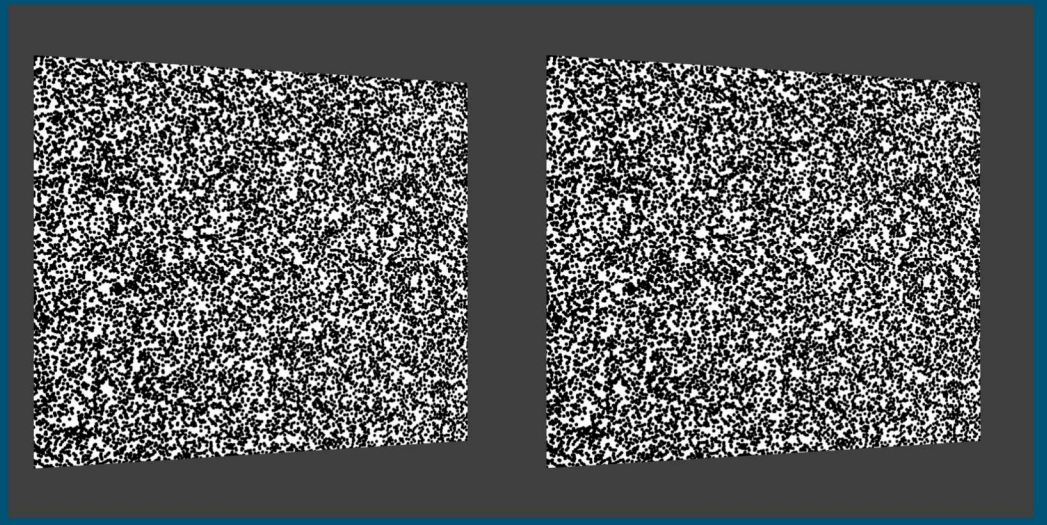
- Note that oversampling the image 10x creates a 100x larger image, and can have a significant effect on render times for each image!

A downsample script using the Python Image Library reduces the image size to the desired resolution using a Bicubic interpolant.

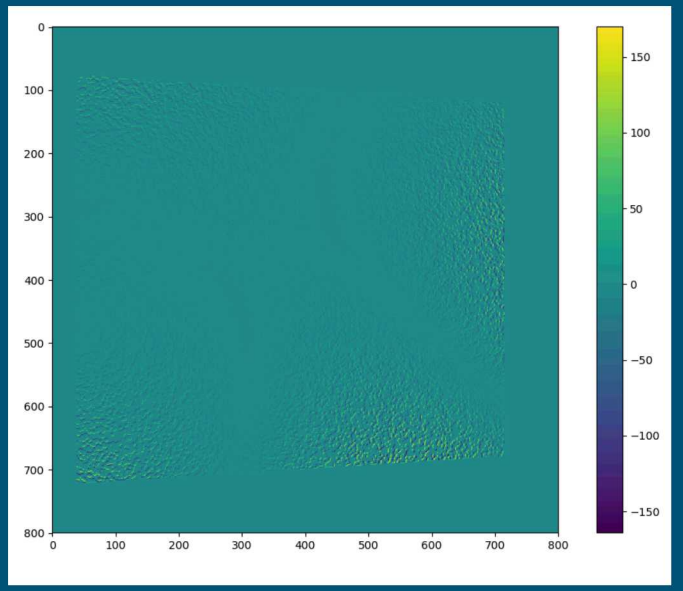
Images seem identical to the naked eye, but a subtraction of the two images shows subpixel motions.



Blender Native Antialiasing Strategy



Reference Image and Image 100



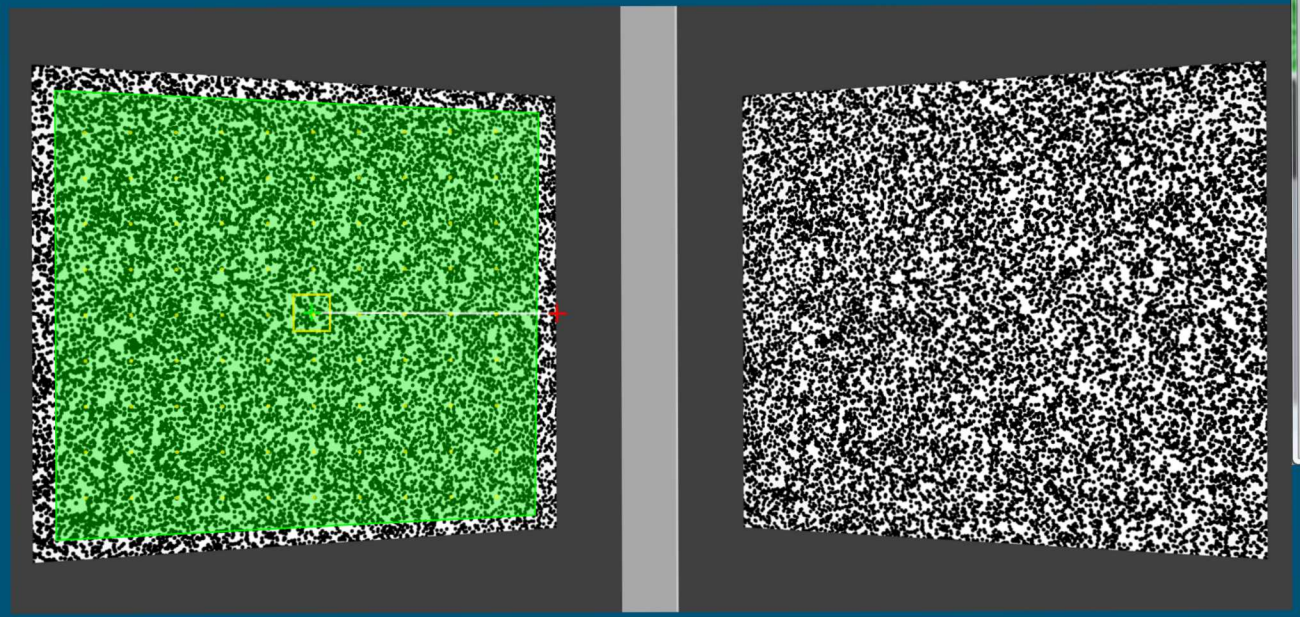
Subtraction of Image and Image 100
(values out of 255)

Plate Example: DICe Analysis

Calibration images were loaded into DIC

- RMS Error: 0.127045
- Average Epipolar Error: 0.202899

Analysis was performed on the images.



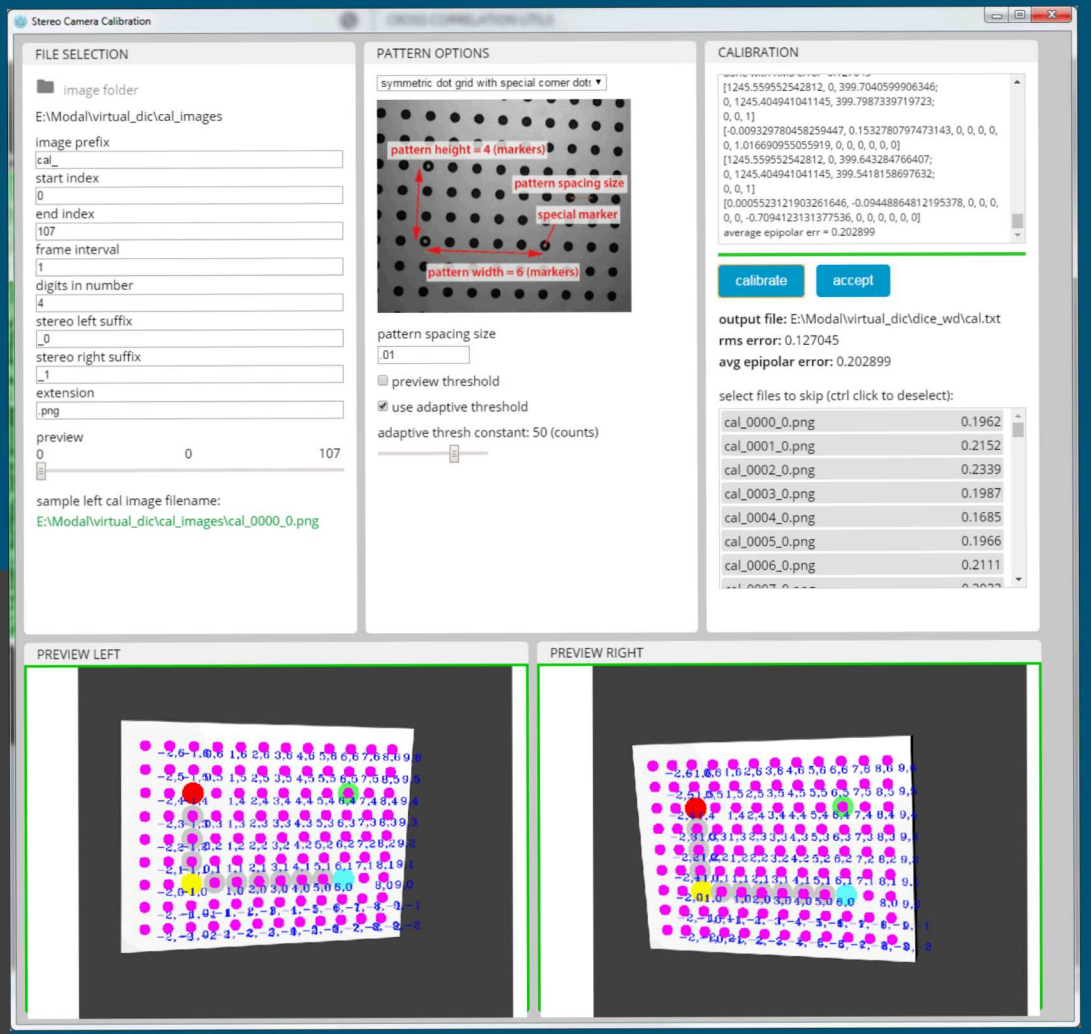


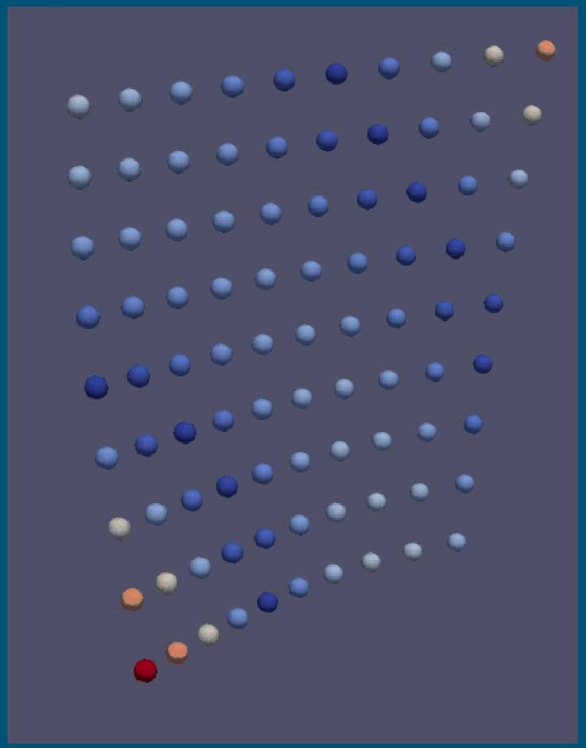
Plate Example: DIC results

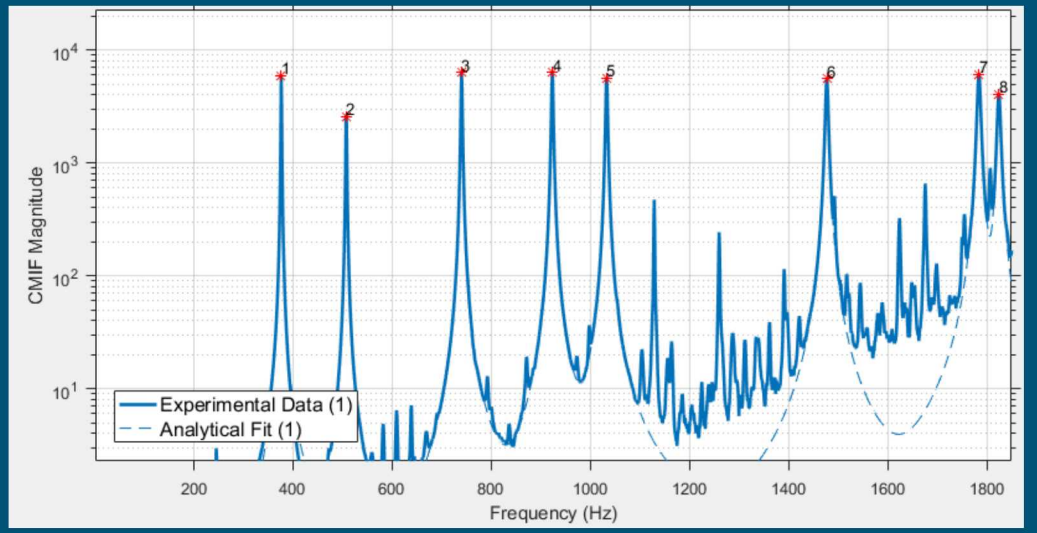
Data looked reasonable

Was able to fit modes reasonably

Some problems with the data that are outside of the DIC simulator (at this point I was just trying to get it to work, not worrying about data quality):

- Leakage – Displacements did not ring down by the end of the measurement frame
- Aliasing – Integrated using sampling frequency 10x the bandwidth of interest, downsampled time signals without any kind of antialiasing filter.
- Noise-free input FFT – Hammer impact was perfectly measured, so dips in frequency domain go down to zero. Image data has an effective noise floor, so FRF goes to infinity at these locations.



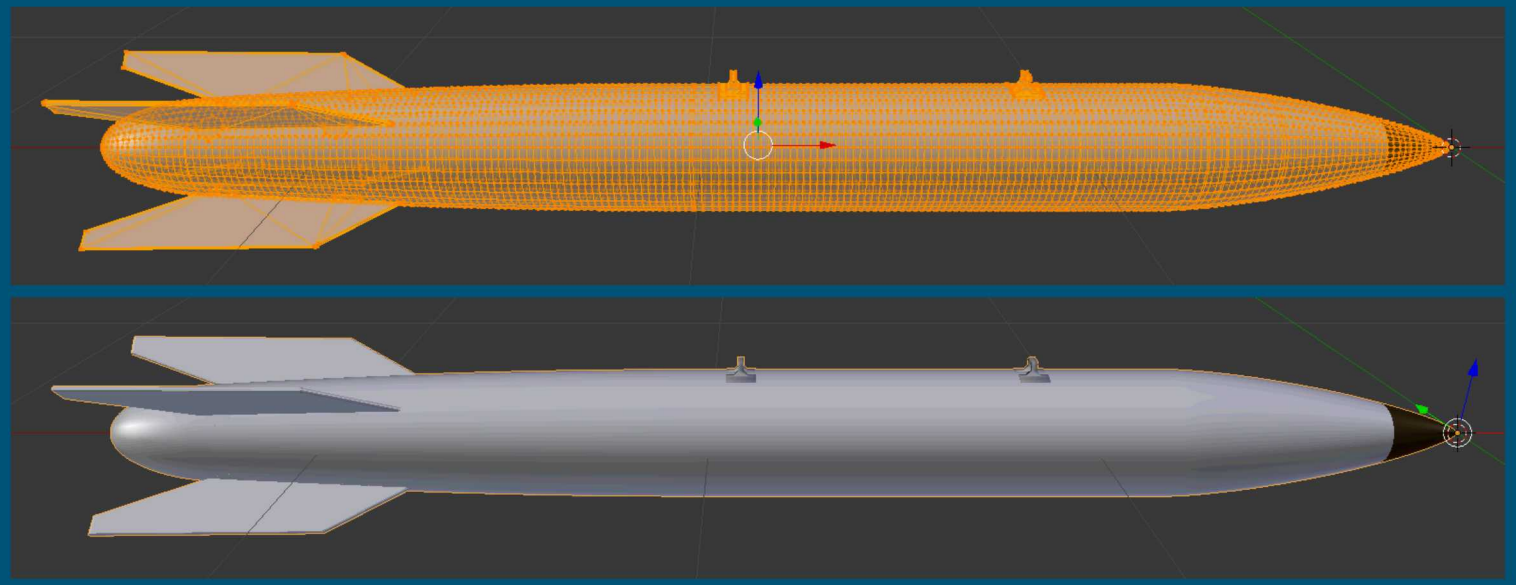


Bomb Example: Test Planning

The second example will illustrate the test-planning capabilities of the DIC simulator framework, using a bomb mesh and real test data.

A bomb mesh had been created for creating illustrations in test reports

- No physics behind the mesh (mode shapes, mass or stiffness matrices, etc.)
- Simply a set of vertices and their connectivity.



Bomb Example: Mapping the Mesh to Test Data

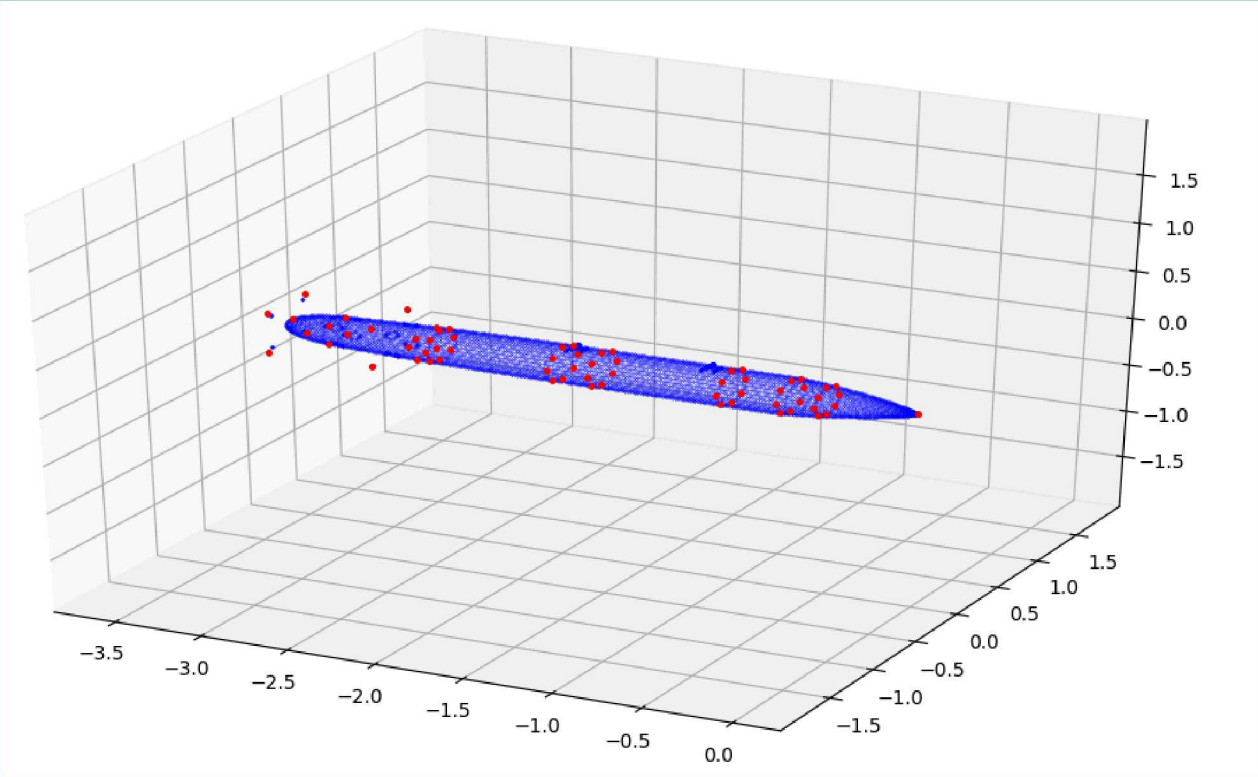
We have test data at a number of locations on the Bomb, so we need to expand those to the Bomb mesh.

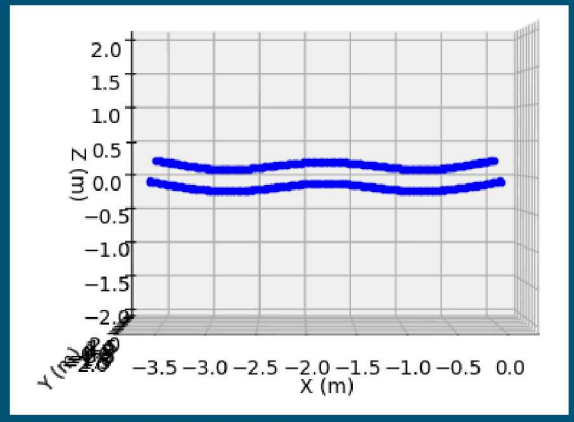
One popular technique is SEREP, which can expand data from one set of degrees of freedom to another set if a set of basis shapes is available.

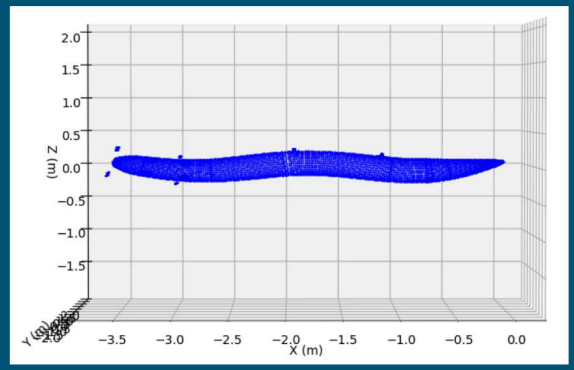
Create a set of basis shapes using beam bending shapes of the bomb case plus fin motions

Can use geometry to project beam translations and rotations to the skin of the bomb.

SEREP techniques were used to solve for the "modal qs" that correspond to the expanded beam shapes.





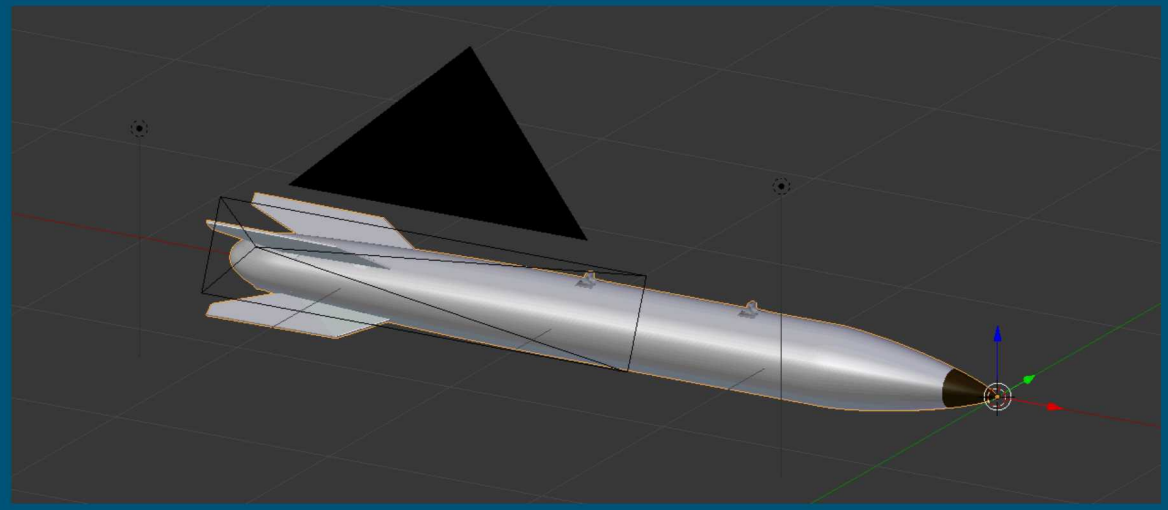


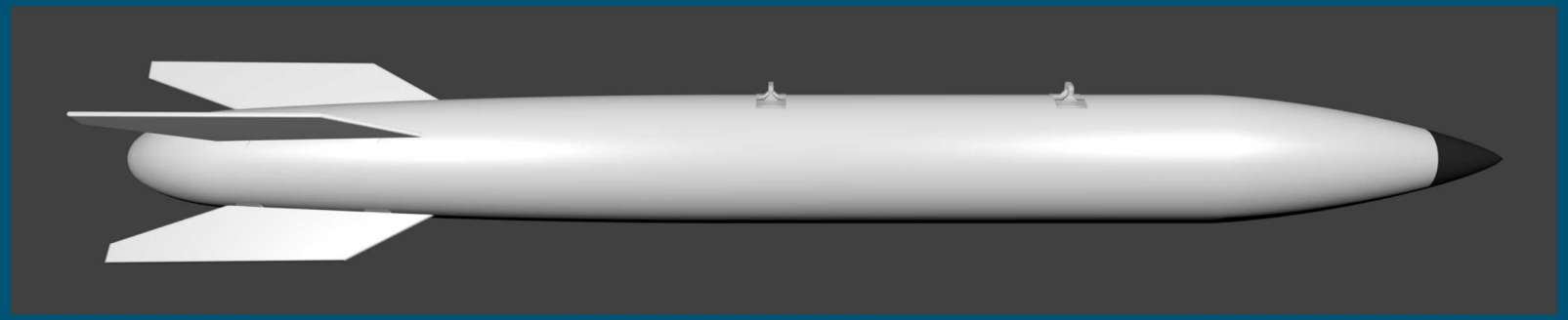
Bomb Example: Scene Setup, Deformation, and Render

Two lights and a “wide angle” camera was used, emulating the constraints of the laboratory in which the system was tested.

Only using one camera to look at motion magnification feasibility

Model was deformed and rendered at each step in the test data.

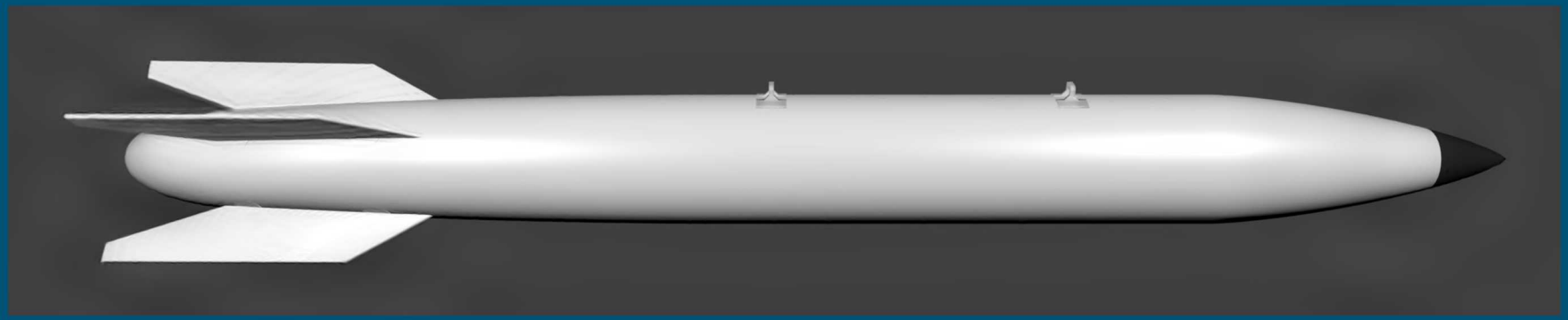




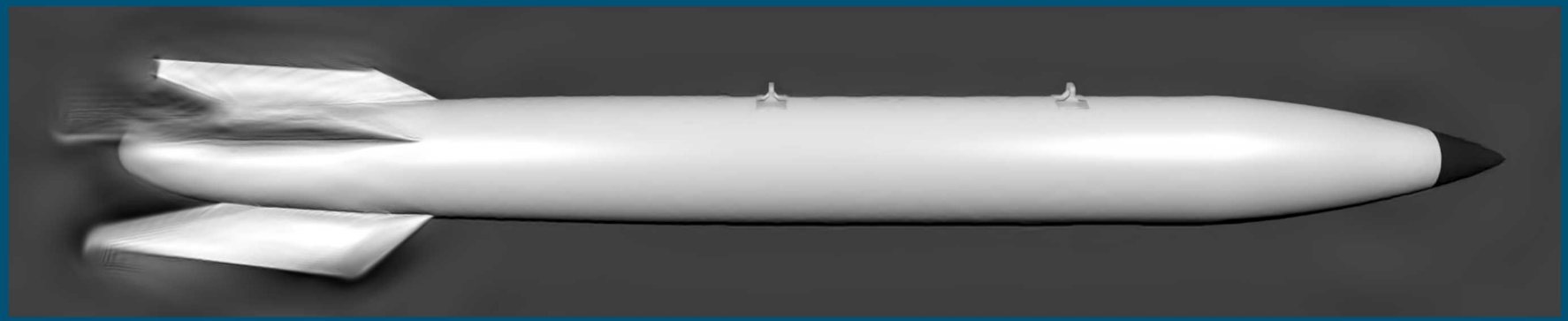
Bomb Example: Motion Magnification

Can utilize the DIC Simulator to determine if deflections are large enough to see what you want to see:

- In this example fins move significantly, but we are unable to magnify motions on the case.



- Or we can increase displacements in the simulator to see how much motion would be required before the desired analysis technique can be performed.



What can still be investigated:

How much oversampling of the image is needed to resolve a specified displacement?

- It would be easy to do a noise floor study where we move the part along an axis increasingly small amounts and see what the computed displacement is. Since we know the actual displacement we imparted, we can easily compare the true and computed displacements to understand where the simulator breaks down.

Implement noise sources to simulate their effect:

- Blender can simulate depth of field – this is a constraint on real systems
- Blender can do refraction, so we could theoretically simulate heat waves – requires a more sophisticated renderer that may take a lot longer to draw a scene
- Blender assumes a perfect pinhole camera, but we can implement lens distortions on the images by using the OpenCV library on the output images.
- Can play with inferior speckle patterns (not black and white, but some grey), nonuniform lighting, specular reflections, etc.
- Can add noise to the images

Implement more challenging problems:

- Could render a finite element simulation where element death occurs:
  - Explosions
  - Impacts
- Investigate more interesting geometry