

Implementing Geometric Surface Imperfections into Sandwich Composite Cylinder Finite Element Method Models

Cyrus J. R. Kosztowny¹

NASA Langley Research Center, Hampton, VA 23681

The buckling responses of certain cylindrical shell structures are extremely sensitive to geometric surface imperfections. The NASA Engineering and Safety Center (NESC) Shell Buckling Knockdown Factor Project (SBKF) is conducting research to develop analysis-based buckling design recommendations. Experiments are used to verify the analysis-based factors, but the sensitivity of the test articles to geometric imperfections requires implementing as-manufactured imperfections into high-fidelity finite element method models. Data collection methods such as structured light scanning are used for all geometric surface data used in this work. Common preprocessing and visualization steps used in SBKF are discussed, and steps on how surface scans are prepared for implementation into a finite element model is described. The Python Tool for Implementing Geometric Imperfections in Reduced Structures (Py_TIGIRS), written specifically for the use with SBKF, is briefly described and uses eight functions to extract, modify, and write geometric imperfections into Abaqus input files. Results of the pre-processing methods and results from Py_TIGIRS are provided and compared for Composite Test Article (CTA) 8.2B. Excellent agreement between the visualized scan data and the FEM-extracted geometry is demonstrated. A brief example of why geometric surface imperfections are significant in nonlinear numerical analyses for thin cylinders in axial compression is provided as motivation to use tools such as Py_TIGIRS. Future developments of Py_TIGIRS including expansion to structures of arbitrary geometry is planned.

I. Nomenclature

CTA8.X	= Composite test article, where 8 denotes the cylinder diameter (in feet) and X is the test-article designation
FEA	= Finite element analysis
FEM	= Finite element method
IML	= Inner mold line
OML	= Outer mold line
QA	= Quality assurance
R	= Cylindrical radial coordinate
Z	= Cylindrical axial coordinate
α	= Known angle between emitter and sensor for certain geometry data collection methods
θ	= Cylindrical circumferential coordinate

II. Introduction

The buckling response of thin cylindrical shell structures is known to be sensitive to geometric surface imperfections [1]. Such surface imperfections may result from manufacturing processes, post-manufacturing handling, or damage resulting from impact. One goal of the NASA Engineering and Safety Center (NESC) Shell Buckling Knockdown Factor Project (SBKF) is to develop and experimentally verify analysis-based buckling design guidelines

¹ Research Aerospace Engineer, Structural Mechanics and Concepts Branch, 8 West Taylor St. Mail Stop 190.

for certain types of thin right circular cylindrical shells made of metallic or composite materials [2]. Sandwich composite cylindrical shell structures designed during the SBKF are the test articles considered in this work [3]. There are many ways to implement geometric imperfections in analytical methods such as cosine shape functions [4] or axisymmetric imperfections [5] or in finite element analysis (FEA) numerical codes [6]. The research described in this paper focuses on the methodology used by SBKF to implement geometric surface imperfections into a finite element method (FEM) model and how the structured light scan data used to collect geometric surface imperfections is processed.

The design of the test article is covered extensively in Ref. [3], but all share similar characteristics. Each test article was designed with a lightweight honeycomb core sandwiched by carbon fiber facesheets of varying ply count and orientation. The total length of each test article was 100 inches and was manufactured on a 94-inch constant-diameter tool. To facilitate experimental testing and introduce load to a cylinder without damaging the ends, the ends of a cylinder were set inside channels machined in metallic pi-shaped end rings and potted using an epoxy grout material. A FEM model of each test article is created within the general-purpose FEM software Abaqus[#] [7]. An example of Composite Test Article (CTA) 8.2B FEM model is provided in Figure 1. The green end sections are the metallic end rings into which the ends of the test article are potted. The blue region is the composite sandwich structure. The FEM model is meshed with approximately 150,000 reduced integration, 4-noded (S4R) shell elements.

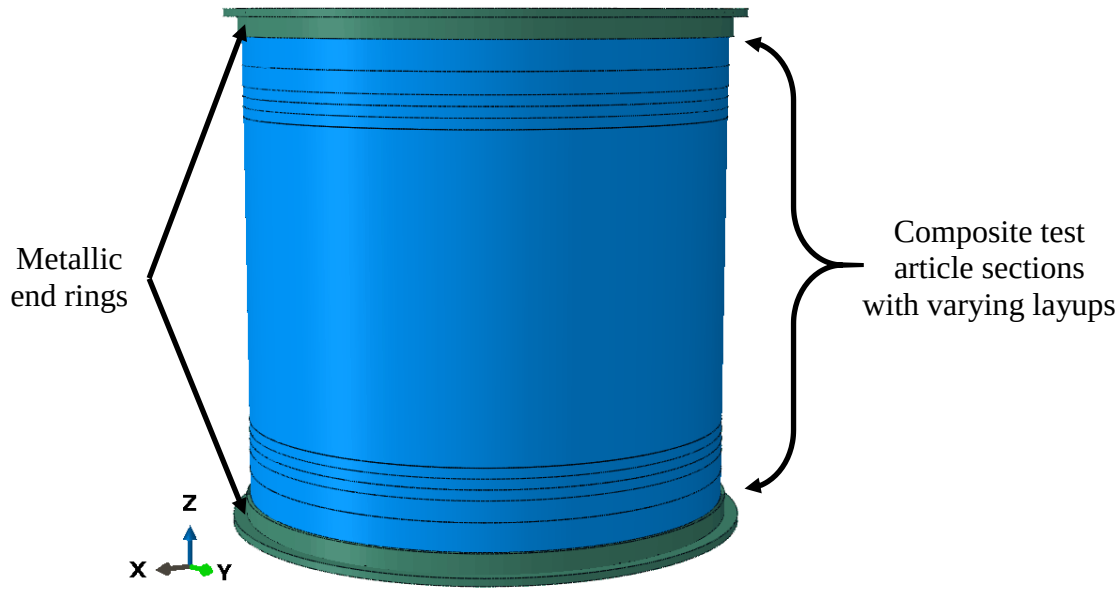


Figure 1. Example of CTA8.2B FEM model with metallic end rings (green) and composite sandwich (blue) sections.

Surface geometry data collection techniques are continually evolving and advancing capabilities. Initial efforts to characterize geometric surface imperfections relied on processing the collected data into a form amenable for analytical methods [8]. Such data was reduced using methods of decomposition such as Fourier transforms or analogous tools. Current practices often utilize directly measured surface geometric imperfections for inclusion in FEA and numerical analyses. A few surface geometry data collection methods including the structured light scanning method used to collect all data used in this work are discussed in Section III. Common practices used in SBKF to preprocess and visualize geometry data along with how it is prepared for implementing into FEM models are discussed in Section IV. The Python Tool for Implementing Geometric Imperfections in Reduced Structures or Py_TIGIRS, written specifically for the use with SBKF, is discussed in Section V with a brief description of the scripting workflow. Reduced structures in this context refers to shell formulations where the thickness of a structure is considered small compared to other dimensions. Results of the CTA8.2B are provided in Section VI. Section VII is reserved for the remaining test articles and are to be included in the final paper. Further motivation for why a tool such as Py_TIGIRS

[#] The mentioning of a company or product is not an endorsement by the National Aeronautics and Space Administration (NASA).

is important for imperfection-critical analysis response is provided in Section VIII. Future improvements to Py_TIGIRS are briefly discussed in Section IX, and concluding remarks are provided in Section X.

III. Surface Geometry Data Collection

Various types of surface geometry data collection methods are available. Different methods have positives and negatives depending on what the needs of the user are, and no single method produces consistently better results for every situation simply due to how each user defines what better might be. One of the first methods initially used for the collection of surface geometry data for imperfection measurement in cylindrical-shell buckling was a force-sensitive contact probe. Such devices are commonly used for quality assurance (QA) purposes, but using the contact probe over predetermined areas of interest on the structure can result in high-quality data collection that may be used beyond common QA practices such as inclusion in high-fidelity FEA. Physically contacting the structure is a concern for fragile or extremely compliant structures. Physical requirements to reach all sections to be measured is a constraint that may make scanning large structures infeasible. Imperfection collection via contact probe methods also often require more time to complete large structures compared to other collection methods while often producing a data set with significantly fewer measurements.

Noncontact surface geometry data collection methods have experienced increasing usage due to addressing some of the drawbacks previously mentioned with the contact probe collection method while also achieving acceptable accuracy. Laser scanning methods frequently used in research and industry use triangulation to record surface geometry. The distance from the laser emitter to the sensor and the offset angle, α , are both known. Whether the laser scanner is projecting a point or a line, the position of the surface is calculated based on sensor-collected reflections from the emitter. This method can capture large quantities of data with good accuracy but, depending on the value of the emitter-to-sensor distance and α , proximity to the scanning surface is often a requirement. For large structures, structures with small internal geometric features such as cavities, or concave surfaces that are difficult to access, this scanning may be infeasible. Structured light scanning methods also rely on triangulation to record surface geometry. Instead of projecting a single point or line, a pattern of light is projected onto the surface of interest. Stereoscopic cameras, offset from the projector by known distances and angles, record the reflected patterns. The imaged patterns are then processed to calculate position data about the scanned surface. Structured light scanning can capture entire surfaces in the field-of-view and therefore may capture surface data faster than some laser scanning methods. However, direct line-of-sight requirements still make large structures, internal cavities, or certain concave surfaces difficult to scan. The quantity of collected data can be similar to laser scanning methods and have similar accuracy. Structured light scanning methods conducted at NASA Marshall Space Flight Center (MSFC) were used for the surface geometry data collected for this work.

The result of collecting structured light scans and processing the data for the presently described work was a series of points in three-dimensional (3-D) space. Each point, marked by cartesian coordinates in X-, Y-, and Z-axes, represents a point on the surface of the physical test article. Two separate series of points were provided for analyzing each composite test article. The series of the exterior surface was referred to as the outer mold line (OML) despite the surface having been processed beyond the manufacturing stage through the addition of metallic load-introduction end rings. The scan of the interior surface was referred to as the inner mold line (IML). When using visualization software, the series of points from the surface geometry scans may resemble a large group of points in 3-D space. In this work, the group of points are referred to as a point cloud. The decision to separate the OML points from the IML points enabled easier processing and required fewer calculations to determine metrics of interest such as OML surface position averages and thickness values around the circumference of the test article. Separating the OML from the IML points, while convenient, was not necessary nor is required to extend the methods used in this work to other topics.

IV. Pre-processing and Visualization of Surface Geometry Data

Prior to implementing the surface geometry imperfections obtained from structured light scanning into a FEM model, surface scan data was processed using Python scripting to provide consistent data and reduce the likelihood for numerical errors. As each point cloud was unique, each file may have been processed slightly differently between the various test articles. Differences in processing between the IML and OML point clouds of the same test article were kept to a minimum. The IML and OML point clouds often differed only by the count of points removed from the series simply because the total number of points from the structured light scan were different between surfaces.

Simple operations such as plotting the surface data, interpolating the series of points onto a grid pattern to reduce computational time for plotting the surface data, truncating the point clouds to omit the end rings, and filtering the remaining points into a grid pattern are processing steps used to prepare the surface data and are presented next.

A. Pre-processing of the structured light scan point clouds

Plotting of the raw surface data received from the structured light scanning process is often the first task for inspecting the point clouds. Python 3.7 [9] with Matplotlib [10], NumPy [11], SciPy [12], and Pandas [13] libraries were used for this task, but other software able to read and perform operations on standard comma separated variable (csv) – type data format could also be used. Once the data in cartesian coordinates is read, a transformation to a cylindrical coordinate system simplifies much of the plotting and operations. In the present work, the Z-axis is the axis of revolution for the cylinder and thus is also the height, or axial, position. X- and Y- axes are the perpendicular axes which are transformed into the cylindrical radial- (R-) and circumferential- (θ -) coordinate axes. Note that a radius of 0.0 inches corresponds to the Z-axis. Transformation into cylindrical coordinates is often the only pre-processing operation conducted before direct visualization is possible. Other inspections, such as descriptive statistics including maxima, minima, and measures of distribution can be extracted. However, the end rings typically skew this type of information while descriptive data of the test article itself is usually of higher interest.

B. Interpolating onto a grid pattern for faster visualization

Structured light scanning methods may produce significantly more surface point measurements than are typically needed for visualizing the full-length scale of the test article. For example, CTA8.2 OML point cloud contained approximately 6.7 million points as received after scanning. Plotting all 6.7 million points for this size of test article is typically unnecessary and takes computationally more time than feasible to rapidly iterate through future point cloud operations. To ensure future operations on the data are able to be rapidly visualized, the point clouds are interpolated onto a rectangular grid in the axial-circumferential space using the `scipy.interpolate.griddata` function within the SciPy library. There are many options within the function for using interpolation techniques such as nearest-neighbor, linear, bilinear, etc. and for the pre-processing effort, the nearest-neighbor technique for radial position was sufficient for quickly visualizing the surface data.

C. Truncating the point cloud to exclude the metallic end rings

It was mentioned in Section IV. A. that the extracted descriptive statistics of the structured light scans in this work are skewed because of the inclusion of the end rings. Truncating the axial range of the scan data at the axial positions between the end rings removes the points associated with the end rings from the point cloud. After truncation, the effects of the end rings no longer skew the statistical description. Truncating the end ring points also no longer skew default maxima and minima values when contour plotting so that geometric surface variation is pronounced without intentionally altering the range of the color contours. In this work, truncation is usually set between 45 inches and 46 inches about the mid-length of the test article.

D. Sampling the truncated point cloud for sufficient point density

Prior to using Py_TIGIRS to implement geometry into FEM models, the truncated point clouds are often sampled to reduce the overall number of points in the OML and IML datasets. The reduction can be performed numerous ways, but the method is often dictated by how evenly spaced the original points are in each point cloud. If the surface scans have relatively evenly distributed point density, a simple random selection of the point cloud may be sufficient. However, if the density around regions of the test article vary, a more structured sampling method may be needed. In order to achieve sufficient data in both the OML and IML surfaces for the interpolation conducted in Py_TIGIRS, a grid sampling may be used. Similar to the interpolation for visualization mentioned in Section IV. B., a rectangular grid can be constructed in the circumferential-axial space to interpolate the radial positions at regular intervals. In this instance of interpolation, it has been found that using at least a linear interpolation method at minimum, opposed to the nearest-neighbor method previously used for plotting purposes only, returns a sufficiently sampled surface. Sampling the truncated point cloud may not be necessary depending on how the data in each point cloud is distributed, but sampling can add value by reducing potential for interpolation errors between the IML and OML surfaces within Py_TIGIRS.

V. Implementing Surface Geometry Data in FEM Models with Py_TIGIRS

The set of functions in the scripting tool Py_TIGIRS were conceived to be interchangeable and called only if the specific function was necessary to perform operations. In addition to the eight essential functions for implementing imperfections, there is the executed script that serves as a wrapper for the eight functions. The purpose of this wrapper is to provide a single-point execution for the tool and handle data from one function to the next. The eight functions that are called when Py_TIGIRS is executed are provided in Table 1 along with brief summaries of their purposes. The following sections are brief descriptions of the function, key inputs and outputs, and any user-defined settings.

Table 1. Functions and their purpose within Py_TIGIRS

Function	Purpose
Abaqus Input Modifier	Imports select model data from input file
Data Reader	Imports IML and OML surface scans
Radial Searcher	For each shell node, a “radial window” for IML and OML points is searched and returns lists of points with data
Potted End Handler	Extrapolates radial position for end-most known ring of nodes into the potting rings
Shell Element Thickness	Calculates average shell element thickness
Section Thickness Calculator	Alters core thickness to match calculated averages with section thickness
Section Thickness Sorter	Bins elements with similarly sized core and within predefined element sets
Abaqus Input Writer	Copies input file line-for-line with updated model parameters

A. Abaqus Input Modifier Function

The Abaqus Input Modifier Function opens and reads the nominal-geometry Abaqus input file to be implemented with geometric imperfections. The function reads the input file line by line and searches for Abaqus Keywords for information about the nodal positions, element-node connectivity, and shell section definitions of selected element sets. The nodal positions are transformed from the cartesian coordinate system into a cylindrical coordinate system. The Z-axis for both coordinate systems are aligned in this work and represents the axial position along the cylinder. The transformed nodal positions, connectivity, and shell section definitions are the primary output of the Abaqus Input Modifier Function. It was anticipated that not all regions of a cylindrical shell model would require imperfection implementation. Therefore, shell section and only element set names including predetermined words would be modified. Shell sections that had names that did not include the predetermined words would not be altered during the imperfection implementation. The predetermined words are important user-defined values in the Abaqus Input Modifier Function. Note that the nominal input file is not altered by the function as it is opened as read-only.

B. Data Reader Function

The Data Reader Function takes the filenames of the pre-processed OML and IML point clouds as user-defined variable names, opens the files, and imports the point cloud position data. Simple transformation into a cylindrical coordinate system, as previously conducted for the pre-processing visualization, is also performed in this function. Two separate data arrays, one for the IML and the other for the OML, are returned as output from the Data Reader Function.

C. Radial Searcher Function

From a broad perspective, Py_TIGIRS is written like an interpolation method. The Radial Searcher Function is the interpolation component between the nodal positions of the nominal geometry and the as-scanned imperfect surface geometry. The Radial Searcher Function performs the interpolation only on nodes that are assigned to elements in element sets labeled with the predetermined keywords used from the Abaqus Input Modifier Function.

For each node belonging to an element set that is to be implemented with imperfections, the Radial Searcher Function returns all points in the IML point cloud and in the OML point cloud that are within a bounded search-window. The searching direction is set as the radial axis, and the bounds of the search window are user-defined ranges of circumferential and axial values. For example, if nodes in a cylindrical shell model are nominally circumferentially spaced at every degree for a total of 360 nodes, the circumferential bounds of the search window can be specified to be ± 0.5 degree. A similar example for axial bounds and if nodes are spaced every one inch, then the search window can be set to be ± 0.5 inch. This particular example for circumferential and axial bounds utilizes nearly all of the available points in the IML and OML point clouds. The bounds may be smaller or larger and are determined by the user. Larger windows may return more points from each point cloud and therefore lessen the influence of outliers, while smaller windows may return fewer points (or even none) and subject the calculated average surface radial position to greater sensitivity to outliers.

The average radial position for the separate IML and OML point sets are calculated and recorded in association with each node. The averaged IML radial surface position and the averaged OML radial surface position are used to determine the midsurface position in the Radial Searcher Function and indirectly calculate thickness values in another function. To avoid the use of offsets in shell section definitions, the midsurface radial position is calculated using the

simple geometric midpoint of the IML and OML surface radial averages. The nominal node radial position is perturbed to be the calculated midsurface radial position.

D. Potted End Handler Function

As part of the point cloud pre-processing steps discussed in Section IV, the metallic end rings were removed from the point clouds used in Py_TIGIRS. A portion of each end ring, however, contains composite sandwich shell and is included in the nominal model. If the portion of the shell potted in each end ring were left in the nominal geometry configuration, there would be no guarantee of slope or position continuity in the surface of the shell model where the imperfect shell joins the perfect shell. Cylindrical shells are known to be sensitive to end conditions, and introducing a discontinuity of this type was both undesirable from a numerical accuracy perspective and not representative of the real test article. Since the scan data was of the metallic end ring surface and not the composite shell directly, it was decided to omit the data of the metallic end rings in the Radial Searcher Function.

The Potted End Handler Function was written as one solution to address not having surface scan data for the region of the composite shell within the end rings. The Potted End Handler Function takes the node closest to the end rings that was perturbed in the Radial Searcher Function and extrapolates the IML, OML, and the midsurface averaged radial positions to the remaining nodes of the same circumferential coordinate. While this method does not guarantee slope continuity, it has been observed that discontinuities are small compared to the scale of the test articles and end-condition effects are anticipated to be small.

There are no user-defined inputs for the Potted End Handler Function. The output of the function completes the interpolation process within Py_TIGIRS as all shell nodes of elements defined for composite facesheet sandwich core construction have been modified to include radial perturbations based on the structured light scan data.

E. Shell Element Thickness Function

Each node has radial average positions for the IML and OML surfaces as interpolated from the Radial Searcher Function. The thickness of the sandwich composite, defined as the IML radial average position subtracted from the OML radial average position for each node, can be implemented in the analysis model using nodal thickness definitions. One disadvantage with using nodal thickness definitions, however, is the lack of control in specifying which laminae experience altered thicknesses from the nominal definition. A nodal-based thickness variation would scale all the nominal thicknesses through the entire section definition instead of allowing for specific layers to alter thickness. As used for SBKF, Py_TIGIRS implements all thickness deviation from nominal in the honeycomb core. This assumption was based on observations that the cured ply thickness of the facesheet laminae were not varying regardless of the number of plies in a given cross-section. To accommodate the assumption that all thickness changes manifest in the core, an element-based thickness variation implementation method was selected.

The Shell Element Thickness Function iterates over each shell element associated with the predetermined keywords defined in the Abaqus Input Modifier Function. These elements are the same as those that contain nodes that were iterated through the Radial Searcher Function for interpolation of the IML and OML surface averages. Using an element-based thickness specification enables each shell element to be assigned a total shell thickness, and this value is then used by the next function. The total element thickness value is calculated to be the simple mean of the four node thickness values of the four nodes associated with the element. The thickness values for each element node are calculated as the difference between the OML and IML radial position averages previously calculated in the Radial Searcher Function. There is no user-defined input to the Shell Element Thickness Function, and the output is an array of elements and the calculated total shell thickness of each element.

F. Section Thickness Calculator Function

Since each element is associated with a nominal element set and therefore the nominal shell section definition, the number of plies, thicknesses for each ply, and materials for each ply in the element can be referenced to determine the change in honeycomb core thickness. Within the shell section definition, the thickness of each facesheet ply may be specified directly (kept as the nominal thickness value in this work) while only changing the honeycomb core thickness value. The ability to reference the nominal shell section definitions was essential to determine changes in core thickness for multiple stacking sequences used throughout the test article.

The Section Thickness Calculator Function iterates over each shell element to sum what the total element thickness is for the nominal configuration. The function then compares the nominal total shell thickness to the calculated total shell thickness based on scanned data. The difference between the two total shell thicknesses are added to the thickness value of the honeycomb core. If the total thickness from the scan is smaller than the nominal total thickness, the assigned core thickness will be less than the nominal core thickness. Alternatively, if the total thickness from the scan is larger than the nominal total thickness, the assigned core thickness will be more than the nominal core thickness.

There are no user-defined inputs for the Section Thickness Calculator Function and outputs are an array of elements and their altered core thickness.

G. Section Thickness Sorter Function

When each element with altered core thickness is assigned a unique element set and unique shell section definition, the preprocessing time after submitting the job to execute is prohibitive. To reduce the computational preprocessor expense, elements of the same stacking sequence and with similar core thickness values are grouped together into bins. There are many numerical schemes with which to determine the range of thickness values to group together. A method was developed that calculated the range of core thickness values and number of bins such that each consecutively thicker grouped-core bin would result in a core thickness change of approximately 0.001 inch. For example, if a stacking sequence had calculated core thickness values ranging from 0.19 inch to 0.20 inch, then ten bins are created with which to assign elements. The assigned core thicknesses for each bin take the midpoint of the individual bin range. Returning to the previous example, every element containing core thicknesses ranging from $0.190 \text{ inch} \leq \text{thickness} < 0.191 \text{ inch}$ would be grouped together into the first bin and assigned a core thickness value of 0.1905 inch. The second bin would consist of elements with core thickness values ranging from $0.191 \text{ inch} \leq \text{thickness} < 0.192 \text{ inch}$ and be assigned a core thickness value of 0.1915 inch. With this method, it is possible to have bins containing no elements if there is sufficient granularity in the calculated core thickness values.

Using the outlined method for the Section Thickness Sorter Function, the preprocessing computational time is significantly reduced while preserving most of the fidelity of the calculated thickness variation. There are no user-defined inputs to the Section Thickness Sorter Function, and the output is an array of nominal shell section definitions, the bins for the varying thickness shell section definitions, and the element belonging to each bin.

H. Abaqus Input Writer Function

The final function called within Py_TIGIRS is when all of the calculated and updated model information is written directly to an updated Abaqus input file. The Abaqus Input Writer Function opens the original input file and, for all lines that are unchanged from Py_TIGIRS, copies over into a new file line for line. When the function encounters the nodal position definition section, the radially perturbed nodal positions are written to the new input file after being transformed back into cartesian coordinates. The element-node connectivity remains unchanged and is therefore copied line for line. New element sets, associated with the binned shell section definitions, are also written to the input file after all nominal-model element sets are finished being copied to the new file. All shell section definitions from the Section Thickness Sorter Function are written to the new input file. Bins containing no elements are not written to the file to reduce the size of the input file. The function continues until the end of the nominal model input file is reached, and then all files are closed. The inputs to the Abaqus Input Writer Function are the filename for the geometrically imperfect input file, and there are no outputs from the function. Py_TIGIRS terminates upon conclusion of the Abaqus Input Writer Function.

VI. Results for CTA8.2B

Results from the pre-processing effort, and after processing with Py_TIGIRS, for CTA8.2B are provided. The pre-processed, truncated, and grid-interpolated structured light scan data for the IML point cloud, OML point cloud, midsurface calculation, and total sandwich thickness calculation are shown in Figure 2, Figure 3, Figure 4, and Figure 5, respectively.

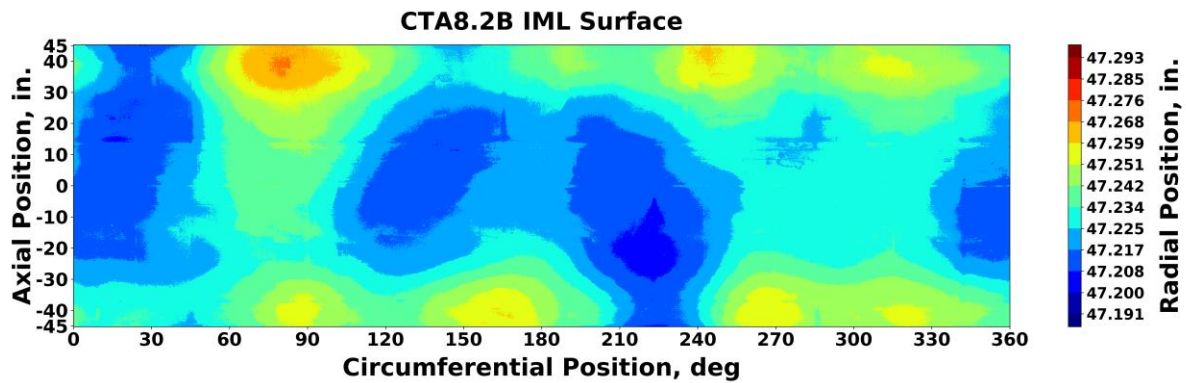


Figure 2. IML surface radial-position contours of CTA8.2B from structured light scans.

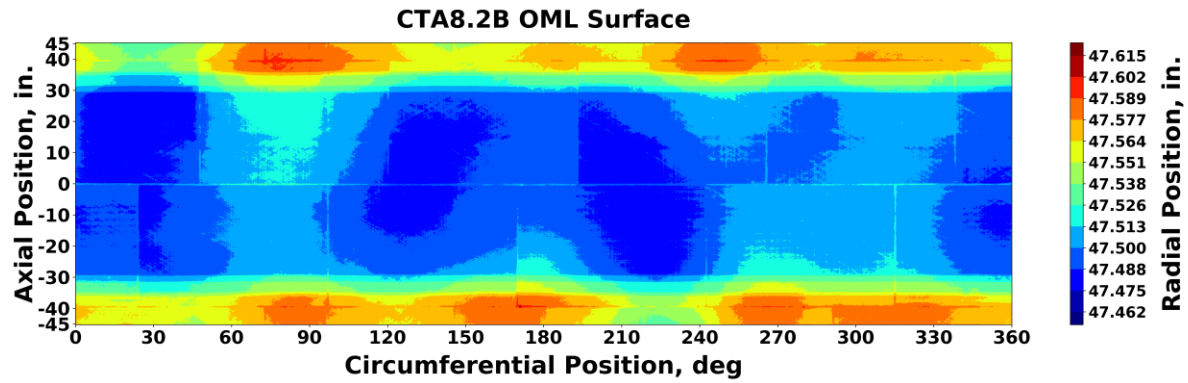


Figure 3. OML surface radial-position contours of CTA8.2B from structured light scans.

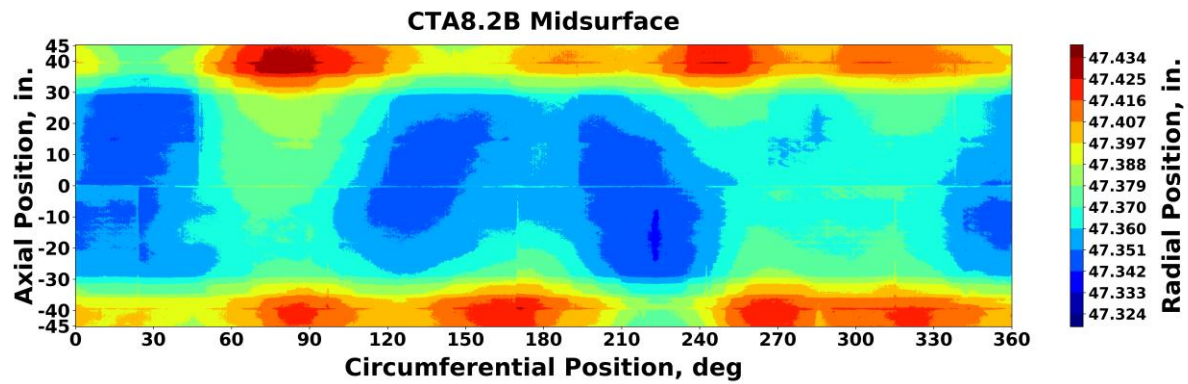


Figure 4. Midsurface radial-position contours of CTA8.2B calculated from structured light scans.

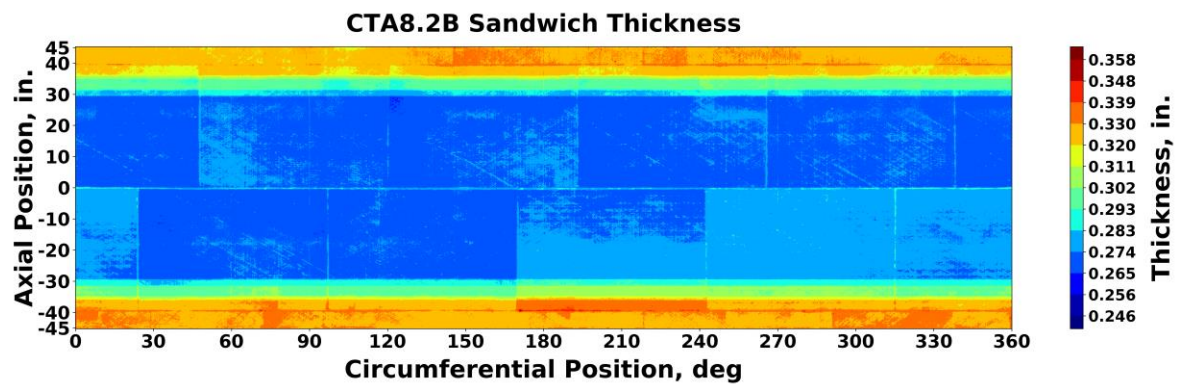


Figure 5. Total sandwich thickness contours of CTA8.2B calculated from structured light scans.

The IML surface contours are shown with a radial position range of just over 0.1 inch. More variation is observed in the OML surface due to the design and being manufactured on a constant-diameter tool. A slight undulation pattern is visible in the midsurface contour plot, and similar qualitative behavior is observed in both the IML and OML surface contours. Thicker regions near the ends are seen in Figure 5 where the transition from the layup near the potted region to the midlength-region layup occurs and is highlighted by the green contour regions outside of ± 30 inches. The rectangular grid pattern observed in Figure 5 is explained by the manufacturing method and is a result of the honeycomb core panels spliced during fabrication.

Contour plots of the midsurface and total sandwich thickness can also be created using data extracted from the analysis model after geometric imperfections have been implemented via Py_TIGIRS. The FEM model midsurface radial position contour plot is shown in Figure 6 and the model total shell thickness contour plot is shown in Figure 7. Data from the analysis model are taken at the element level and therefore are not aligned to the pre-processing interpolation grid used for the previous figures.

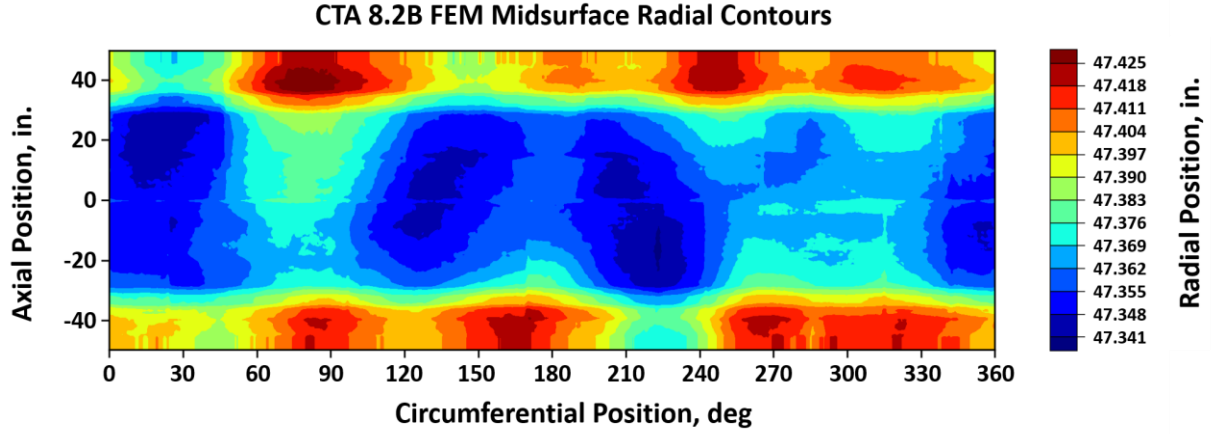


Figure 6. FEM model midsurface radial position contours of CTA8.2B from Py_TIGIRS.

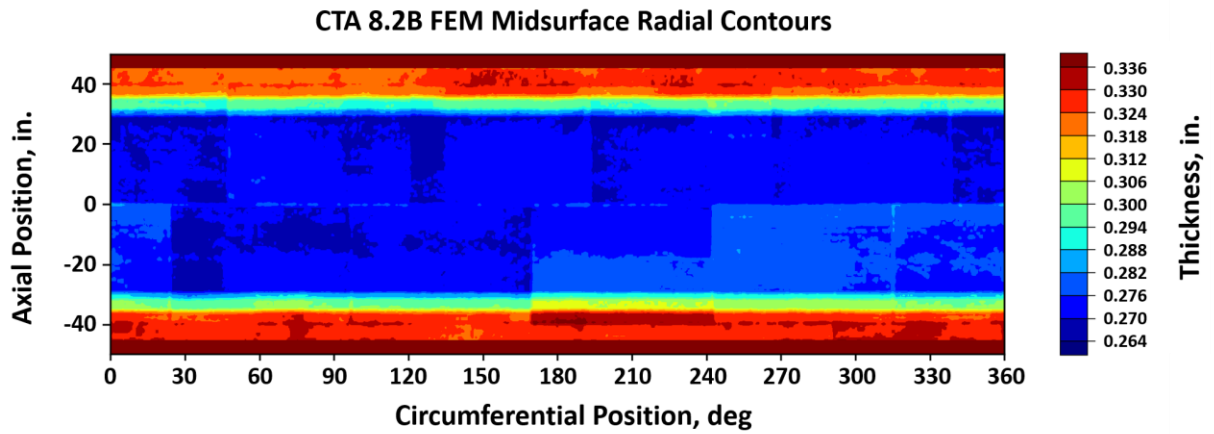


Figure 7. FEM model total shell thickness contours of CTA8.2B from Py_TIGIRS.

Plotting the element midsurface position and total thickness values accounts for the minor changes in value ranges between the pre-processing data and the analysis model extracted data. There is excellent agreement between the pre-processing midsurface and total thickness contours and the corresponding contours extracted from the analysis model.

VII. CTA 8.1, 8.2, and 8.3

Data from CTA 8.1, 8.2, and 8.3 are left for the final version of the proposed paper. All data is available from the structured light scans and the model analyses. The contour plots and corresponding discussion are omitted from the abstract for the sake of brevity.

VIII. Effects of Imperfections on Cylinder Buckling Analyses

The motivation to develop a tool like Py_TIGIRS is well established in literature. There is substantial work on the sensitivity to geometric imperfections on cylindrical shell buckling, and a summary of that body of work is outside the purpose of this work. Including surface geometric imperfections in analysis models, however, can significantly reduce the computationally predicted buckling load compared to the nominal geometry case when using nonlinear

analyses. A comparison of the normalized load versus normalized applied end shortening response for CTA8.2B is shown in Figure 8. The details of the analyses, while important to the overall modeling strategy for SBKF, are not the focus of this work. Instead, the significant decrease in predicted buckling load of the FEM model results with imperfections, approximately 85% of the nominal geometry analysis model, is the key observation. Using a tool such as Py_TIGIRS can facilitate the adoption of improved and updated buckling knockdown factors for cylindrical shells when surface geometric imperfection data are available. Such updated factors reduce conservatism in the design of thin cylindrical shell structures.

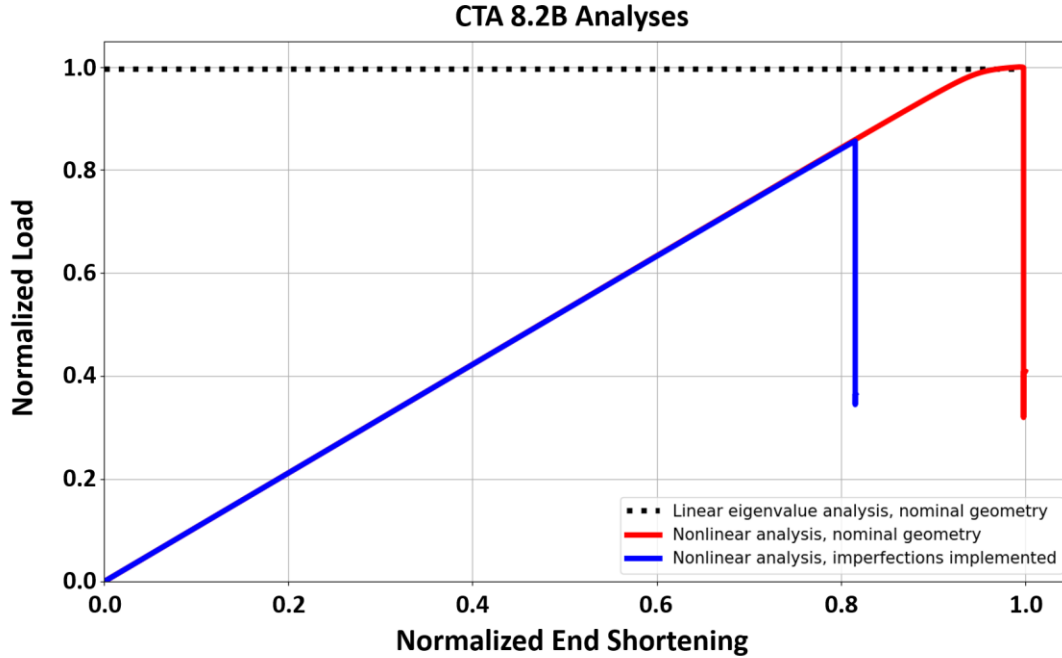


Figure 8. Various responses for axial compression numerical analyses of CTA 8.2B.

IX. Suggested Future Development

Py_TIGIRS may be enhanced in numerous ways. Future development is planned for enabling layer-by-layer thickness variation in shell definitions, the ability to process solid-element section thickness variation in model definitions, expansion to operate on structures beyond simple right-circular cylindrical shapes, numerical efficiency schemes to reduce the time required to perform standard interpolation, and scripting individual functions within Py_TIGIRS to operate in parallel. Though Py_TIGIRS was initially created for a specific geometry, sandwich construction, and end-use purpose, generalization of the methodology outlined in this work may enable inclusion of as-manufactured surface geometric imperfections for finite element analysis in many industries.

X. Concluding Remarks

Geometric imperfections are well known to lower the buckling load for certain types of thin cylindrical shell structures manufactured from metallic and composite materials. Different data collection methods that are in use were briefly discussed, and structured light scanning was used for all geometric surface data used in this work. Common pre-processing and visualization steps used in Shell Buckling Knockdown Factor Project (SBKF) were discussed, and the steps for preparing surface scans for implementation into a FEM model were described. The Python Tool for Implementing Geometric Imperfections in Reduced Structures (Py_TIGIRS), written specifically for the use with SBKF project, was described with eight functions performing steps to extract, modify, and write geometric imperfections for use in Abaqus input files. Results of the pre-processing methods and results from Py_TIGIRS were provided and compared for CTA8.2B, with other test articles 8.1, 8.2, and 8.3 to be included in the proposed paper. Excellent agreement between the visualized scan data and the FEM model-extracted geometry was demonstrated. A

brief example of why geometric surface imperfections are significant in nonlinear numerical analyses for thin cylinders in axial compression was provided. Future developments of Py_TIGIRS including expansion to structures of arbitrary geometry was discussed.

Acknowledgments

The author acknowledges Mr. Michael Lindell and Dr. Leonard Oremont (from NASA Langley Research Center) for thoughtful discussions and computer code examples provided throughout SBKF. Mr. John Ivester of NASA Marshall Space Flight Center is also recognized for performing all structured light scans and supplying the data used for this work.

References

- [1] Koiter, W. T., “The Stability of Elastic Equilibrium,” (in Dutch), Ph.D. Dissertation, Delft University, 1945: English translation AFFDL-TR-70-25, Feb. 1970.
- [2] Hilburger, M. W., “Developing the Next Generation Shell Buckling Design Factors and Technologies,” 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Honolulu, Hawaii. April 23-26, 2012, <https://doi.org/doi:10.2514/6.2012-1686>.
- [3] Przekop, A., Schultz, M. R., Kosztowny, C. J. R., Song, K., Lindell, M. C., Hilburger, M. W., and Rudd, M. T., “Design and Analysis of Buckling-Critical Large-Scale Sandwich Composite Cylindrical Test Articles,” NASA/TM-2020-000000, to be published.
- [4] Almroth, B. O., “Influence of Imperfections and Edge Restraint on the Buckling of Axially Compressed Cylinders,” 7th Structures and Materials Conference, Cocoa Beach, Florida, April 18-20, 1966, <https://doi.org/doi:10.2514/6.1966-1702>.
- [5] Amazigo, J. C., and Budiansky, B., “Asymptotic Formulas for the Buckling Stresses of Axially Compressed Cylinders With Localized or Random Axisymmetric Imperfections,” *Journal of Applied Mechanics*, Vol. 39, No. 1, pp. 179–184, <https://doi.org/10.1115/1.3422608>.
- [6] Hilburger, M. W., and Starnes, J. H., “Effects of Imperfections of the Buckling Response of Composite Shells,” *Thin-Walled Structures*, Vol. 42, No. 3, 2004, pp. 369–397, <https://doi.org/10.1016/j.tws.2003.09.001>.
- [7] Abaqus Standard, Version 6.14, Dassault Systemes Simulia Corp., Providence, Rhode Island, USA, 2014.
- [8] Arbocz, J., and Williams, J. G., “Imperfection Surveys on a 10-ft-Diameter Shell Structure,” *AIAA Journal*, Vol. 15, No. 7, 1977, pp. 949–956.
- [9] Van Rossum, G., and Drake, F. L., *Python 3 Reference Manual*, CreateSpace, Scotts Valley, CA, 2009.
- [10] Hunter, J. D., “Matplotlib: A 2D Graphics Environment,” *Computing in Science & Engineering*, Vol. 9, 2007, pp. 90–95, <https://doi.org/10.1109/MCSE.2007.55>.
- [11] Oliphant, T. E., *A Guide to NumPy*, Trelgol Publishing USA, 2006.
- [12] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walk, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, I., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, Vol. 17, 2020, pp. 261–272, <https://doi.org/10.1038/s41592-019-0686-2>.
- [13] McKinney, W., “Data Structures for Statistical Computing in Python,” 9th Python in Science Conference, 56-61, 2010, <https://doi.org/10.25080/Majora-92bf1922-00a>.