A framework for geometric modeling and structural analysis of composite laminates

Onur Rauf Bingol

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Mechanical Engineering Commons

Recommended Citation
https://lib.dr.iastate.edu/etd/17404

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
A framework for geometric modeling and structural analysis of composite laminates

by

Onur Rauf Bingol

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Mechanical Engineering

Program of Study Committee:
Adarsh Krishnamurthy, Major Professor
Ming-Chen Hsu
James Oliver
Baskar Ganapathysubramanian
Soumik Sarkar

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2019

Copyright © Onur Rauf Bingol, 2019. All rights reserved.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>x</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xi</td>
</tr>
<tr>
<td><strong>CHAPTER 1. INTRODUCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1 Generation of geometric models</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2 Generation of structural analysis models</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Dissertation Outline</td>
<td>3</td>
</tr>
<tr>
<td>1.3 References</td>
<td>4</td>
</tr>
<tr>
<td><strong>CHAPTER 2. AN INTEGRATED FRAMEWORK FOR SOLID MODELING AND STRUCTURAL ANALYSIS OF LAYERED COMPOSITES WITH DEFECTS</strong></td>
<td>5</td>
</tr>
<tr>
<td>2.1 Abstract</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Introduction</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Background and Related Work</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Framework for Modeling 3D Composite Structures</td>
<td>11</td>
</tr>
<tr>
<td>2.4.1 Components of the Framework</td>
<td>12</td>
</tr>
<tr>
<td>2.4.2 Integrating CAD and FE Analysis</td>
<td>12</td>
</tr>
<tr>
<td>2.4.3 CAD Model of the Composite Laminate</td>
<td>14</td>
</tr>
<tr>
<td>2.4.4 Incorporating Stiffeners into the CAD Model</td>
<td>16</td>
</tr>
<tr>
<td>2.4.5 Incorporating Defects into the CAD Model</td>
<td>18</td>
</tr>
<tr>
<td>2.4.6 Processing NDE Data to Identify Delaminations</td>
<td>20</td>
</tr>
<tr>
<td>2.4.7 Structural Analysis Using the Cohesive Model</td>
<td>20</td>
</tr>
<tr>
<td>2.4.8 Model Builder API</td>
<td>23</td>
</tr>
<tr>
<td>2.5 Application of the Framework to Model Composites</td>
<td>24</td>
</tr>
<tr>
<td>2.5.1 Sample Composite with Impact Damage</td>
<td>24</td>
</tr>
<tr>
<td>2.5.2 3D Models of Composites</td>
<td>25</td>
</tr>
<tr>
<td>2.5.3 Incorporating Stiffener to the 3D Composite Model</td>
<td>26</td>
</tr>
<tr>
<td>2.5.4 Incorporating Delaminations to the 3D Composite Model</td>
<td>27</td>
</tr>
<tr>
<td>2.5.5 Incorporating Fiber Breakage</td>
<td>28</td>
</tr>
<tr>
<td>2.5.6 Finite Element Analysis of the 3D Composite Model</td>
<td>28</td>
</tr>
</tbody>
</table>
2.6 Extensions and Future Work ................................................. 32
  2.6.1 Shell Solid Coupling ................................................. 32
  2.6.2 Framework Extension ............................................... 34
  2.6.3 Limitations ............................................................ 36
  2.6.4 Future Work .......................................................... 37
2.7 Conclusions ................................................................. 37
2.8 References ................................................................. 38

CHAPTER 3. NURBS-PYTHON: AN OPEN-SOURCE OBJECT-ORIENTED NURBS MODELING FRAMEWORK IN PYTHON ......................................................... 41
  3.1 Abstract ................................................................. 41
  3.2 Introduction ............................................................. 41
  3.3 Design Considerations .................................................. 43
  3.4 A Brief Introduction to NURBS ........................................... 44
  3.5 Components of the Framework .......................................... 47
    3.5.1 Core Components and Data Structures ............................ 47
    3.5.2 Curve and Surface Evaluation .................................... 49
    3.5.3 Shape Splitting and Knot Insertion ............................... 52
    3.5.4 Surface Generator .................................................. 53
    3.5.5 Exporting and Importing Data .................................... 54
    3.5.6 Visualization Components ........................................ 55
    3.5.7 Testing and Algorithm Validation ................................ 56
  3.6 Code Examples ........................................................... 57
  3.7 Additional Components of the Framework ............................. 61
    3.7.1 Surface Generator Customization ................................ 61
    3.7.2 Visualization Customization ...................................... 62
  3.8 Additional Code Examples .............................................. 62
  3.9 Performance Metrics .................................................... 65
  3.10 Conclusions and Future Work ......................................... 66
  3.11 References ............................................................. 67

CHAPTER 4. MODELING AND ISOGEOMETRIC ANALYSIS OF LAMINATED COMPOSITE STRUCTURES ................................................................. 69
  4.1 Abstract ................................................................. 69
  4.2 Introduction ............................................................. 70
  4.3 Modeling laminated composites using NURBS volumes .............. 72
    4.3.1 Construction of individual layers ................................ 73
    4.3.2 Construction of laminates from layers ............................ 78
    4.3.3 Validation of the methods ......................................... 78
  4.4 Isogeometric analysis of laminated composites ....................... 79
    4.4.1 Cantilever bending of a 3-dimensional beam ...................... 81
    4.4.2 Cantilever bending of a 3-dimensional doubly-curved plate ... 82
    4.4.3 Effect of layer generation methods on IGA ....................... 84
  4.5 Conclusion .............................................................. 85
  4.6 References .............................................................. 85
### CHAPTER 5. SUMMARY, DISCUSSION AND FUTURE WORK

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Summary and Discussion</td>
<td>89</td>
</tr>
<tr>
<td>5.2 Future Work</td>
<td>91</td>
</tr>
<tr>
<td>5.3 References</td>
<td>92</td>
</tr>
</tbody>
</table>

### APPENDIX. PUBLISHED SOFTWARE

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>93</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table 3.1</th>
<th>Comparison of evaluation time between interpreted and Cython-compiled versions of NURBS-Python. Sample sizes: $S_{\text{curve}} = 16384$ and $S_{\text{surface}} = 1048576$.</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>Comparing the accuracy of the surface reconstruction methods; our proposed method, interpolation and least squares (LS) approximation, using Hausdorff and minimum distance metrics.</td>
<td>79</td>
</tr>
</tbody>
</table>


LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Material breakdown of Boeing 787, from Bouvet (2017).</td>
<td>1</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>Illustration of the steps required to incorporate defects into composite models and perform structural finite element analysis.</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Components of the layered composite modeling framework. The CAD Model Builder is used to generate a layered composite structure and the FE model builder generates a script that can assemble the layers, apply appropriate boundary conditions, and perform structural finite element analysis.</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Layer generation and bonding steps of the automated framework.</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Illustration of creating Layer elements from (a) a parametric surface (i.e. a NURBS surface) and, (b) the existing layers by using their offset or original faces as the mold.</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Illustration of the process of adding a hat stiffener onto the existing composite laminate. After placing the hat stiffener mold (orange) on top of the existing layered structure, a new mold (yellow) is generated for the subsequent layers which will be placed on top of the stiffened structure. For this illustration, new layers generated in positive (upward) direction use this updated mold shape, whereas the new layers generated in negative (downward) direction would use the planar-shaped mold.</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>Illustration of imprinting delamination outline to the faces. User can choose any face to imprint delamination. Faces are defined by offset and original faces.</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>Illustration of splitting layers using a sheet body. The Sheet body can be a NURBS surface or a plane defined a point and a direction vector.</td>
<td>19</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>If the fiber breakage does not extend to the end of the composite region, the surface is extended to the closest edge. Appropriate tie boundary condition is then applied to the intact (pink) region.</td>
<td>20</td>
</tr>
<tr>
<td>Figure 2.9</td>
<td>Illustration of the cohesive model used for delamination analysis.</td>
<td>21</td>
</tr>
</tbody>
</table>
Figure 2.10  (a) A photograph of the composite sample analyzed using bi-directional ultrasonic testing scan. The yellow outline shows the region of interest with the size of $2 \times 2$ in$^2$ indicating the location where the ultrasonic testing scan was run. The top view of the ultrasonic scan data is incorporated inside the yellow outline for illustration of the delamination shape. (b) The isometric (left) and side (right) views of the volumetric rendering of the ultrasonic data of sample indicating the delamination shape in red color. The side view shows the delamination between the layers after the composite laminate is damaged.

Figure 2.11  Some examples of the 3D composite laminate models generated by the automated framework. Individual layer thickness is 0.199mm.

Figure 2.12  Thickness analysis of the stiffened layers generated using (a) sweeping operation, and (b) thickening operation.

Figure 2.13  5-layered planar composite laminates illustrating fiber breakage of different shapes in the 3rd layer. The user inputs a list of coordinates that defines the shape of the fiber breakage and the specific layer. The framework then splits the corresponding layer to emulate the fiber breakage.

Figure 2.14  Bending analysis of a 8-layered composite laminate generated by the automated framework. (a) Location of the delamination on the edge between the 2nd and 3rd layers with the markers representing the fixed boundary conditions for each layer, (b) Loading setup for static bending represented by the arrows along the -z direction and markers on the edge representing the fixed boundary conditions, (c) Fully deformed composite laminate after static bending, (d) Details of the layers at the delaminated edge.

Figure 2.15  Force (N) vs Displacement (mm) graphs comparing delaminated and non-delaminated 8-layered composite structures at different locations.

Figure 2.16  Force vs. Displacement curve comparing the effect of fiber breakage on the same composite laminate illustrated in Figure 2.13 with the stacking sequence of [0 -45 0 45 0]. The ”breakage” label corresponds to the model with the fiber breakage and ”intact” label corresponds to the same model without fiber breakage. As expected, the composite laminate with the fiber breakage shows slightly higher deflection than the one without fiber breakage. The inset graph is the zoomed version that shows the difference between the curves.

Figure 2.17  Bending of the stiffened and delaminated 8-layered composite laminate generated by the automated framework using the stacking sequence of [0/ -90/90/0/0/90/ -90/0].
Figure 2.18 8-layered delaminated laminate generated by the automated framework with a trapezoidal stiffener element. (a) shows location of the delaminations on the edge between the 1st and 2nd layers, and under the stiffener in the middle of 4th and 5th layers with the markers representing the fixed boundary conditions for each layer, (b) shows the boundary conditions and the loading setup, (c) shows the edge delamination on the deformed composite laminate after static bending, and (d) displays a cut-view of the delaminated section between 4th and 5th layers (position x = 10.99, deformation scale factor = 20, showing only feature edges)...

Figure 2.19 Bending test of a 8-layered delaminated curved composite laminate generated by the automated framework. (a) shows the location of the delamination inside the laminate model, (b) shows the fixed edges and the loading direction, (c) shows the displacement field (U_magnitude), and (d) displays a cut-view of the delaminated section between 2nd and 3rd layers (position x = -7.60, deformation scale factor = 5, showing only feature edges)...

Figure 2.20 Shell-solid coupling of the 5-layered planar composite laminate region with the shell model. (a) shows the boundary conditions and loading setup of the shell model and the 5-layered composite laminate constructed by cutting an arbitrary rectangular region on the shell model, (b) shows the displacement field (U_magnitude) of the shell model meshed coarsely (3.0 mm), (c) shows the displacement field of the constructed 5-layered planar composite laminate, and (d) displays the stress field of the same region...

Figure 3.1 Class diagram showing the inheritance of Curve classes...

Figure 3.2 Class diagram showing the inheritance of Surface classes...

Figure 3.3 Class diagrams showing the inheritance of abstract Evaluator, surface generator and visualization classes...

Figure 3.4 Generating a triangulated surface with NURBS-Python using \( \delta = 0.25 \). (a) Positions of the evaluated points on the parametric space, (b) triangulated surface on the parametric space, (c) A scatter plot on the 3D space illustrating the positions of the individual evaluated surface points. (d) Triangulated surface in 3D...

Figure 3.5 NURBS curves and surfaces generated by the library and plotted using Matplotlib implementation of the visualization component. (a) A circular NURBS shape generated from 7 control points, (b) Bzier decomposition of the circular shape, each curve segment is colored differently, (c) A cylindrical NURBS surface, (d) Bzier decomposition of the cylindrical surface, each surface patch is colored differently. The colors on the decomposed shapes are randomly generated...

Figure 3.6 An unclamped curve plotted using Matplotlib implementation of the visualization component. Blue dotted-dashed line shows the control points polygon and the black solid line shows the evaluated curve...
Figure 3.7 NURBS multi-surfaces (*MultiSurface* class) generated by the library and plotted using Plotly implementation of the visualization component. (a) Heart valve modeled from 3 surfaces (Xu et al., 2018). (b) Wind turbine blade modeled from 28 surfaces (Herrema et al., 2017). (c) Human heart modeled from 297 surfaces (Krishnamurthy et al., 2016).

Figure 3.8 The generated surface is visualized using Matplotlib implementation of the visualization component. The control points polygon and the legend are removed from the figure using the visualization configuration class.

Figure 4.1 Translation of control points to generate offset surfaces. The blue grid is the control points grid of the progenitor surface, the black grid is the control points grid of the offset surface. Tangent vectors (dark orange) and a single translation vector (cyan) are computed for each control point on the progenitor grid.

Figure 4.2 Cantilever bending of a 3-dimensional beam using *gIGA* library (a) initial geometry, (b) final geometry after bending.

Figure 4.3 Displacement convergence analysis for 3-dimensional beam.

Figure 4.4 Displacement gradient on the control points grid for 3-dimensional beam.

Figure 4.5 Cantilever bending of the bowl-shaped plate. Green is the initial geometry and blue is the final geometry after deformation.

Figure 4.6 Displacement of the control points at the free end of the bowl-shaped plate.
ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis.

First and foremost, I would like to thank my supervisor, Dr. Adarsh Krishnamurthy for his guidance, patience and support throughout this research and the writing of this thesis. His insights, invaluable expertise and constructive comments have encouraged me throughout my graduate studies and most importantly, helped me to look forward more positively.

I would also like to thank my committee members for their efforts and contributions to this work: Dr. Ming-Chen Hsu, Dr. James Oliver, Dr. Baskar Ganapathysubramanian and Dr. Soumik Sarkar. I would like to specially thank Dr. Hsu for his guidance at the final stages of my graduate studies and Dr. Sarkar for his mentorship throughout my graduate career.

I would like to thank my teammates; Arian Jafari, Anushrut Jignasu, Sambit Ghadai, Harshil Shah and Aditya Balu for their great support in deliberating over our problems and findings, as well as providing happy distraction to rest my mind outside of my research. I will never forget the fruitful conversations. I would also like to thank my friends in Ames; Hakan Yavas, Simge Cinar, Onur Seker, Pinar Seker, Muslum and Aliye Ilgu, Koray Ozcan and Halil Ibrahim Fedakar for helping me sorting out the issues whenever it is needed.

Finally, my family. You have been the greatest supporter in all my life. I could never thank you enough for your wise counsel and sympathetic ear. I am so lucky to have you on my side.
ABSTRACT

Laminated fiber-reinforced polymer (FRP) composites show considerable promise in structural applications due to their good combination of low weight and high strength. However, the manufacturing costs of laminated composites is significantly higher than their metallic counterparts. As a consequence, estimating the residual life of composites becomes critical, and can enable reusability in applications that demand lower mechanical strength requirements. One of the major factors affecting the residual life of the laminated composites is the defects introduced during manufacturing or in service. A common way of determining defects in the composite laminates is using non-destructive evaluation (NDE) techniques. In this study, a framework for modeling and structural analysis of composite laminates is presented. The framework follows the laminate manufacturing process and incorporates structural elements, such as stiffeners, as well as defects, such as delaminations, determined using NDE techniques. Each layer composing the laminate is modeled separately and combined to generate the final laminate. The layer combination process is called bonding and involves computation of boundary conditions for the constitutional model being selected for the analysis. Then, the final laminate model and the computed boundary conditions are used during the structural analysis. The initial framework used commercial off-the-shelf (COTS) software, i.e. 3D ACIS Modeler for 3-dimensional modeling and SIMULIA Abaqus for structural analysis via finite element modeling. The framework was then extended to use the NURBS library, NURBS-Python, and the isogeometric analysis library, gIGA, which were developed as a part of this study and released as free and open-source software on GitHub. Using NURBS for modeling and isogeometric analysis for structural analysis provide several advantages, such as directly operating on the exact geometry, and therefore; achieving better estimations on interlaminar and intralamellar stresses and strains, which has significant importance in determining the residual life of the composite laminates.
CHAPTER 1. INTRODUCTION

In the modern world of manufacturing, laminated fiber-reinforced polymer (FRP) composites are being increasingly used due to their excellent combination of high strength and low weight. Recent developments show that replacing metallic components with carbon FRP composites has more advantages in long term use despite their initial high cost. Considering aircraft structures, FRP composites enable improvements in fuel efficiency as well as flight control and performance. For example, 50 percent of the Boeing 787 (see Figure 1.1), mainly the structural components such as fuselage and wings, is made of composite materials ranging from carbon and glass fiber-based laminates, and sandwich structures (Bouvet, 2017).

Figure 1.1 Material breakdown of Boeing 787, from Bouvet (2017).
From engineering perspective, the major limiting factor of switching to composite materials is their high cost, since composite alternatives of metallic parts can be up to 10 times more expensive. In addition, it is very hard to recycle a composite material in contrast to their metallic counterparts. As a result, residual life estimation of the composite structure becomes critical. Residual life assessment can determine whether it is required to replace the part under given operating conditions, i.e. the structural deterioration sustained by the part during production as a result of manufacturing defects or damage in service.

In this study, we create a modeling and analysis framework to determine the residual life of laminated (or namely, multi-layer) composites used in aircraft structures. The framework virtually follows the manufacturing process for composite structures, incorporates structural elements such as stiffeners, defects such as delaminations, and analyze the resulting virtual structure using structural analysis methods.

1.1 Overview

The modeling and analysis framework follows the manufacturing steps of the laminated composite structures. Each layer is generated separately with a designated fiber orientation and properties, and then, bonded together to form a laminate. This approach is called layer-by-layer model generation in this work. The framework can be split into two separate parts; geometric model generation and preprocessing for structural analysis.

1.1.1 Generation of geometric models

Geometric models, namely CAD models, are generated within the framework using a NURBS (Non-Uniform Rational Basis Spline) surface as input. NURBS are the de facto standard for surface representation. The input NURBS surface model is used as a mold to generate solid/volumetric representation of the initial composite layer/lamina. The initially generated composite layer can be used as a mold to generate the latter composite layers. These layers can still be generated using the NURBS surface mold input. After the layer generation, defects and structural support elements
can be incorporated into the model. Initial focus of this work is to incorporate delaminations as defects and stiffeners as the structural support elements. This process called *layer-by-layer model generation* and is explained in detail in Chapter 2 as *CAD Model Generation*. The layer-by-layer model generation process allows users to generate the models with minimal effort and imposes no restrictions on the final laminate geometry.

1.1.2 Generation of structural analysis models

Structural analysis is a crucial step to determine the residual life of the composite laminates. The models generated for analysis require not only the geometric model but also additional data for the layer-layer interactions, i.e. information on adjacent faces and their constitutive model for determining the interaction type. Thus, the analysis model generator *bonds* the geometric models representing the layers together, replicating the process of gluing composite layers or combining pre-impregnated composite sheets in a mold to manufacture the final laminate.

The bonding step detects the adjacent faces of the layers combined in the final laminate model. Since, layer generation heavily depends on the user input and due to its flexibility in their generation, the adjacent faces needed to determined can be more than one depending on the composed structure of the layer or incorporation of defects and structural elements. After detection of the adjacent faces, any constitutive model chosen for the specific analysis scenario can be applied to the laminate model. Finally, the structural analysis model containing the geometric model can directly be analyzed using user-preferred analysis suites.

1.2 Dissertation Outline

This study aims to develop a software framework for modeling and analysis of laminated composite structures which are commonly used in aircraft and space industries. Since the materials used to manufacture laminated FRP composites are expensive in contrast to their metallic counterparts, in addition to laminated composites' limited reusability, it becomes extremely important to determine their residual life. As a result, it would be possible to determine an approximate
time for servicing or replacing the aircraft component or find another application for composite structures which are not suitable for aircraft industry. The framework integrates a state-of-the-art solid modeling kernel and a finite element analysis suite, which is explained in detail in Chapter 2.

The initial framework is created using state-of-the-art modeling and simulation software. During the development, it was observed that the modeling software used may not be directly suited for the layer-by-layer model generation approach. Thus, a general purpose, modular and extensible geometric modeling framework is developed using NURBS mathematical model. The NURBS model can provide the exact representation of the composite laminate geometries. The new framework is explained in Chapter 3.

Finally, the geometric modeling framework is extended for structural analysis of the composite structures using isogeometric analysis. Isogeometric analysis method can directly operate on the NURBS geometries, providing more detailed and exact results in compared to regular finite element analysis. A brief description and the operating principles of isogeometric analysis framework with some initial results are outlined in Chapter 4.

1.3 References

CHAPTER 2. AN INTEGRATED FRAMEWORK FOR SOLID MODELING AND STRUCTURAL ANALYSIS OF LAYERED COMPOSITES WITH DEFECTS

A paper published in *Computer-Aided Design*

Onur Rauf Bingol, Bryan Schiefelbein, Robert J. Grandin, Stephen D. Holland and Adarsh Krishnamurthy

2.1 Abstract

Laminated fiber-reinforced polymer (FRP) composites are widely used in aerospace and automotive industries due to their combined properties of high strength and low weight. However, owing to their complex structure, it is difficult to assess the impact of manufacturing defects and service damage on their residual life. Non-destructive evaluation (NDE) of composites using ultrasonic testing (UT) can identify the presence of defects. However, manually incorporating the damage in a CAD model of a multi-layered composite structure and evaluating its structural integrity is a tedious process. We have developed an automated framework to create a layered 3D CAD model of a composite structure and automatically preprocess it for structural finite element (FE) analysis. In addition, we can incorporate flaws and known composite damage automatically into this CAD model. The framework generates a layer-by-layer 3D structural CAD model of the composite laminate, replicating its manufacturing process. The framework can create non-trivial composite structures such as those that include stiffeners. Outlines of structural defects, such as delaminations detected using UT of the laminate, are incorporated into the CAD model between the appropriate layers. The framework is also capable of incorporating fiber/matrix cracking, another common defect observed in fiber-reinforced composites. Finally, the framework can preprocess the resulting 3D CAD models with defects for direct structural analysis by automatically applying the
appropriate boundary conditions. In this paper, we show a working proof-of-concept of the framework with capabilities of creating composite structures with stiffeners, incorporating delaminations between the composite layers, and automatically preprocessing the CAD model for finite element structural analysis. The framework will ultimately aid in accurately assessing the residual life of the composite and making informed decisions regarding repairs.

2.2 Introduction

Laminated fiber-reinforced polymer (FRP) composite materials are being increasingly used in automobile and aircraft industries due to their high strength-to-weight ratios. Recent developments in composite production allow replacement of the structural elements of high performance air and ground vehicles with composite counterparts. An example of these developments is the composite wings and fuselage of the Boeing 787 Dreamliner. Due to the increasing use of composites in critical structural parts of such vehicles, it is important to assess the residual strength of composites, in the presence of production defects or in-service damage. Ultrasonic Non-Destructive Evaluation (NDE) is the preferred method for identifying composite defects such as delaminations. Although ultrasonic testing can be used to identify the presence of such defects, in order to determine the structural integrity of the composite, the damage needs to be modeled. However, there are no reasonable automated methods to create a concrete CAD representation of the composite structure and then incorporate a model of the damage to evaluate their structural integrity.

In this paper, we propose an automated framework to model the composite structure using CAD modeling tools and incorporate defects measured using ultrasonic testing. The framework can then build a structural finite element (FE) model that can be used to assess the residual strength of the composite laminates (Figure 2.1). Performing FE analysis of the complete layered composite structure for large-scale components such as aircraft fuselage is prohibitively expensive. Hence, we focus on a small region of the composite structure to perform the layered FE analysis. This layered region can then be attached to the shell model of the entire structure using suitable boundary conditions. The modeling framework is a class library that can perform virtual manufacturing of
small regions of laminated composite structures. It can create a detailed layer-by-layer CAD model and a corresponding script to preprocess the layers for FE analysis. In addition, it can insert flaws into the layer assembly to represent the structural significance of defects.

The framework provides a set of functions that operate analogous to the manufacturing process for composite laminates. Manufacturing a composite laminate involves creating a mold, and then placing multiple layers of fiber over the mold, and gluing them together. The framework includes classes representing such a layer, and implements operations such as creating a layer that follows a mold shape, creating a layer that follows a previous layer’s shape, and bonding layers together with or without a defect. The multiple laminae in the structure are generated by offsetting layers from the mold shape, represented using non-uniform rational basis spline (NURBS) surfaces. The framework abstracts the CAD operations for creating a multi-layered laminate structure, which can get tedious if each layer is manually created. In addition, manual creation of the layered structure for curved laminates can lead to small gaps between the layers, which can lead to failure of the FE analysis. The framework overcomes these issues by keeping track of the offset faces of each layer and using them as mold surfaces for any subsequent layer, ensuring that the surfaces between the layers are exactly the same for any two adjacent layers.

Figure 2.1 Illustration of the steps required to incorporate defects into composite models and perform structural finite element analysis.
CAD-based finite element analysis generally involves three major steps: (1) Creating or loading a solid model, (2) applying boundary conditions, and (3) generating a mesh and solving for a numerical solution. These three steps are usually performed manually, and for many simple models, it is sufficient. For a detailed layer-by-layer solid model of a laminate—especially one with defects—it is prohibitively tedious, complicated, and error prone to manually apply the correct boundary condition to each boundary. Automatically applying all the correct boundary conditions can be very difficult in practice. The challenge is to create the boundary condition between the two surfaces as they are being created or assembled. Unfortunately, finite element analysis software do not generally allow assigning of boundary conditions until the entire model is complete (and if they do, it may be incorrect if the face numbering subsequently changes due to model changes). Hence, the intended boundary conditions need to be stored during the model construction phase, and then assigned later once the model is complete. Our framework keeps track of the layers during the model construction process and correctly assigns the boundary conditions.

The framework provides a highly customizable and user-friendly systems solution to the problem of structural analysis of laminated composites. It makes use of industry-standard tools to develop a well-defined, structured system based on the manufacturing process of composite laminates. The framework is highly flexible to implement new features or customize existing ones for modeling different aspects of composite structures. In order to automate the finite element analysis, the framework auto generates code to apply the appropriate boundary conditions between the layers of the laminated structure. The main contributions of this paper include:

- A composite CAD model builder that can create layer-by-layer CAD boundary representation of a composite structure from user-defined instructions that follow the composite manufacturing process. The CAD model builder supports creation of curved composites and composite structures with stiffeners.
- An automated pipeline to incorporate concrete representations of composite defects such as delaminations and fiber breakage.
• A finite element model builder that can generate a script to assemble the layered composite structure and to apply appropriate boundary conditions between the composite layers for both intact and defect regions.
• A code generation architecture that permits the CAD and FE models to be generated in parallel.

This paper is arranged as follows. In Section 2.3, we highlight some of the related work relevant to our modeling framework. In Section 2.4 we discuss the different components of the composite modeling framework. We then show the application of this framework in modeling 3D representations of the composite laminates and perform structural analysis in Section 2.5. Finally in Section 2.6, we outline some methods by which the proposed framework can be extended to other CAD and FE packages.

2.3 Background and Related Work

We follow the FRP composite laminate manufacturing process to create an easy-to-use API for designers. A FRP laminate is composed of layers (or plies) of fibers, such as glass or carbon fibers, embedded in a matrix material, such as epoxy resin. The layers with different lay-up orientation of fibers are glued to each other using a predefined stacking sequence for desired mechanical strength. After the lay-up process, the composite laminate is vacuum-molded or heated in a pressure vessel (autoclave), to cure the epoxy resin (Soutis, 2005).

Composite laminates are expensive to manufacture due to the complexity of the layup process. However, it is possible to produce lighter and higher strength structural elements that can be directly used in mission-critical applications. Similar to any other types of materials, defects in composite structures might occur during production or in service. The anisotropic and non-homogeneous nature of composites combined with their layered structure makes the detection and characterization of defects difficult.

Non-destructive evaluation (NDE) methods can be used for damage characterization of composite structures. Deng et al. (2004) developed a graphical user interface for visualization of NDE
data superimposed on composite structures. A recent study by Smith et al. (2015) introduced a method to incorporate defects into 3D CAD models of composite laminates using data from micro-CT X-Ray and ultrasound. Bliznakova et al. (2014) also developed a framework for generating computational models of small CFRP composite parts for use with NDE X-Ray imaging. However, these researches focused on simplified geometries and were not designed for automation.

Most layer-by-layer models are very simple geometries for mechanistic structural analyses. Previous studies on modeling of composite laminates mostly focused on simulation of composite structural behavior using FEA in the sense of mathematical representations of microstructures for "virtual testing" (Davies and Ankersen, 2008; Pineda et al., 2009; Panchal et al., 2013; D’Mello et al., 2016). Recently, a method to describe isogeometric analysis of shell models of composite laminae with curved shapes using NURBS representations in order to predict the failure mechanisms has been studied by Guo and Ruess (2015). These studies mostly focused on stress development on crack tips.

Delamination is a common type of defect that can reduce the mechanical strength and stability of the layered composite and cause catastrophic failures at unexpectedly low loads (Cantwell and Morton, 1991). Delamination is the separation of interior layers of a composite laminate caused by manufacturing defect or impact damage (Orifici et al., 2008) and it usually grows under shear stress (Richardson and Wisheart, 1996). It is possible to incorporate delaminations into FE models by first meshing the geometry and then duplicating the nodes that lie on the delamination without any linking between them, which will allow separation. However, using this approach the CAD model needs to be re-meshed after every step of delamination growth, which can be computationally intractable. In this paper, we include the delamination as part of the CAD model by splitting the adjacent faces of the layers into delaminated and intact regions. In addition, we model the delamination using cohesive surfaces that allow for the delamination to grow without remeshing each step.
2.4 Framework for Modeling 3D Composite Structures

Our composite modeling framework provides an automated application programming interface (API) that is capable of creating customized 3D CAD models representing the layered structure of a composite laminate and apply user-defined boundary conditions for structural analysis using a finite element analysis (FEA) software. The framework can be used to process defect data, which is obtained using non-destructive evaluation (NDE) of FRP composite samples and incorporate them into 3D models of the composite laminate. Finally, the framework can be used to preprocess the model for structural analysis using FEA by applying the appropriate cohesive and contact boundary conditions between the layers of the laminate.
2.4.1 Components of the Framework

The automated framework consists of three major components, the CAD Model Builder, the Finite Element (FE) Model Builder, and the Model Builder API. An overview of the framework showing the details of these three major components is shown in Figure 2.2. The CAD Model Builder implements the set of operations for constructing the layers, imprinting delaminations, and identifying adjacent faces for the FEA software. It uses a commercial solid modeling kernel, ACIS, to generate the laminae and incorporate the defects. Our API abstracts the solid modeling kernel functions from the user and provides functionality that is focused on generation of composite laminates and for incorporation of defects. The FE Model Builder implements the set of operations required to construct a finite element model using the ABAQUS FEA package. The ABAQUS programming interface is used to apply the appropriate boundary conditions, external loading, material properties, and meshing. The combined Model Builder API expresses high-level lamina operations, such as creating and bonding layers, in terms of CAD model builder and FE model builder operations. The code generator allows CAD and FE operations, written as if to be performed in parallel, to execute in the intricate sequence required by FE software.

2.4.2 Integrating CAD and FE Analysis

Integrating CAD and FE analysis involves automatically applying all the correct boundary conditions, which can be very challenging in practice. The intended boundary conditions need to be stored during the model construction phase, and then assigned later once the model is complete. An obvious approach to storing boundary conditions is to define a data structure that describes the desired boundary conditions in detail, which can be populated during model creation. The issue with such an approach is that it is inflexible; the data structure must anticipate every possible boundary condition that may be desired later (the same problem applies to meshing as well). As a consequence, adding new boundary condition types to the framework will break backward compatibility; and will render any model generated using the previous version of the framework obsolete.
In a previous effort, Holland et al. (2016) used anonymous functions as a vehicle to pass instructions for assigning boundary conditions from the model creation phase to the boundary creation phase. Anonymous functions are generated on the fly during the execution of the code. The anonymous function would be automatically defined when the model was created with code to identify the correct faces and apply the correct boundary condition. It would be stored with the model, and then executed later during boundary condition phase to assign the correct boundary conditions. The major drawback of this technique was that the model creation phase and boundary condition phase is executed in the same context by the same interpreter making separate phases tightly bounded to each other.

In building this modeling framework for composite laminates, we similarly need to store the boundary conditions between model creation and boundary condition assignment phases. However, we did not want to combine the phases so closely under the same execution environment as would be required to use anonymous functions from the model creation phase in the boundary condition assignment phase. We wanted to keep the solid modeling operations using the ACIS solid modeling
kernel separate from the finite element operations executed under ABAQUS’s scripting interface. Hence, we developed an alternative solution that achieves similar results, without using anonymous functions.

We created proxy objects and classes for the FE Model Builder that store the sequence of operations that is performed rather than executing the operations immediately. Any objects that are created or returned are proxy objects that represent the result of the operation that has not yet been performed. Operations on the proxy objects get stored as well. Eventually, after the CAD Model Builder is complete, the sequence of FE Model Builder operations can be exported as generated code and executed within ABAQUS’s scripting interface. This code loads the solid model into ABAQUS, applies the specified boundary conditions, generates a suitable mesh, and performs the structural analysis. The validity of using ABAQUS with composites has been shown previously in different studies Mustapha et al. (2012); Zhang et al. (2008).

In this way, CAD Model Builder and FE Model Builder operations can be intermingled within the modeling framework. The CAD Model Builder operations are executed immediately, whereas the FE Model Builder operations are separated into different queues and executed in the order required by the ABAQUS FEA package. An illustration of the usage of the framework for creating a 2-layer composite structure is shown in Figure 2.3. As a result, the boundary conditions on the finite element model are applied correctly without requiring a complicated data structure.

2.4.3 CAD Model of the Composite Laminate

We generate the 3D CAD model of the layered composite structure using the CAD Model Builder. This component follows the operations performed during manufacturing of the composites. It creates the composite structure layer-by-layer; and similar to a real production, requires a mold and a thickness to construct each layer.

The initial mold of the CAD Model Builder is a parametric NURBS surface. NURBS is the de facto industry standard for representing curves and surfaces using control points and basis functions, which are controlled by knot vectors Piegl and Tiller (2012); Stroud and Nagy (2011).
Figure 2.4 Illustration of creating Layer elements from (a) a parametric surface (i.e. a NURBS surface) and, (b) the existing layers by using their offset or original faces as the mold.

We use a custom-built NURBS library to manage the NURBS objects in the CAD Model Builder component. This NURBS library requires the degrees, knot vectors, and control point grid to calculate the initial mold surface. In addition, this NURBS surface can also be automatically extracted from an existing shell model. This initial mold surface is called *original face* in the framework to represent the mold in the FRP composite production process. The solid modeling kernel takes this NURBS surface and converts it to a sheet body, which is the CAD representation of a body with no thickness. The framework then uses the thickening operation to generate the final closed solid body, as illustrated in Figure 2.4(a).
We can make use of either sweeping or thickening to create the solid model of the composite lamina from the mold surface. The mold surface can be swept along a user-defined path (Stroud, 2006), to create a closed solid model of the lamina. Thickening (Stroud, 2006) differs from the sweeping operation in that it generates an offset of the initial sheet body in the given thickness direction, and then generates the side faces between the original and the offset surfaces to create closed 3D solid body. The thickening operation generates a more realistic representation of the composite lamina, since it creates a solid body with uniform thickness.

Succeeding layers can be generated using either the original mold surface or the offset surface from the previous layer as mold. The original face and the offset face of the initial layer are used to generate layers in the positive or negative directions, respectively. The generation of the next layer using an existing layer is illustrated in Figure 2.4(b). Based on the user input for the new layer direction, the framework determines the correct face and calculates the new mold for the chosen direction. The framework then calls the solid modeling kernel to perform the thickening operation and generate the new layer as a closed solid body from the calculated mold. However, the solid modeling kernel does not store any meta-data regarding the composite structure in the solid body object. Therefore, every time a new layer is generated from an existing layer, our framework traverses through all the faces of the existing layer to find the appropriate original and offset faces for the chosen layer generation direction.

2.4.4 Incorporating Stiffeners into the CAD Model

The CAD Model Builder has the ability to incorporate stiffeners into the 3D representation of the composite laminates. Stiffeners are mainly used to increase the bending rigidity of structural materials. Composite stiffeners have special designs to accommodate the layered structure. A commonly used stiffener, "hat" stiffener, has a trapezoidal cross-section over which additional layers are bonded. Figure 2.5 illustrates the process of incorporating a hat stiffener to the composite laminate and generation of layers over the stiffened structure.
Figure 2.5  Illustration of the process of adding a hat stiffener onto the existing composite laminate. After placing the hat stiffener mold (orange) on top of the existing layered structure, a new mold (yellow) is generated for the subsequent layers which will be placed on top of the stiffened structure. For this illustration, new layers generated in positive (upward) direction use this updated mold shape, whereas the new layers generated in negative (downward) direction would use the planar-shaped mold.

The hat stiffener cross-section is provided by the user as input. The framework then creates a closed wire body from this input and sweeps the newly generated wire body parallel to the offset face of the topmost layer. This creates the solid model of the hat stiffener on top of the composite laminate. Following the actual production process of composites, the framework bonds the hat stiffener on top of the composite laminate by imprinting the adjacent faces of the hat stiffener and the offset faces of the top layer. This imprinting operation splits and generates new faces on the offset surface of the topmost layer. To generate new layers above the stiffener, the framework creates a new mold using the shape of the combined topmost faces of the layer and the hat stiffener. The mold is created by stitching the copies of the free faces of the top layer and the copies of the faces of the hat stiffener in the offset direction and creating a sheet body. The framework then uses the new stiffener-shaped mold for generating new layers on top of the hat stiffener.
Figure 2.6  Illustration of imprinting delamination outline to the faces. User can choose any face to imprint delamination. Faces are defined by offset and original faces.

In production of FRP composites, the stiffener shape may be removed after the composite structure is manufactured. The CAD Model Builder can replicate the stiffener removal process by removing the solid body representing the stiffener from the final CAD output and removing the corresponding adjacent faces used for applying boundary conditions.

2.4.5  Incorporating Defects into the CAD Model

In the actual production process of composites, each layer is placed down on the existing composite laminate or mold and then glued. This step of the FRP composite production process is replicated by the framework. After the layers are generated, using the bonding methods of the framework, the adjacent faces of the layers are imprinted Stroud (2006); Stroud and Nagy (2011) to each other. This doesn’t have any effect on the layers that are not modified to incorporate defects. However, this imprinting step is crucial for layer faces having defects, such as delaminations.

The process of incorporating delaminations to the CAD model of the laminates involves an input of the 3D coordinates of the delamination outline and the layers between which the defect will be incorporated. The framework converts the 3D coordinates into a closed wire, represented using a b-spline curve. The framework then finds the offset face of the first chosen layer and the original face of the second chosen layer and imprints the projected delamination shape onto these faces. The projection is an important and required step for incorporating the delamination shape
between the chosen layers as the layers might have been generated from a non-planar shaped mold and the wire should conform to the actual layer shape. The steps for incorporating a delamination into the composite laminate model is illustrated in Figure 2.6. Imprinting a closed delamination outline onto a face splits it into two faces, representing the delaminated and the intact regions, respectively.

Our framework also has the ability to replicate fiber breakage within a composite layer. Fiber breakage is the unexpected breaking of the reinforcement fibers in the composite during production or in-service conditions, reducing the mechanical strength and durability of the composite. The framework can introduce fiber breakage defects into a layer by splitting it into two solid models using the curved surface of the fiber-breakage. This creates two separate solid bodies in the layer structure (Figure 2.7), resulting in the generation of multiple faces on the offset and original sides of the layer. These newly generated faces are imprinted on the adjacent layer faces in the original and offset directions, respectively, to maintain the consistency in applying the boundary conditions during the finite element analysis of the final composite structure. If the fiber breakage does not extend to the boundaries of the layer, we extend the fiber breakage surface to the closest face of the layer on both ends. We then apply tie boundary conditions to these extension faces that treats this region as intact in the structural analysis.
Figure 2.8 If the fiber breakage does not extend to the end of the composite region, the surface is extended to the closest edge. Appropriate tie boundary condition is then applied to the intact (pink) region.

2.4.6 Processing NDE Data to Identify Delaminations

The ultrasonic NDE data processing techniques described in Bingol et al. (2017) is used to obtain the 3D coordinates of the delamination outline from raw ultrasonic testing data. We briefly outline the techniques in this paper for completeness. Front-wall correction is first applied to the raw ultrasonic testing data to correct for variations in the location of the top surface. This then helps in identifying the location of the delamination between the corresponding plies of the composite laminate. Once the delamination shape is located, it is extracted and cleaned using erosion/dilation methods commonly used in binary segmentation. After the image cleaning step, the outline of the resultant delamination shape is extracted using edge detection. The outline is then input to the CAD Model Builder as a set of 3D coordinates, and can directly be used by the automated framework for defect incorporation.

2.4.7 Structural Analysis Using the Cohesive Model

The framework uses a cohesive model (Alfano and Crisfield, 2001) to simulate the bonding between the different layers of the laminated structure. An example of the use of cohesive model
in the presence of delaminations is shown in Figure 2.9. In the cohesive model, the bonded regions are modeled using a force-displacement relation between the laminae of the composite that can represent debonding and enables simulation of delamination growth.

A contact boundary condition is applied to the delaminated region, which prevents the inter-penetration of the surfaces of the lamina in the delaminated region while allowing for the lamina to separate freely. To assist convergence of the 3D structural models, a region of free boundary without any cohesive or contact model is used denoted as the No Model Zone. The border of the cohesive zone is meshed using appropriately small elements to allow for delamination growth in dynamic simulations.

The No Model Zone is concretely represented in the CAD model by offsetting the delamination shape inward. Then this offset shape is imprinted on the surfaces between the laminae to generate the Contact and the No Model Zone. The FE Model Builder is then used to apply contact boundary conditions to the region bounded by the innermost outline and cohesive boundary conditions to the region outside the outermost outline. An example of this operation is shown in Figure 2.11(d).

Each face in the composite model is uniquely identified using a point and normal vector pair, which can be used to apply the appropriate boundary conditions to the face. However, these point and normal vector pairs need to be initialized when each face of the composite model gets generated either through the layer generation process or through any process that splits an existing face. Another challenge in finding these points and normal vectors is that these points have to be inside the
surfaces (not on the boundaries or vertices) in order to unambiguously identify them while applying the boundary conditions. We have developed a geometric algorithm that finds a point and a vector automatically in all trimmed surfaces generated after the imprinting and splitting operations, such as incorporation of delaminations. The framework stores the layer, surface, and mold data in predefined structures, namely Layer, LayerBody, LayerSurface and LayerMold classes. These classes allow the framework to keep track of all generated layers, surfaces, and molds in addition to their relations between each other. The algorithm determines points and vectors belonging to each surface in each layer and stores the evaluated point-vector information along with the geometrical and topological data created by the solid modeling kernel.

The geometric algorithm for finding the point-normal pair inside each surface utilizes the bounding box of the trimmed surface. A guess point is initialized as the lowest point of the diagonal of the bounding box. The guess point is then moved along the diagonal until a point belonging to the trimmed surface is found. However, if the diagonal does not intersect with the trimmed surface, we fall back to two other algorithms to find the surface point. One of the fallback algorithms picks a random point on the trim curve of the surface in 3D space and moves a small distance along the normal direction to the trim curve to find a point belonging to the surface. However, in some cases, the normal evaluation fails (for example, if the edges are stored implicitly, such as a line equation instead of a parametric curve). In such cases, we use the second fallback algorithm that picks a random parametric point on the edge of the trim curve in the parametric space and translates this point along a random direction. The edge point is then repeatedly translated along different random directions until a point inside the trimmed surface is found. In practice, we found that these algorithms are sufficient to find a point normal pair that lies inside the trimmed region for each trimmed surface. After finding the points corresponding to the all trimmed surfaces, the framework uses solid modeling kernel to find the normal vector of the surface at the point.

After finding the point-vector pair that identify the delaminated and non-delaminated trimmed surfaces, the CAD Model Builder finds surfaces adjacent to each other to aid in the bonding step during the finite element model generation. The bonding step is analogous to bonding the composite
laminates in composites manufacturing, in which all layers are glued to each other. The adjacent surface information is evaluated using the data stored in the Layer classes. To identify that two surfaces are geometrically adjacent to each other, the algorithm uses the stored point-normal pairs of each surface. If these points lie inside the surfaces and the vectors are anti-parallel (up to a predefined tolerance value), the surfaces are marked adjacent. CAD Model Builder component takes this information to build up a list containing point-vector couples of all adjacent faces and passes this to the FE Model Builder.

2.4.8 Model Builder API

The combined Model Builder API to the framework handles the user-input and distributes the corresponding API calls to either the CAD Model Builder or the FE Model Builder in the required order. In addition, to facilitate the dynamic generation of a finite element processing script based on the output of the CAD Model Builder, a code storage and generation scheme is adopted. The code generation must be able to queue up FE preprocessing commands and generate the script which will be input to ABAQUS to run the analysis. The code storage capability allows the user to add commands to different storage categories (model initialization, internal boundary conditions, assembly commands, external boundary conditions, and meshing commands), which are executed in the order required by ABAQUS.

The model initialization instructions initialize and handle any importing of the CAD model. The assembly commands instantiates the individual laminae as ABAQUS parts based on the geometry imported from the CAD Model Builder. The internal boundary conditions specify the regions that need to be bonded using continuity, bonded using cohesion, or assigned a contact interaction property. The external boundary conditions apply additional force or displacement boundary conditions. The meshing instructions seed and instantiate a suitable mesh with appropriate meshing parameters. For example, the laminae that contain a delamination can be meshed using free tetrahedral elements, while laminae free of delaminations can be meshed with swept quadrilateral elements. Finally, the analysis is submitted to the ABAQUS FE solvers.
2.5 Application of the Framework to Model Composites

In this section, we show the application of our framework to create several examples of CAD models of multi-layered composite structures, incorporating delaminations or fiber breakage, and generating a script that can be used to perform static structural analysis on the resulting composite model.

2.5.1 Sample Composite with Impact Damage

We used a CFRP sample that was impact damaged as our initial test sample to obtain a delamination outline. The sample was measured using bi-directional ultrasonic testing. The shape and location of the region of interest with respect to the laminate structure is used to correctly register the scan. Figure 2.10(a) shows the composite sample used and the $2 \times 2$ in$^2$ scan region.
2.5.2 3D Models of Composites

The CAD Model Builder is capable of generating a layer from any mold surface, represented as NURBS, with a user-defined thickness value that match the desired composite structure. Figure 2.11(a) displays the 8-layered 3D composite laminate generated from a sample 3rd degree curved NURBS surface and Figure 2.11(b) displays the 8-layered 3D composite laminate generated from a planar surface. The CAD Model Builder generates every layer as a separate body and does not perform any boolean unite operation, since this would remove the internal surfaces. The layers are glued to each other by the FE Model Builder component of the automated framework as instructed by the Model Builder API.
2.5.3 Incorporating Stiffener to the 3D Composite Model

The CAD Model Builder is capable of generating layers using the mold shape generated by the combination of the planar layer and the trapezoidal hat stiffener. As described in the Section 2.4.3, the stiffener element is generated from a user-defined trapezoidal shape that is swept horizontally along the layer to generate a trapezoidal prism, representing a hat stiffener placed on top of the composite structure. Figure 2.11(c) displays an example of such a composite structure. After generating the first 4 planar layers, a hat stiffener is generated on top of the 4th layer and the succeeding 4 layers are generated with the new mold shape that includes the stiffener. The stiffener is either bonded to the composite structure or removed based on user requirement.

Generating the correct geometry of the stiffened layers requires performing the thickening operation instead of sweeping. Figure 2.12 compares the thickness differences between swept and thickened offset surfaces of the stiffened layers. The sweeping operation leads to thickness variation along the inclined faces.
Figure 2.13  5-layered planar composite laminates illustrating fiber breakage of different shapes in the 3\textsuperscript{rd} layer. The user inputs a list of coordinates that defines the shape of the fiber breakage and the specific layer. The framework then splits the corresponding layer to emulate the fiber breakage.

2.5.4 Incorporating Delaminations to the 3D Composite Model

The CAD Model Builder is capable of incorporating delaminations extracted from ultrasonic scans between the layers of the composite structure. Figure 2.11(d) shows the wireframe model of the 3D 8-layered planar composite laminate with the delamination incorporated between the 2\textsuperscript{nd} and 3\textsuperscript{rd} layers. In order to generate the cohesive, contact, and no-model zones (Figure 2.9), the CAD Model Builder offsets the delamination outline inwards, projects both outlines on the layers, and imprints them. In the case of Figure 2.11(d), the imprinting operation generates 2 additional faces on the designated layer face representing the delaminated area in the composite laminate. The inner face is assigned contact boundary condition and the outer face is assigned cohesive boundary condition. The layers are otherwise bonded with continuity (tie) boundary conditions.
2.5.5 Incorporating Fiber Breakage

Figure 2.13 shows two wireframe models of a 5-layered planar composite laminates with fiber breakage defects. After creating the specific layer in which the fiber breakage needs to be incorporated, the user inputs a list of coordinates that form a curved path of the fiber breakage. This curved path is converted into a wire within the automated framework and is projected to the chosen layer to generate a splitting surface as illustrated in Figure 2.7. The splitting operation uses this surface to split the layer and generates 2 different closed solid bodies, adjacent to each other. These solid bodies are processed by the framework to generate correct LayerBody and LayerSurface objects for further analysis in the FE software.

2.5.6 Finite Element Analysis of the 3D Composite Model

The finite element analysis can be configured using user-defined material properties, external boundary conditions, and meshing parameters. The end result of the Model Builder API is a CAD file and a script that can be used by the FE analysis software to set up the structural simulation of the resulting CAD model from the CAD Model Builder.
Figure 2.15  Force (N) vs Displacement (mm) graphs comparing delaminated and non-delaminated 8-layered composite structures at different locations.

Figure 2.14 shows the results of the structural FE analysis of a 8-layered composite structure with a single edge delamination. A static force is applied in the downward direction to the bottom face of the of the 8-layered 3D composite structure for 40 steps. The applied force is incremented by 3.125 N in each step. The material properties used in the analysis are: Young’s modulus $E_1 = 1.415 \times 10^5$ MPa, $E_2 = 8.5 \times 10^3$ MPa, and $E_3 = 8.5 \times 10^3$ MPa; Poisson’s Ratio $\nu = 0.33$ in all directions; Shear Modulus $G_{12} = G_{13} = 5.02 \times 10^3$ MPa, $G_{23} = 2.35 \times 10^3$ MPa; Fiber angles in the stacking sequence $[0, -45, 45, 90, 90, 45, -45, 0]$. Figure 2.14(a) shows the delaminated model; the delaminated region is marked with a red outline. Figure 2.14(b) displays the loading setup for the static bending analysis showing the direction of the force applied with the small arrows visible on the model and the fixed boundary conditions. Figure 2.14(c) shows the deformed model after the last loading step of the FE analysis. The delaminated edge after the last step of the FE analysis shows the separation of the layers due to the effect of shear on the delaminated region (Figure 2.14(d)). It is also possible to observe the non-uniformity of the displacement field due to the delaminations present in the structure.
Figure 2.16  Force vs. Displacement curve comparing the effect of fiber breakage on the same composite laminate illustrated in Figure 2.13 with the stacking sequence of \([0 -45 0 45 0]\). The "breakage" label corresponds to the model with the fiber breakage and "intact" label corresponds to the same model without fiber breakage. As expected, the composite laminate with the fiber breakage shows slightly higher deflection than the one without fiber breakage. The inset graph is the zoomed version that shows the difference between the curves.

We expect to see higher displacement values in delaminated composite structure. Figure 2.15 compares the displacement values of two composite models with (red) and without (black) delamination. In Figure 2.15(a) the nodes are chosen from the center location of the free end to analyze the effect of delamination on the composite structures, and in Figure 2.15(b) the nodes are chosen from a position just below the separated layers (also shown in Figure 2.14(d)). As expected, higher displacement values were observed on the delaminated composite structure compared to the non-delaminated one, showing the capability of our framework in simulating composite structures with defects.

Figure 2.16 illustrates the effect of fiber breakage on the composite laminates. The 5-layered composite laminate model illustrated in Figure 2.13 with and without the fiber breakage is used for the comparison of the deflection at the free end. The fixed ends are chosen to allow maximum possible deflection on the laminate with fiber breakage. Stacking sequence used for both laminate
models is \([0 -45 0 45 0]\). As expected, the laminate with the fiber breakage deflects slightly more than the one without fiber breakage. At the maximum load of 125 N, the difference in the displacement between the laminates is 0.023 mm.

Figure 2.18 illustrates another 8-layered composite laminate with an edge delamination, stiffener, and a delamination under the stiffener. A trapezoidal hat stiffener element is placed on top of the 4\(^{th}\) layer to set the shape of the mold used to generate the 5\(^{th}\) layer. The stiffener element is later removed. One of the big challenges during the development of this automated framework is the introduction of a full-sized delamination under the stiffener which requires imprinting on the faces under the stiffener, but only the touching faces have boundary conditions. Figure 2.18 shows the analysis results of a stiffened composite laminate with a delamination between layers 4 and 5 along with an separate edge delamination between layers 1 and 2. As with the previous example, analysis of this model shows layer separation in the delaminated regions.

The framework is capable of generating composite laminates with different stacking sequence configurations. This allows users to observe the interaction between the different stacking sequences, the structural stiffening elements, and delamination defects. As an example of measuring the effect of the stacking sequence on the displacement field, a composite laminate with the same material properties and boundary conditions as the previous example but with a different stacking sequence of \([0/-90/90/0/0/90/-90/0]\) is generated using the automated framework and the results are illustrated in Figure 2.17. The effect of the delaminations on the uniformity of the displacement field can be observed since a symmetric stacking sequence is used during the generation of the composite laminate. In addition to the effect of the stacking sequence, it can also be observed that the displacement field becomes non-uniform at both ends of the delaminated region.

Figure 2.19 illustrates the capabilities of the automated framework in generating curved laminates with delaminations. We introduced a delamination between 2\(^{th}\) and 3\(^{rd}\) layers of a 8-layered composite laminate generated from a curved mold which is also illustrated as a 3D model in Figure 2.11(a). The stacking sequence applied during the structural analysis of the composite laminate is \([0 -45 45 90 90 45 -45 0]\) and all other finite element analysis properties are the same with the
previous analyses. Similar to the previous examples discussed in this paper, we observed a similar displacement field as well as a clear layer separation in the delaminated region. The separation inside the curved composite laminate can be observed at $x = 7.60$ via the View Cut Manager property of ABAQUS visualization interface.

2.6 Extensions and Future Work

2.6.1 Shell Solid Coupling

We have developed some preliminary methods in our framework to couple the regional layered composite model with the reduced dimensional complete shell structure and performed coupled structural analysis. Figure 2.20 illustrates our preliminary work on shell-solid coupling of the planar composite laminates. A shell model is a single sheet body representing the shape of the base composite structure. The framework loads the shell model and cuts the arbitrary region representing the damaged region on the shell model. The edges of the cutout region is rounded to prevent stress concentration on the corners during structural analysis. As previously discussed in the previous examples, the framework builds the multi-layered composite laminate using the cutout region as the mold and sets the appropriate boundary conditions between the layers.

In order to couple the shell model with the layered structure, the framework also sets the boundary conditions between the layers and the shell model. To set these boundary conditions, the
Figure 2.18  8-layered delaminated laminate generated by the automated framework with a trapezoidal stiffener element. (a) shows location of the delaminations on the edge between the 1st and 2nd layers, and under the stiffener in the middle of 4th and 5th layers with the markers representing the fixed boundary conditions for each layer, (b) shows the boundary conditions and the loading setup, (c) shows the edge delamination on the deformed composite laminate after static bending, and (d) displays a cut-view of the delaminated section between 4th and 5th layers (position x = 10.99, deformation scale factor = 20, showing only feature edges).

framework first identifies the inner loop defining the cutout region and then, it finds the middle points of each edge in the inner loop and a corresponding surface normal vectors evaluated on the shell model. Then, the framework translates the evaluated middle points along the normal vector by half of the thickness value and then finds the closest points corresponding to the side faces of the composite laminate. This process is repeated for each layer generated from the cutout region of the shell model.

Figure 2.20(a) illustrates the fixed boundary conditions on the shell model and the loading setup for the static bending. The load direction is on -z direction and all the analysis parameters are the same used on the previous examples. The framework also sets tie boundary conditions (not shown on the figure) between the shell model and the assembled inner composite laminate. Figure 2.20(b) displays displacement field after a static bending analysis. We used a course mesh of size 3.0 mm for the shell structure to speed up the analysis. Figure 2.20(c) and Figure 2.20(d) show displacement and stress fields of the inner 5-layered composite laminate, respectively. As expected, the largest displacement is observed at the middle region of the inner composite laminate and the stress field
Figure 2.19  Bending test of a 8-layered delaminated curved composite laminate generated by the automated framework. (a) shows the location of the delamination inside the laminate model, (b) shows the fixed edges and the loading direction, (c) shows the displacement field ($U_{\text{magnitude}}$), and (d) displays a cut-view of the delaminated section between 2\textsuperscript{nd} and 3\textsuperscript{rd} layers (position $x = -7.60$, deformation scale factor = 5, showing only feature edges).

is uniform with minimal stress concentration at the corners of the inner composite laminate. This is still preliminary work and more validation of the coupling needs to be performed to perfect the method.

2.6.2 Framework Extension

The automated framework can be extended in two different ways: (1) adding new features, such as a new composite feature (e.g. stiffener or a damage model) and (2) extending it to different modeling platforms. Since the automated framework is designed using Separation of Concerns principle, all the components of the automated framework can be replaced with alternative implementations. For instance, the Layer structure only acts as a data container and the modeling API only deals with the solid modeling kernel and the FEA package. Interaction of these components are handled using the abstract base classes which are implemented with no dependencies to the external software packages.

Adding a new feature to the framework can be performed by implementing the model generation operations in the modeling API. For example, if the user wants to implement a T-stiffener, the only requirement is creating the function that generates the CAD representation of the molds that would
Figure 2.20  Shell-solid coupling of the 5-layered planar composite laminate region with the shell model. (a) shows the boundary conditions and loading setup of the shell model and the 5-layered composite laminate constructed by cutting an arbitrary rectangular region on the shell model, (b) shows the displacement field ($U_{\text{magnitude}}$) of the shell model meshed coarsely (3.0 mm), (c) shows the displacement field of the constructed 5-layered planar composite laminate, and (d) displays the stress field of the same region.

allow the automated framework to generate a T-stiffened layer. The Layer structure is capable of storing multiple solid bodies and offsetting and imprinting operations are designed to be shape-agnostic. Therefore, the user does not need to change any of the internal functionality related to layer generation or damage incorporation.

The automated framework is also designed to allow integration with different solid modeling kernels and FEA software packages. The design perspective of the CAD Model Builder is hiding the complexity of the solid modeling kernel APIs, directly providing users a simplified interface to generate composite laminates with or without artifacts, such as delaminations or stiffeners. To attain this perspective, the CAD Model Builder utilizes the best practices to maintain low coupling and high cohesion between the its components while exporting only the necessary functionality to the Model Builder API. The subcomponents of the CAD Model Builder handling the geometric representations and operations are implemented using generic programming techniques to allow different implementation scenarios (or allow different configurations). The main modeling component of the CAD Model Builder is generalized up to the most possible extent; the common operations, such as file reading, face adjacency list generation, interaction with the sub components, are implemented
in the base classes and the functionality which requires interaction with the solid modeling kernel is implemented in the subclasses.

The software design perspective of the FE Model Builder is creating a thin layer on top of the finite element software package API, allowing the existing finite element software users to directly apply their knowledge without the need to learn another API. The flexibility of the FE Model Builder comes from its integration with the code generation component. Since the framework is mainly designed to be used only to generate the code that could be executable by the FEA software package, changing the configuration of the FE Model Builder would allow users to use any finite element software package which allows a scripting interface.

The most important challenge we foresee in extending the framework is the difference between the modeling algorithms used by the solid modeling kernels. Although most solid modeling kernels provide similar APIs, the operational differences between them disallow users to switch them on the fly or making them inter-operate with small changes. Even though the design of the software is extensible, a huge amount of testing might be required to extend the framework to a different solid modeling kernel. The same also applies to the finite element software package.

2.6.3 Limitations

There are some challenges and limitations to the current implementation of the framework. For example, we have not implemented a more complex composite manufacturing processes that include features such as T-stiffeners, resin buildup, fillers, layer drop-off, etc. These features can be specifically addressed as extensions to the framework as described in the previous section. These will include adding a modeling component to the CAD Model Builder and setting up the appropriate boundary conditions using the FE Model Builder. For example, the layer drop-off can be modeled similar to the hat-stiffener with one layer ending in the middle of an existing layer and generating a new mold shape that forms the top surface of all bonded layers.

We have not independently validated the results of the structural analysis. We only use well validated FE models available in ABAQUS and the results obtained were exactly the same as those
that would have been obtained on manually setting up the models in ABAQUS. The composites models in ABAQUS have been well validated by previous studies Mustapha et al. (2012); Zhang et al. (2008). In addition, the quantitative displacement results in the models are on the same order as those that would be expected from theoretical analysis using classical composite plate theory.

2.6.4 Future Work

Future work on the automated and integrated framework will focus on incorporating different geometries of composite stiffeners, such as T-stiffeners and grid stiffeners. We will also focus on implementing stiffeners to the curved composite structures. Furthermore, we will work on more complicated shape examples of performing multi-scale analysis of large composite structures, such as curved shell models and arbitrary cutout regions. Such models can then be used to predict the influence of defects in a relatively small region on the residual strength of the large and complex-shaped composite structures.

In addition to new geometric features, we will also work on extending the framework to different solid modeling kernels and FEA packages. Although such an extension would require extensive testing and will be heavily dependent on the FEA methods implemented in the corresponding software, it will allow users more flexibility on choosing the CAD and FE software. In addition, it will also help in the wider adoption of the framework by researchers.

2.7 Conclusions

We have presented an automated framework for building composite laminates with defects for structural analysis. Our framework can incorporate complex structures such as stiffeners into a layer-by-layer CAD model of composites. The framework automates the model setup process, thereby removing tedious operations needed for setting up the boundary conditions in the model before analysis can be performed. In addition, the framework can incorporate complicated delamination shapes obtained from ultrasonic testing of the composite between the corresponding layers of the laminate. The framework then automates the process by generating a CAD model
and a corresponding script that can correctly set up the boundary conditions to perform structural analysis. In addition, we have developed preliminary methods to couple the detailed layer-by-layer model with defects with a lower dimension shell model of the entire structure. Incorporating such high-fidelity damage models can improve the accuracy of residual strength predictions and can lead to better decisions regarding repair of damaged composite structures.

We will be releasing our framework as a free and open-source project publicly on GitHub. Open-sourcing would allow wider adoption of the framework and allow users to integrate it with their own composites modeling pipeline. In addition, they can extend the framework to their specific requirements with various solid modeling kernels and finite element software packages.

**Acknowledgements**

This work is supported by National Aeronautics and Space Administration (NASA) under contract number NNL15AA12C through Center for Non-Destructive Evaluation (CNDE) at Iowa State University and, is a part of NASA Advanced Composites Project. We would like to thank Dr. William P. Winfree and Dr. Cheryl A. Rose from NASA for their valuable input and support for this project. We would also like to thank Dr. Ronald Roberts of CNDE at Iowa State University for the UT scans and the initial data.

2.8 References


CHAPTER 3. NURBS-PYTHON: AN OPEN-SOURCE
OBJECT-ORIENTED NURBS MODELING FRAMEWORK IN PYTHON

A paper published in SoftwareX
Onur Rauf Bingol and Adarsh Krishnamurthy

3.1 Abstract

We introduce NURBS-Python, an object-oriented, open-source, pure python NURBS evaluation library with no external dependencies. The library is capable of evaluating single or multiple NURBS curves and surfaces, provides a customizable visualization interface, and enables importing and exporting data using popular CAD file formats. The library and the implemented algorithms are designed to be portable and extensible via their abstract base interfaces. The design principles used in NURBS-Python allows users to access, use, and extend the library without any tedious software compilation steps or licensing concerns.

3.2 Introduction

Non-Uniform Rational B-Splines (NURBS) are accepted as the industry standard for the representation of geometry in mechanical computer-aided design (CAD) systems. In addition, they are used in many other fields such as robotics and self-driving cars that require dealing with geometric elements for trajectory generation and smoothing. Traditionally, NURBS algorithms are developed using compiled languages, such as C (SINTEF, 2018) or C++ (Eason Kang, 2018; Robert McNeel & Associates, 2018) in order to achieve better performance over interpreted languages. However, running simple geometrical queries using compiled libraries require tedious and complicated setup that depends primarily on the operating system and computer architecture. A NURBS library written
using an interpretable language such as Python, can considerably reduce the overhead of compiling the library, and lead to widespread usage of the NURBS algorithms in different applications.

In order to develop a widely accessible NURBS library, we have implemented it using Python. Python (Rossum, 1995) is an interpreted high-level programming language that is widely used by non-programmers and scientists. By design, python code can work on most modern platforms. The reference implementation, CPython provides a well-designed C programming interface for interacting with different libraries. As a result, modern libraries, such as Theano, TensorFlow, Cognitive Toolkit (CNTK), provide user interfaces implemented in Python to reduce the programming interface learning effort and development time, while using Python’s C programming interface for accessing low level libraries, such as nVidia’s CUDA and cuDNN.

This paper describes the NURBS-Python package and its programming interface. NURBS-Python is a computational geometry library specifically designed for evaluating rational and non-rational B-spline curves and surfaces. NURBS-Python is an open-source library designed to have minimum external dependencies. The library provides a fully extensible object-oriented data structure, evaluation, and visualization capabilities implemented using the Python programming language. The library can be utilized using a direct object-oriented application programming interface (API). It does not contain elements that could restrain its flexibility, such as a graphical user interface (GUI) or a domain-specific language implementation. Instead, it allows users implement any graphical user interface using its abstract base classes. The optional visualization component included in the package implements this abstract base and can be used for plotting the curves and surfaces. The abstract base, data API, and evaluation capabilities are self-contained with no external module dependencies. On the other hand, the visualization components implement most commonly used plotting and visualization libraries, such as Matplotlib and Plotly.

This paper presents the different components of the NURBS-Python library. The main contributions of this paper include:

- An object-oriented and self-contained NURBS framework providing easy-to-use data structures and extensible algorithms.
• A pure Python NURBS computational library with no extra dependencies or compilation requirements.
• Utilizing existing plotting libraries to visualize NURBS curves and surfaces.
• A free and open-source extensible framework without any licensing concerns.

This paper is arranged as follows. In Section 3.3, we highlight our design considerations and compare NURBS-Python with existing packages and in Section 3.4 we outline the NURBS formulations. In Section 3.5, we discuss the components of the framework and their the implementation details including the different algorithms used. Finally, in Section 3.6 we provide some code examples describing the framework with different curve and surface examples.

3.3 Design Considerations

NURBS-Python is an object-oriented NURBS evaluation library with data structures suited for geometric operations. NURBS have a compact definition any NURBS shape (curve or surface) can be defined by its degrees, knot vectors, and a set of control points. These are usually input by the user, however, the library is also capable of automatically generating a uniform knot vector, partly simplifying the knot vector input for some cases. The geometric output variables are computed after evaluating the shape. The curve and surface objects are interactive; it is possible to change any input variable at runtime and the library re-evaluates the shapes automatically.

In order to achieve the best compatibility, NURBS-Python is designed to only use modules that come with the core python distribution (Rossum, 1995), also called as Python Standard Library including the mathematical evaluation libraries. Having no external dependencies allows users to have a lightweight and portable package that can be integrated with different architectures with minimal effort. A self-contained pure python library also protects users from binary interface incompatibilities and eliminates extra compilation steps and installation of third-party software for running a simple code segment. Due to its modular and object-oriented nature, the evaluation capability of NURBS-Python is easily extensible to a variety of platforms, such as HPC clusters or GPUs.
Moreover, it can be used for various use cases such as integration with CAD, CAM, robotics, and machine learning libraries and for educational purposes for teaching geometric modeling concepts. Development of NURBS date back to 1950s and therefore, it would be unwise to think that NURBS-Python is the only library of its kind. However, we are not aware of any pure-Python standalone NURBS evaluation library for direct comparison. Nevertheless, we would like to compare our library with some of the existing open-source libraries containing NURBS components. We would like to note that the following libraries are not stand-alone NURBS evaluation libraries and mostly designed to support other uses such as isogeometric analysis. One of the most famous and commonly used Python library for isogeometric analysis is \textit{igakit} (Dalcin and Collier, 2018). Although \textit{igakit} has a complete NURBS evaluation implementation, it is heavily dependent on third-party libraries and require compilation for usage. Additionally, \textit{igakit} does not provide a separate package for its NURBS evaluation module. Moreover, the software design approach between \textit{igakit} and NURBS-Python is different. The NURBS evaluation component of \textit{igakit} uses a mathematical approach, which doesn’t differentiate between the dimension of the NURBS shapes. On the other hand, NURBS-Python’s design approach is focused specifically on curves and surfaces, and can be extended to volumes. Although, this is just a matter of design preference, we believe that our approach improves the usability of the library. There are also several NURBS libraries developed using MATLAB/Octave (de Falco et al., 2011; Vázquez, 2016; Nguyen et al., 2015). These implementations are also mainly designed to support isogeometric analysis as well as developed in programming languages that are considered proprietary or closed-source.

### 3.4 A Brief Introduction to NURBS

A NURBS shape is defined as a vector-valued function of one or more parameters which maps a \(k\)-dimensional space into, at minimum, a \(k + 1\)-dimensional space. This function is simply the vector product (or a tensor product, depending on the \(k\) value) of basis or blending functions by a set of \(n\)-dimensional control points (Piegl and Tiller, 2012; Cottrell et al., 2009).
The basis functions in NURBS are evaluated using the Cox-de Boor recursion algorithm as described in Equation 3.1, where $\xi$ is the parameter value, $N_{i,p}$ is the $i$th order basis function and $p$ is the degree of the shape defined for the parametric dimension in consideration (Piegl and Tiller, 2012).

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \leq \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases} \quad (3.1a)$$

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p} - \xi_i}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \quad (3.1b)$$

The derivatives of the basis functions can be evaluated using Equation 3.2 (Piegl and Tiller, 2012).

$$\frac{d^k}{d\xi^k} N_{i,p}(\xi) = \frac{p!}{(p-k)!} \sum_{j=0}^{k} \alpha_{k,j} N_{i+j,p-k}(\xi) \quad (3.2a)$$

$$\alpha_{0,0} = 1 \quad (3.2b)$$

$$\alpha_{k,0} = \frac{\alpha_{k-1,0}}{\xi_{i+p-k+1} - \xi_i} \quad (3.2c)$$

$$\alpha_{k,j} = \frac{\alpha_{k-1,0} - \alpha_{k-1,j-1}}{\xi_{i+p+j-k+1} - \xi_{i+j}} \quad (3.2d)$$

$$\alpha_{k,k} = -\frac{\alpha_{k-1,k-1}}{\xi_{i+p+1} - \xi_{i+k}} \quad (3.2e)$$

where $j = 1, \ldots, k - 1$. The derivatives are used to compute tangents, normals, and binormals of the NURBS shapes.

After computing the basis functions, a single point on the curve corresponding to the parameter $\xi$ is evaluated by simply multiplying the basis functions with the corresponding control points, $B_{i,j}$ and summing up the multiplication results, as described in Equation 3.3 (Piegl and Tiller, 2012).

$$C(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) P_i \quad (3.3)$$
The same evaluation method also applies to the surfaces. Surfaces are defined on a 2-dimensional parametric space described using \((\xi, \eta)\). To calculate a single point corresponding to these parameters, the basis functions on each parametric dimension, \(N_{i,p}(\xi), M_{i,q}(\eta)\), need to be evaluated and multiplied with the control points, \(P_{i,j}\), and summed up as described in Equation 3.4 (Piegl and Tiller, 2012).

\[
S(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) P_{i,j}
\] (3.4)

These equations are also applicable to Bzier curves and surfaces, since NURBS are a superset of Bzier shapes.

NURBS shapes can be divided into two types with respect to their knot vector structure: clamped and unclamped. A clamped shape can be understood from the repetitions of the knots at the beginning or at the end of its knot vectors. In an unclamped shape, there would be no repeating knots on both ends of the knot vector. Unclamped shapes still follow all NURBS properties and they can be evaluated using the same equations. For instance, the knot vector in Equation 3.5 defines a clamped shape at both ends, where \(p\) is the degree and \(m + 1\) corresponds to the number of knots in the knot vector (Piegl and Tiller, 2012).

\[
U = \{a, ..., a, u_{p+1}, ..., u_{m-p-1}, b, ..., b\}
\] (3.5)

All knot vectors obey the following equation:

\[
m = p + n + 1
\] (3.6)

where \(p\) is the degree, \(m + 1\) is the number of knots in the knot vector and \(n + 1\) is the number of control points (Piegl and Tiller, 2012). The values in the knot vectors are non-decreasing. These properties are used to validate the knot vectors.
3.5 Components of the Framework

We describe the main components and the features of NURBS-Python on the following sections. More details can be found in 3.7.

3.5.1 Core Components and Data Structures

The core component involves the data structures for representing the shapes in the form of curves and surfaces and the evaluation functionality as well as the abstraction layer for providing extensibility. It includes input validation methods, which validates all user inputs with respect to the mathematical description of NURBS discussed in Section 3.4, and a caching system, which is directly integrated to improve interactivity of the library. NURBS-Python provides abstraction via the Abstract module. This module provides templates for future extension of the library and
tries to maintain the programming interface as standard as possible between the current and the extended modules.

The data structure and the evaluation operations are available in NURBS and BSpline modules representing rational and non-rational versions of the non-uniform basis splines, respectively. The only difference between these modules is the structure of the control points. Evaluation functionality in NURBS module requires weighted control points, whereas BSpline requires no weights. The logic behind this design is generating an easy-to-understand environment by logically separating rational and non-rational algorithms and eliminating the need for extra information, such as weights, for users who only prefers to work with non-rational surfaces and curves.

The BSpline and NURBS modules contain two classes for representing the geometrical shapes. Curve class represents a single-manifold n-dimensional curve shape and Surface class represents a two-manifold 3D surface shape. All NURBS and B-Spline classes implement evaluate method for shape evaluation and evalpts property to retrieve evaluated points. These classes automatically
evaluate the shape when required, such as plotting the shapes or retrieving the evaluated points from the object instance; therefore, it is not needed to explicitly evaluate the shape before any operation that requires the evaluated shape. The class diagrams of BSpline and NURBS modules showing their inheritance are shown in Figures 3.1 and 3.2. Moreover, the Multi module contains two classes, MultiCurve and MultiSurface, for storing and evaluating multiple curves and surfaces simultaneously.

### 3.5.2 Curve and Surface Evaluation

The curve and surface evaluation is handled by the Evaluator module and the abstract base class used is AbstractEvaluator. The class diagram showing the inheritance of the included Evaluator modules are shown in Figure 3.3. This module contains knot vector span finding algorithms using linear and binary search techniques, basis function and point evaluation algorithms as
Figure 3.4 Generating a triangulated surface with NURBS-Python using $\text{delta} = 0.25$. (a) Positions of the evaluated points on the parametric space, (b) triangulated surface on the parametric space, (c) A scatter plot on the 3D space illustrating the positions of the individual evaluated surface points. (d) Triangulated surface in 3D.

suggested by Piegl and Tiller (2012). The Evaluator modules are designed to allow users to extend algorithms easily or use them as an evaluation strategy, i.e. change them at runtime without need of re-creating the object instance. Therefore, it makes easier to compare, mix and reuse different evaluation algorithms with the same shape data.

In order to evaluate a shape (a curve or a surface), the user first sets the degrees, the knot vectors, the control points and the sample size or evaluation delta which corresponds to the number of points to be evaluated by NURBS-Python. The included evaluation algorithms can handle clamped and unclamped shapes. The evaluation interval delta, or the $\text{delta}$ property, should be between 0.0 and 1.0, otherwise the library warns the user to pick a value between the interval. Sample size, represented by the $\text{sample\_size}$ property, can be any number bigger than 1 and it should be an integer value. NURBS-Python is designed to fix any discrepancies by applying type casting when a value could properly be converted into the correct type without additional user input.

The following equation shows the relation between $\text{delta}$ ($d$) and the $\text{sample\_size}$ ($S$):

$$S = \frac{1}{d} + 1 \quad (3.7)$$

The evaluation of $\frac{1}{d}$ mostly results in a floating point value; therefore its result is type-casted to the integer primitive type. The type casting operation rounds the result of the division to the
Figure 3.5 NURBS curves and surfaces generated by the library and plotted using Matplotlib implementation of the visualization component. (a) A circular NURBS shape generated from 7 control points, (b) Bzier decomposition of the circular shape, each curve segment is colored differently, (c) A cylindrical NURBS surface, (d) Bzier decomposition of the cylindrical surface, each surface patch is colored differently. The colors on the decomposed shapes are randomly generated.

lower integer value. The \texttt{delta} property is set to 0.01, by default. The main steps in evaluating curves and surfaces are:

1. Find spans on the given knot interval which are determined via the input sample size.
2. Compute basis functions for the spans using Equation 3.1.
3. Evaluate the shape by finding all the control points that belong to the given knot interval and performing the multiplication with the basis functions and summation using Equations 3.3 or 3.4.

After evaluation, the evaluated points are automatically cached and can then be accessed using the \texttt{evalpts} property. The internal cache is used to speed up the responsiveness of the library, since the evaluated points could only change if any of the evaluation variables (degree, knot vector, control points, or the evaluation delta) changes. If any change occurs in these variables, the cache is reset and the shape is re-evaluated.

NURBS-Python also has capabilities to evaluate derivatives of the curves and surfaces using the algorithms suggested by Piegl and Tiller (2012). The method \texttt{derivatives} is designed to evaluate $n^{th}$ order derivatives of the curves and surfaces. The geometric interpretation of the derivatives,
such as tangents, normals, and binormals, have their own methods, tangent, normal, binormal respectively. Curve tangents are computed from the 1st derivative at the given parametric value, normals are computed from the 2nd derivative, and binormals are computed by vector cross-product of tangents and normals. These vectors correspond to the Frenet-Serret Frame which is a right-handed system of a pairwise orthonormal vectors that follow the curve. In a Frenet-Serret frame, the tangent \((T)\), normal \((N)\) and binormal \((B)\) vectors are perpendicular to each other and can be computed from 1st derivatives, 2nd derivatives, and via the equality \(B = T \times N\), respectively. Surface tangents are computed from the 1st derivative with respect to each parametric direction, and the surface normal is computed by vector cross-product of the tangents in each direction.

3.5.3 Shape Splitting and Knot Insertion

The Multi module has capabilities to evaluate multiple curves and surfaces. NURBS-Python uses Multi objects to return split or decomposed shapes. Curve classes have split and surface classes have split_u and split_v methods corresponding to each parametric direction. Both curve and surface classes have decompose methods for Bzier decomposition. Both shape splitting and decomposition can be achieved by using an evaluator method insert_knot which is implemented using the algorithms by Piegl and Tiller (2012).

The splitting algorithm takes a parameter value \(u\) between 0.0 and 1.0, finds the multiplicity \(s\) of the parameter over the knot vector in the chosen parametric direction and inserts \(r\) number of knots (calculated using the equation \(r = p - s\), where \(p\) is the degree of the shape in the chosen parametric direction). Although this operation is simply considered as splitting (Piegl and Tiller, 2012), it is not enough to generate 2 different shapes with separate knot vectors and control points arrays. In order to generate two different shapes, NURBS-Python needs to find the exact span on the knot vector which defines the split point. The split knot span in consideration can be found by adding 1 to the input parameter value, \(u + 1\). NURBS-Python then splits the knot vector using the split knot span into 2 separate vectors. Since the generated shapes are always clamped, the \(u\) value is added to the end of the first knot vector and to the beginning of the second knot vector to
satisfy the rules discussed in Section 3.4. The control points array is separated into two arrays at the split location by adding the span of the input parameter, \( u \) and the number of knots inserted, \( r \). Finally, the separated knot vectors and control points arrays are saved into the \texttt{MultiCurve} or \texttt{MultiSurface} container object.

Bziers decomposition is performed by applying tree expansion on the shapes. The shape is split at the middle knot locations. i.e. the ones that are not 0.0 or 1.0s. The split shapes can be considered as the leaf nodes. If any leaf node is still splittable (i.e. not a Bzier segment or a patch), the algorithm continues splitting at the middle knot locations until no middle knots remain in the knot vectors. For curves, the decomposition algorithm is directly applied as they are described with one parametric direction. The resultant curve segments are stored in a \texttt{MultiCurve} container object. For surfaces, the decomposition algorithm is applied on first \( v \) parametric direction and then \( u \) parametric direction. Similar to the decomposed curves, the resultant Bzier surface patches are stored in a \texttt{MultiSurface} container object.

### 3.5.4 Surface Generator

We designed the surface generator module to generate sample surfaces for our previous work on incorporation of defects on layered composite structures (Bingol et al., 2019). The classes Grid and GridWeighted are used to generate surfaces that are compatible with BSpline and NURBS, respectively. The class diagrams are displayed on Figure 3.3. Both classes are initialized with width, \( x \) and height, \( y \) with zero thickness. The grid mesh is generated by inputting number of divisions in both width and height directions. These divisions define the control points locations and they are uniformly distributed over the generated surface. These steps are enough for generating a planar surface in desired dimensions with desired number of control points. Additional details on surface generation can be found in 3.7.1.
3.5.5 Exporting and Importing Data

Despite being designed as a low-level library, NURBS-Python includes a CAD interoperability and exchange module for extending its usability with the other software. The interoperability module provides control points grid manipulation operations, such as changing the array structures from weighted to unweighted, extracting and replacing weights and changing the row order. The exchange module provides support for exporting control points and evaluated points as CSV and VTK polydata formats, and exporting surfaces using common CAD formats, such as OBJ, STL, and OFF.

Since the CAD formats defined here require a triangulated surface, NURBS-Python includes a simple triangulation functionality (Figure 3.4). This functionality uses the parametric correspondence of the 3D surface points using the \texttt{delta} or \texttt{sample.size} property as described using Equation 3.7 to generate the triangle mesh. These properties determine the distance between the surface points on the parametric space and this information can be used to pick the closest points that could generate non-overlapping triangles. The size of the triangles, and therefore the smoothness of the surface can be fine-tuned by the user input.

NURBS-Python is designed to work with Python's default container classes, such as \texttt{list} and \texttt{tuple} to import new shapes. We implement some importing functionality from text files and \texttt{libconfig}-type files out-of-the-box; however, we prefer not to invent another file format and instead make the package work with Python's default container classes. Any class or package that can output the data as \texttt{list} or \texttt{tuple} is compatible with NURBS-Python for data import. For libraries that output the control points in different formats, such as $(x,y,z,w)$ in OpenNURBS (Robert McNeel & Associates, 2018) or separate $(x,y,z)$ and $w$ arrays, where $x,y,z$ are the coordinates of the control points and $w$ is the weight value, NURBS-Python's \texttt{compatibility} module can be used to convert the control points into the format that NURBS-Python can read as well as saving them as text files.
Importing from the CAD exchange formats, which directly support NURBS data structures, such as IGES and X3D, is still work in progress and will be released in the next major version of NURBS-Python.

3.5.6 Visualization Components

NURBS-Python implements 2 common python visualization libraries, Matplotlib (Hunter, 2007) and Plotly (Plotly Technologies Inc., 2015) using its abstract base classes, VisAbstract for general plotting and VisAbstractSurf for surface plotting customizations. These provide native ability to visualize the NURBS shapes. Their class diagrams are illustrated on Figure 3.3. Figures 3.5 and 3.7 illustrate single and multi surface visualization capabilities using different plotting libraries. Figure 3.6 illustrates a 2-dimensional curve unclamped on the both sides and plotted using Matplotlib implementation of the visualization component.

The curve visualization classes VisCurve2D and VisCurve3D directly uses the line plots and surface visualization class VisSurface uses the triangulation method discussed in Section 3.5.5. To plot the control points grid mesh, VisSurface uses an algorithm that connects the closest 4 control
points to generate quads. The closest points are identified from the structure of the control points array.

Each visualization class can be configured using a configuration class. The abstract base of the configuration classes is `VisConfigAbstract` as referred in Figure 3.3. The configuration class can only make visual changes to the output curve or surface plot, such as changing the figure size and the resolution, hiding legend from the plot as illustrated in Figure 3.8. In our design, it is only possible to configure at the visualization instance generation step, otherwise it will use the default configuration implementation with optimal values (i.e. everything is visible on a screen with 1280x720 pixels resolution) set inside the visualization modules.

### 3.5.7 Testing and Algorithm Validation

To test the library and validate the results of the geometrical algorithms, we have implemented unit and functional tests for NURBS-Python using `pytest` and all tests are connected to a continuous integration (CI) system for automated testing of the library. We have achieved the geometrical
evaluation validity by implementing multiple tests with different inputs for checking the complete shape evaluated with a relatively large delta value or only specific regions (e.g. only validating the affected region after the knot insertion operation).

The automated tests included cover all the algorithms, python properties (getters and setters) and most of the helper functionality. We were able to achieve full code coverage via the tests on the algorithms validation excluding the input checking and data validity parts of the methods. At the time of this writing, we were able to achieve around 70% code coverage with 257 automated tests.

### 3.6 Code Examples

The following examples illustrate how to generate a curve and a surface, and then visualize it using NURBS-Python. We start with a 3-dimensional curve example.
from geomdl import BSpline, utilities
from geomdl.visualization import VisMPL

# Create a curve instance
crv = BSpline.Curve()

# Set curve degree
crv.degree = 3

# Set control points
ctrlpts = [[10, 5, 10], [10, 20, -30],
           [40, 10, 25], [-10, 5, 0]]

# Auto-generate the knot vector
knotvector = utilities.generate_knot_vector(crv.degree, len(crv.ctrlpts))

# Evaluate the curve
crv.evaluate()

# Set the visualization component
vis = VisMPL.VisCurve3D()

# Plot the curve
vis.render()

The code listing starts with importing the modules and then we create a curve instance, controlled by the variable **crv**. We set the curve degree and input control points using the property **ctrlpts**. The control points are represented as list of n-dimensional coordinates using Python lists. Using the **utilities** module, we generate a uniform knot vector automatically and set it using the **knotvector** property. Finally, we evaluate the curve, although the library would automatically evaluate the curve or the surface when the the evaluated points are requested by an internal component or a user. The evaluation method computes the curve points using the default Evaluator
algorithm. For the visualization part, we set the visualization module designed for plotting 3D curves using the `vis` property and executing `render` method of the Curve class will plot the 3D curve by calling Matplotlib functions.

The following code listing generates a surface using NURBS-Python and plots the surface using the Plotly implementation of the visualization component.

```python
from geomdl import BSpline, utilities
from geomdl.visualization import VisPlotly

# Create a surface instance
surf = BSpline.Surface()

# Set degrees
surf.degree_u = 3
surf.degree_v = 2

# List of control points
control_points = [[0, 0, 0], [0, 4, 0], [0, 8, -3],
                  [2, 0, 6], [2, 4, 0], [2, 8, 0],
                  [4, 0, 0], [4, 4, 0], [4, 8, 3],
                  [6, 0, 0], [6, 4, -3], [6, 8, 0]]

# Set control points
surf.set_ctrlpts(control_points, 4, 3)

# Auto-generate knot vectors
surf.knotvector_u = utilities.generate_knot_vector(surf.degree_u, surf.ctrlpts_size_u)
surf.knotvector_v = utilities.generate_knot_vector(surf.degree_v, surf.ctrlpts_size_v)

# Set sample size
surf.sample_size = 25
```
# Evaluate surface
surf.evaluate()

# Set the visualization component
vis_component = VisPlotly.VisSurface()
surf.vis = vis_component

# Plot the surface
surf.render()

The surface generation example is similar to the curve generation example. The main difference is in setting the control points. The control points shown with the variable `control_points` on the above example are stored in a list of 3D coordinates. However, a surface is defined over a 2-dimensional parametric space and therefore, requires a grid of control points. To allow user input as a single dimensional array of coordinates, we implemented a structure only applicable to the surfaces. On this structure, the \( v \) index varies first. That is, a row of \( v \) control points for the first \( u \) value is found first. Then, the row of \( v \) control points is found for the next \( u \) value. This variation is controlled by a separate function, `set_ctrlpts` as Python properties cannot be arranged to accept multiple variables as the same time. We also didn’t want to confuse the users by implementing structures, such as Python dictionaries as the input. The `set_ctrlpts` function takes the control points and the number of control points in \( u \) and \( v \) directions as the input.

It would not be possible to provide examples for all the features of NURBS-Python in this paper, and hence we have also released a set of example scripts publicly on Github with the intention of providing templates to the NURBS-Python users. We constantly add more examples for the new features, integration and usage scenarios that we encounter while using the library. We encourage NURBS-Python users to refer to the examples repository (https://github.com/orbingol/NURBS-Python_Examples) for more possible usage and integration scenarios.
3.7 Additional Components of the Framework

3.7.1 Surface Generator Customization

Although generating a planar surface grid in desired size and exporting it as a text file for further customizations could be enough for most users, NURBS-Python also provides facilities to manipulate the shape of the generated surface. The bumps method includes an algorithm that allows users to generate hills (or bumps) on the surface. This algorithm generates 2 random numbers corresponding to width and height on the interval of the generated surface. These numbers correspond to the location (coordinates) of the peak of the hill to be generated. Then, the algorithm checks for surrounding locations for existing hills (i.e. non-zero \(z\) value). If there are no hills generated previously, then the method applies the \(z\) value, which is a user input argument named as \(bump\) \(height\), to the peak location and the surroundings are generated by gradually dividing \(z\) value to value computed by another input argument \(base\_extent\) which simply generates a gradient from the peak of the hill to the base. In addition, the users can input a padding value using \(base\_adjust\) argument which confines (i.e. a negative \(base\_adjust\) value) or extends (i.e. a positive \(base\_adjust\) value) the area on the \(x\)-\(y\) plane of the grid where the hills are generated. The algorithm can pick either \(+z\) or \(-z\) direction to generate the hill. Since the algorithm depends on random value generation, it could get stuck on an infinite loop. Therefore, the algorithm stops after 25 hill generation trials by default, and number of trials can be changed using the max\_trials input argument.

In addition to the hill generation algorithm, the surface generator also provides geometric operators for rotating the surface on \(x\), \(y\), and \(z\) axes about the input angle, and translation of the surface center to the input 3-dimensional location using the translate method.

Users can query the bounding box of the shape using bbox property. This property, when called by the user, automatically computes the bounding box of the evaluated shape and caches the values to eliminate excess bounding box computations. After the first computation, the values are always returned from the internal cache.
3.7.2 Visualization Customization

The visualization component can be set or changed at runtime using the `vis` property of the `Curve` and `Surface` classes. The plotting of the shape takes place when the user calls `render` method of the these classes. The plotting behavior can be controlled with additional input keyword arguments of the `render` method. For instance; the user can save the plot with or without opening the plotting window or change the color of the control points and shape plots.

The library allows re-using all possible visualization options on the designated shape element. This means that a single `VisSurface` instance can be used to plot different surfaces contained in different `Surface` instances in `BSpline` or `NURBS` modules. The same applies to the `Curve` classes. However, it is not possible to use a surface visualization object with a curve class instance, or vice versa, due to inherent differences in the data structures.

3.8 Additional Code Examples

The following code listing demonstrates the surface generator module, `CPGen` and its interoper-}

```python
from geomdl import BSpline, CPGen, utilities
from geomdl.visualization import VisMPL as vis
from geomdl import exchange

# Generate a plane with the dimensions 50x100
surfgrid = CPGen.Grid(50, 100)

# Generate a 10x10 grid
surfgrid.generate(10, 10)

# Generate 1 bump at the center of the grid
surfgrid.bumps(num_bumps=1, all_positive=True, bump_height=45, base_extent=4,
                base_adjust=-1)
```
# Create a BSpline surface instance
surf = BSpline.Surface()

# Set order of the surface
surf.order_u = 4
surf.order_v = 4

# Get the control points from the generated grid
surf.ctrlpts2d = surfgrid.grid

# Set knot vectors
surf.knotvector_u = utilities.generate_knot_vector(surf.degree_u, surf.ctrlpts.size_u)
surf.knotvector_v = utilities.generate_knot_vector(surf.degree_v, surf.ctrlpts.size_v)

# Set sample size of the surface
surf.sample_size = 30

# Visualization component and its configuration
conf = vis.VisConfig(ctrlpts=False, legend=False)
surf.vis = vis.VisSurface(conf)

# Plot the surface
surf.render()

# Export the surface as a .stl file
exchange.export_stl(surf, "surface.stl")

In this example, we have generated the control points grid using the surface generator module, represented by CPGen. Then, we generate a bi-cubic surface and automatically generate uniform knot vectors on each parametric direction. The generated surface is plotted using the Matplotlib component of the visualization module and finally, saved as a .stl file.
The following example illustrates the control points import facility of NURBS-Python along with Bzier decomposition and translation functionalities. The control points file `ex_surface03 cptw` is an ASCII text file and it can be found on the examples repository.

```python
from geomdl import NURBS
from geomdl import exchange
from geomdl import operations
from geomdl.visualization import VisMPL

# Create a NURBS surface instance
surf = NURBS.Surface()

# Set degrees
surf.degree_u = 1
surf.degree_v = 2

# Set control points
surf.set_ctrlpts(*exchange.import_txt("ex_surface03 cptw", two_dimensional=True))

# Set knot vector
surf.knotvector_u = [0, 0, 1, 1]
surf.knotvector_v = [0, 0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1, 1, 1]

# Decompose the surface
surfaces = operations.decompose_surface(surf)

# Translate one of the surface patch
operations.translate(surfaces[1], (-0.25, 0.25, 0), inplace=True)

# Set number of samples for all split surfaces
surfaces.sample_size = 50

# Plot decomposed surfaces
vis_comp = VisMPL.VisSurfWireframe()
```
As described in the previous examples, we generate a NURBS surface instance using a control points file. The initial surface is decomposed into Bzier patches and right after the decomposition, one of the Bzier patches is translated by the vector $[−0.25, −0.25, 0]$. Finally, decomposed surfaces are plotted via Matplotlib implementation of the visualization module.

### 3.9 Performance Metrics

It would not be possible to reach any conclusions from the running time of the interpreted code. However we have used a performance improvement method using an external module called Cython (Behnel et al., 2011). Cython corresponds to a compiler specifically designed for wrapping external code into a compiled Python module.

To assess the performance difference between the interpreted and the compiled versions, we compiled NURBS-Python with the Cython compiler and tested using a sample curve and a surface. We used a sample size (i.e. number of evaluated points) $S = 16384$ for the curves and $S = 1024$ for both parametric directions of the surface, resulting in a total of 1048576 evaluated surface points for each surface. Table 3.1 shows our evaluation results in the format of mean ± standard deviation obtained from a computer with Intel Core i7-7700HQ CPU and 16 gigabytes of RAM. The results are measured by applying IPython’s `%timeit` magic on the `evaluate` method with 7 runs. The software versions used for the analysis are Python v3.6.6 and IPython v6.5.0.

<table>
<thead>
<tr>
<th>Library Type</th>
<th>Curve</th>
<th>Surface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted</td>
<td>167 ms ± 6.97 ms</td>
<td>18 s ± 10.8 s</td>
</tr>
<tr>
<td>Compiled</td>
<td>89 ms ± 2.55 ms</td>
<td>6.41 s ± 263 ms</td>
</tr>
</tbody>
</table>

Table 3.1 Comparison of evaluation time between interpreted and Cython-compiled versions of NURBS-Python. Sample sizes: $S_{Curve} = 16384$ and $S_{Surface} = 1048576$.

As expected, we were able to get faster evaluation speeds using the compiled version. The speed increase we obtained by direct Cython compilation was around 2 and 3 times on curves and surfaces,
respectively. The most important thing to consider while performing the Cython compilation is that due to NURBS-Python being a pure Python library with no external dependencies, the compilation and linking requires no additional libraries other than the Python standard library.

3.10 Conclusions and Future Work

We have introduced an open-source object-oriented geometric modeling library with visualization options. We have publicly released the library on https://github.com/orbingol/NURBS-Python and in addition, we provide over 40 example scripts that illustrate the features of the library and some sample usage scenarios on a separate GitHub repository. The scripts to generate some of the figures illustrated on this paper can also be found in that repository. We also provide a complete class documentation with more examples and figures. The documentation is automatically generated and published on ReadTheDocs, a free documentation generation and publishing website. Users can also access to the other reports, such as continuous integration system logs and code coverage graphs via project’s GitHub page. To increase the accessibility of the library on different platforms and reduce the user effort for installation, we have uploaded NURBS-Python to Python Package Index (pypi.org) and Anaconda Cloud (anaconda.org), allowing users to download the library using the package managers pip and conda.

NURBS-Python is designed to be an extensible and open-source framework for geometric modeling. Since it is freely available on a public domain, developers can extend the library in their own liking or integrate it in their own works. Nevertheless, we would like to add some comments on our current work and some possible extension paths for the NURBS-Python library. We will be adding additional spline algorithms, such as knot removal, degree elevation and reduction, as well as fitting, trimming, offsetting and volume parameterizations. We are currently developing a module called shapes for allowing users to generate commonly used NURBS shapes, such as circles, cylinders, torus, etc. Finally, extending the framework to support truncated hierarchical B-splines (THB-splines), T-splines, and polynomial/rational splines over hierarchical T-meshes (PHT/RHT-splines) (Nguyen-Thanh et al., 2017) for adaptive geometric design would be a nice
path for further extension of the library to support engineering applications, such as isogeometric analysis for structural mechanics (Hughes et al., 2005).

Acknowledgements

We would like to express our deepest gratitudes to all NURBS-Python users and contributors around the world for their time and efforts in testing NURBS-Python, reporting the bugs and commenting on the features. These contributions helped us to develop a solid NURBS evaluation framework for Python. As a courtesy, we have included their names in CONTRIBUTING.rst file on our GitHub repository.

3.11 References


CHAPTER 4. MODELING AND ISOGEOMETRIC ANALYSIS OF LAMINATED COMPOSITE STRUCTURES

A paper to be submitted to Computer-Aided Design
Onur Rauf Bingol, Emily Johnson, Ming-Chen Hsu, Adarsh Krishnamurthy

4.1 Abstract

Laminated fiber-reinforced polymer (FRP) composites have been increasingly used in structural applications because of their combination of good mechanical properties and low weight. However, owing to their high costs, computational modeling of composites can provide valuable insights into their design and repair decisions. Traditionally, composite laminates are modeled using Kirchhoff-Love thin shells for performing structural analysis. However, the thin shell method has limitations on computing the interlaminar stress and strains of the composite laminates and does not allow modeling of common interlaminar defects, such as delaminations. In this work, we have developed an integrated modeling and structural analysis framework for laminated composite using NURBS volumes and isogeometric analysis (IGA). Using NURBS volumes instead of thin shells provides higher fidelity for the structural analysis by generating the digital twin of the composite manufacturing processes and allowing for interrogation of interlaminar stresses in the structure. Moreover, isogeometric analysis with NURBS volumes allows faster and more stable analysis compared to traditional finite element analysis (FEA) methods. We demonstrate the utility of our framework by comparing the results of a well-studied benchmark problem using our method. We also use our framework to model a multiply-curved complex composite structure and perform static structural analysis using IGA.
4.2 Introduction

Fiber-reinforced polymer (FRP) composite laminates are becoming an integral part of the modern aerospace and automobile industries due to their high strength combined with their low weight. According to Boeing, 50 percent of the 787 aircraft, including the fuselage and the wings, are composed of advanced composites having the carbon fiber composites as the majority (Hale, 2006). Although they can perform as well, or even better by means of low weight, as the metallic counterparts, the overall manufacturing of the FRP composites is more expensive than the metals due to high material and process costs. The high cost of manufacturing has been led to several advanced in virtual manufacturing and testing using computational methods (D’Mello et al., 2016; Davies and Ankersen, 2008). Computational structural mechanics models of FRP composites can help reduce costs by allowing for optimization of the composite design. In addition, predictive damage models can be used to understand the effect of manufacturing and service defects on the remaining service life of composite structures (Talreja, 1989).

Traditionally, laminated composites are modeled using Kirchhoff-Love thin shells for performing structural analysis (Reddy, 2006; Kiendl et al., 2009; Pigazzini et al., 2018). In thin-shell analysis, the composite laminate is modeled only using its mid-surface and the material properties are homogenized over the thickness. This allows for a simplified analysis using only a surface, which can be represented using standard surface representations such as non-uniform rational B-splines (NURBS). However, it is not possible to get interlaminar quantities of interest, such as interlaminar stresses and strains, using the thin shell analysis. In addition, special techniques are required to model common interlaminar defects, such as delaminations, using only thin shell analysis (Pigazzini et al., 2019). Analyzing interlaminar phenomenon require explicit modeling of the laminar structure of the composites using solid elements.

Another main challenge in modeling laminated composites is to develop the geometric tools required to construct 3-dimensional models that represent the layered structure of the laminate. In our previous works (Bingol et al., 2017, 2019), we introduced layer-by-layer modeling concept which directly follows the composite laminate manufacturing process (Soutis, 2005) using the traditional
solid modeling systems. This concept uses a mold surface and using offsetting techniques included in the solid modeling system, the 3-dimensional model of a layer is generated. The latter can be generated from another mold surface or any surface of the existing layers. We observed that traditional solid modeling systems are not designed to directly handle multi-layer structures, like composite laminates with defects, when *layer-by-layer modeling* approach is used. Such systems are mainly designed to operate using Boundary Representation (B-Rep) and convert the structure to NURBS when necessary, which adds an disintegration between the techniques used and a possible loss of data during the conversion process. In addition, although it is possible to generate thin structures with the traditional solid modeling systems, it is relatively complicated to generate multi-layer thin structures. Using NURBS volumes to directly model the composite laminates would eliminate these problems and also provide a way to directly operate on the exact geometry.

Explicit multi-layer model of composites can provide high fidelity results of structural analysis including allowing for interrogating interlaminar stresses and strains. However, modeling each layer explicitly using traditional finite elements gets computationally intractable. This is because traditional finite elements require elements with aspect ratios close to 1.0 for better accuracy and convergence. This necessitates the use of highly refined elements with length and width of the same order of dimension as the thickness for the analysis, which leads to an $\mathcal{O}(n^2)$ increase in the number of elements for the analysis, making it computationally expensive. Isogoeometric analysis (IGA) (Hughes et al., 2005), which uses the same NURBS functions for both the geometry and analysis, can better handle such thin structures (Cottrell et al., 2009; Schmidt et al., 2010). IGA has been shown to provide better accuracy and convergence properties with fewer, high-order elements.

The parameterization of the geometry is very important to achieve convergent results in IGA. This concept is named as *analysis-aware modeling* (Cohen et al., 2010). There are several methods proposed to achieve *analysis-aware modeling* (Aigner et al., 2009; Wang et al., 2007; Martin et al., 2009; Xu et al., 2011, 2013b,a), however; they are not suitable for the *layer-by-layer modeling* approach, since this modeling approach involves surface offsetting. Computation of offset surfaces are very well studied from late 1980s (Pham, 1992; Maekawa, 1999). These methods are mostly
focused on generating a non-self intersecting offset surfaces having uniform distance between from its the progenitor, but not generating the NURBS volumes. Constructing volumetric NURBS of thin structures is different from creating a solid model of the thin structure. Specifically, the knot vectors needs to be shared between the surfaces that represent the interface between layers. As a consequence, standard NURBS surface offsetting methods cannot be used to construct volumetric NURBS of thin structures. Although it is always possible to compute the control points of the offset surface from the surface points using approximation and interpolation techniques (Piegl and Tiller, 2012), it is shown that this is not feasible due to problems in parameterization (Brakhage and Lamby, 2008).

In this work, we introduce a new method to model laminated composites using volumetric NURBS as an extension to our previous work using layer-by-layer approach (Bingol et al., 2019). We use an input NURBS surface as a mold and generate its offset while maintaining the tensor product structure. As a result, each layer (or lamina) is generated as a NURBS volume. The layers are combined to generate the laminated composite structure. Once the composite laminate is generated, we use IGA to run structural analyses. We use NURBS-Python (Bingol and Krishnamurthy, 2019) library for modeling surfaces and volumes and we developed an object-oriented pure Python analysis library, gIGA, for static structural analysis.

The main contributions of this work are:

- An extension of the previously-introduced layer-by-layer modeling method to apply the laminated composite manufacturing process to generate multi-layer structures with NURBS
- A new approach to generate NURBS volumes from surfaces
- An object-oriented, pure Python, easy to use and install Isogeometric Analysis library, gIGA

4.3 Modeling laminated composites using NURBS volumes

To generate 3-dimensional models of laminated composites, we apply layer-by-layer modeling approach which follows the manufacturing process of the composite laminates. Each layer, represented as a NURBS volume, is generated from a mold, which is represented as a NURBS surface.
The mold can be a user input, or one of the faces of the generated layers. The layers are combined together to generate a laminate, which is also represented as a NURBS volume.

4.3.1 Construction of individual layers

Individual layer generation depends on the input mold surface and affects the final laminate. The input is a NURBS surface. Using the input surface, a NURBS volume representing each layer composing the laminate is generated. Composite laminates have high surface area to thickness ratio. For instance, composite laminates for manufacturing wings are designed with a varying total relative thickness from 11% to 15% (Brakhage and Lamby, 2005). On the other hand, bigger thickness values could cause excessive reduction on the strength of the laminate (Lee and Soutis, 2005). Although the thickness of the laminate depends on the user input and the application, it would be logical to assume that each layer has a very small thickness and therefore, detailed representation of the top and the bottom surfaces would be enough for analysis of interlaminar properties of the composite laminates. Hence, we start with a parametric surface representing the top or the bottom side of the composite layer as the initial input to start generating a single layer. The input surface is called progenitor surface and it can be rational or non-rational.

Using the progenitor surface, a corresponding surface is generated using the thickening operation which is described as generating realistic offsets of the surfaces (Stroud, 2006) and it can be used to construct the detailed representation of the corresponding offset surface to the input progenitor surface. The offset and the progenitor surface must have the same number of control points and exactly the same knot vectors and the degrees on u and v parametric dimensions to generate the volumetric representation. The degree of the 3rd parametric dimension w is always set to 1 and a corresponding knot vector is automatically generated.

One of the important aspects of surface offsetting is determining self and surface-to-surface intersections. Since, the offsetting distance is very small due to the constraints of composite layer generation and the main objective for offsetting is generating the volumetric representation, we believe that developing an extra step to fix intersections is not required for our problem. However,
Figure 4.1 Translation of control points to generate offset surfaces. The blue grid is the control points grid of the progenitor surface, the black grid is the control points grid of the offset surface. Tangent vectors (dark orange) and a single translation vector (cyan) are computed for each control point on the progenitor grid.

To be eligible for precise structural analysis, the error in distance between the progenitor and the offset surface should be minimum.

We used the mathematical representation as formulated in Equation 4.1 to generate the offset and thus, the layer, where $S_0$ is the progenitor surface, $d$ is the offset distance (i.e. layer thickness), $\overrightarrow{N}$ is the unit normal vector and $S$ is the offset surface.

$$S = S_0 \times (d \times \overrightarrow{N}) \quad (4.1)$$
While the surface offsetting equation looks straightforward, one should not expect to get a single unit normal vector unless a planar surface is used as the progenitor. Thus, any offsetting method should be able to sample the progenitor surface, find the unit normal vector of the sampled surface points, multiply unit normal vectors with the layer thickness to generate translation vectors and finally, translate the surface points by the translation vectors. Such a method would produce an exact offset of the progenitor surface; on the other hand, degrees, knot vectors and control points of the resultant offset surface must be estimated. Generation of matching layers (i.e. having the same degrees, knot vectors and number of control points for both progenitor and offset surfaces) without using computationally expensive refinement operations, such as knot insertion, refinement and removal, is a big challenge for generating the volumetric representation of a layer, and hence the laminate, which is going to be used for structural analysis. To overcome this challenge, we propose a fast control points estimation method for generation of matching offsets from progenitor surfaces. We also compare our method with the existing surface interpolation and approximation techniques.

Considering our approach on estimating the control points of the offset surface, the first and probably the easiest method that comes in mind is surface interpolation. Since the offset surface should have fixed degrees, knot vector and number of control points, only the coordinates of the control points ought to be found. The surface interpolation algorithm (Piegl and Tiller, 2012) takes a set of data points as its input, and the output control points have the same size of the input data points array, \( n + 1 \). That is one of big limitations of the interpolation algorithm when we fix the number of control points to the number of control points on the progenitor surface. The interpolation algorithm can also use the derivatives at the input points as data points which increases the size of the data points array to \( 2n + 1 \), skipping the derivative at the last data point. Since, we have a fixed number of control points, using derivatives causes issues when the number of control points is an even number. Therefore, we haven’t included the use of derivatives in our surface interpolation implementation.
The second approach for estimation of the control points is the surface approximation. Surface approximation method is more flexible as it is possible make use of more data (surface) points and the number of control points can still be fixed. In our implementation of the surface approximation algorithm, we precisely interpolate the corner points of the surface and fit the inner control points. This approach is also accepted to work with the majority of the applications that uses surface approximation methods (Piegl and Tiller, 2012).

Both surface interpolation and approximation methods require solution of a system of linear equations to compute the control points of the output spline geometry. The required solution can be obtained by LU Decomposition method. However, it is a computationally expensive algorithm with the time complexity of $O(\frac{2}{3} n^3)$. There are numerous attempts to improve the efficiency of these surface fitting methods using statistical optimization (Liu and Wang, 2012), (Liu et al., 2016), (Zheng et al., 2012) and iterative methods (Lin, 2012), (Yoshihara et al., 2012) which would definitely improve the runtime speed of the surface fitting operation. However, preserving the knot vectors and number of control points are main constraints and the limitations of the NURBS volume generation for modeling laminated composites. To satisfy these conditions, we propose a new method that would estimate the control points of the offset surface; and therefore, construct of the volumetric representation directly under the physical definition of the composite layer. Our method offsets the control points grid of the progenitor surface instead of the surface points and then applies local modifications to minimize the distance error between the progenitor and the offset surface. The proposed method is illustrated on Figure 4.1.

To generate the offset surface, we first get the control points grid of the progenitor surface. A $6 \times 6$ control points grid is illustrated in Figure 4.1(a) as the progenitor surface example. Then, for each control point, we find the unit tangent vectors of the control points corresponding to the combination of positive and negative directions of all parametric dimensions. To find the tangents, we use the next control point closest on the working direction for the current control point in consideration. On the edge and corner cases, we extrapolate the control points grid in the required directions to find the corresponding tangent vector. As a result, we get 4 tangent
vectors as illustrated in Figure 4.1(b) with the dark orange colored arrows. Then, we find the unit normal vectors at each control point using the cross-product of the adjacent unit tangent vectors. This will result in 4 unit normal vectors for each control point. To generate a single unit translation vector for each control point, we compute the vector mean of the 4 unit normal vectors. This would definitely result in different unit translation vector for each control point as illustrated in Figure 4.1(c); and therefore, it would be possible to closely approximate the offset surface without using computationally expensive methods. Finally, to find the offset surface, we do a scalar product of the lamina thickness value by the unit translation vector and then, translate the control points using their corresponding translation vector. The progenitor control points grid, vectors and the offset control points are illustrated in Figure 4.1(d). This method can also accept negative lamina thickness values for flexible generation of the geometries since a negative value simply means offsetting to the reverse direction.

As opposed to the other methods discussed previously, our proposed method is independent of surface degree and size of the control points grid matrix. It takes a list of points and returns a different list of points; therefore, its algorithmic time and space complexity can be considered as linear which is superior to the compared interpolation and approximation methods.

Global interpolation and approximation methods is limited to non-rational geometries. To work with rational surfaces, implementation of local methods is required (Piegl and Tiller, 2012). These methods first try to estimate the Bézier surface patches and then using the knot removal algorithm, they remove the excess knots corresponding to the adjacent surface patches to construct the approximate rational surface. As a workaround for our method, we divide the each unit translation vector by the weight of the corresponding control point and assign the same weight to the generated offset surface control point. Considering the computational complexity of the local methods, our proposed workaround can estimate a offset rational surface in the thickness limits of layer generation.
4.3.2 Construction of laminates from layers

To construct the composite laminates in desired thickness, we follow the layer-by-layer modeling approach for composite laminate generation (Bingol et al., 2019, 2017). Layer-by-layer modeling method starts with a rational surface and offsets it to generate a 3-dimensional solid B-Rep representation of the layer. Using one of the top or the bottom faces of the initial layer, the next adjacent layer is generated and is offset to generate its 3-dimensional solid model. As a result, each layer is constructed separately allowing finite element software packages to assign different boundary conditions for each layer and adjacent face.

Our method replaces the B-Rep model with NURBS volumes, which are 3-dimensional geometries on Euclidean and parametric space, and they can be generated using the evaluation methods discussed in (Piegl and Tiller, 2012) with minor modification using the method We use the method discussed in Section 4.3.1. After obtaining the desired laminate thickness and number of layers, all individual NURBS volumes representing the layers combined on the 3rd parametric dimension w to generate the final NURBS volume representation for the composite laminate. Hence, the final NURBS volume can be used as an input for structural analysis using IGA, which will be discussed in Section 4.4.

4.3.3 Validation of the methods

In this section, we compare the accuracy offset surface reconstruction methods: our proposed method, interpolation and least squares approximation. Thus, Hausdorff and the minimum distance between the progenitor and the offset surface is measured (Hanniel et al., 2012). In all the use cases, we generate a single lamina using the 6×6 control points grid illustrated in Figure 4.1 as the progenitor surface and we used d = 1.0 for the layer thickness, i.e. the offsetting distance. For least squares approximation, a 100×100 surface point sampling is used for estimating the 6×6 control points grid. After obtaining the offset surface control points grid, we re-evaluated the surface and then compared the distance between the progenitor and the offset surface points. Table 4.1 displays the result of the comparison.
Table 4.1 Comparing the accuracy of the surface reconstruction methods; our proposed method, interpolation and least squares (LS) approximation, using Hausdorff and minimum distance metrics.

<table>
<thead>
<tr>
<th>Method</th>
<th>Minimum Dist.</th>
<th>Hausdorff Dist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Method</td>
<td>0.89</td>
<td>1.05</td>
</tr>
<tr>
<td>Interpolation</td>
<td>0.31</td>
<td>1.99</td>
</tr>
<tr>
<td>LS Approximation</td>
<td>0.53</td>
<td>1.66</td>
</tr>
</tbody>
</table>

We observed that our proposed method generated the most accurate offset surface in compared to interpolation and offsetting for the specified control points grid in Figure 4.1.

4.4 Isogeometric analysis of laminated composites

Isogeometric Analysis (IGA) is an analysis model introduced by Hudges and his co-workers (Hughes et al., 2005) and it has been developed for more than a decade by researchers all over the world. IGA is designed over the traditional finite element analysis models (Cottrell et al., 2009) and software (Rypl and Patzák, 2012; Agrawal and Gautam, 2019) with using the NURBS basis (shape) functions. This allows IGA to operate on the exact geometry defined by the NURBS equations (Piegl and Tiller, 2012) which provides several advantages over the traditional finite element model:

- The geometric approximation errors are minimized
- There is no need for remeshing for the analysis
- Problems can be solved more precisely, such as interlaminar and intralaminar stresses for composite laminates

There are several software (Nguyen et al., 2015; Vuong et al., 2010; Dalcin et al., 2016; Kamensky and Bazilevs, 2019) developed in various programming languages to operate on several platforms for running isogeometric analysis. We found that working with these software involves some extra requirements, such as package compilation or experience in the base software. Most of them have
limited examples and documentation. We believe that users need not to be focused on the internals of the software but only their research. Therefore, we developed a user-friendly, modular, object-oriented Isogeometric Analysis library in Python, named \texttt{gIGA}, which is built over our NURBS library, \texttt{NURBS-Python} (Bingol and Krishnamurthy, 2019), for structural analysis of the composite laminates. Similar to \texttt{NURBS-Python} modeling library, \texttt{gIGA} is a pure Python library, i.e. does not require any compilation steps. However, the current version does not contain pure Python matrix solvers (for solving the stiffness equation, $Kd = f$), instead it implements exact (LUP decomposition) and iterative (conjugate gradient with preconditioning) matrix solvers of well-known and widely-used 3rd party scientific Python packages, i.e. SciPy and NumPy (Jones et al., 2001).

We start the isogeometric analysis with a simple proof of concept for \texttt{gIGA} by running cantilever bending analysis on a 3-dimensional beam, and then on a bowl-shaped plate. Finally, we discuss the effect of layer generation methods on the isogeometric analysis.
4.4.1 Cantilever bending of a 3-dimensional beam

The 3-dimensional beam to be analyzed has the dimensions of $15 \times 1 \times 2$. The degrees on the $u$, $v$ and $w$ parametric directions are 3, 3 and 1, respectively. The corresponding uniform knot vectors are generated automatically via NURBS-Python. We took modulus of elasticity $E = 10 \text{ GPa}$, Poisson’s ratio $\nu = 0.3$ and a uniform load vector $f = 1.0 \text{ N/mm}^3$ on the $-z$ direction as the analysis parameters to find the displacement after bending. The initial and the final geometry are displayed in Figure 4.2.

To prove the results, we compared the theoretical displacement with a displacement convergence analysis of the computational results. To find the theoretical displacement, we used the equation of the uniform-loaded cantilever beams as stated in Equation 4.2, where $\delta_B$ is the theoretical deflection at the free end, $q$ is the uniform load on the beam, $E$ is the modulus of elasticity, $I$ is the moment of inertia, $h$ is the height of the beam and $w$ is the width of the beam.

\[
\delta_B = \frac{qL^4}{8EI} \quad (4.2a)
\]
\[
I = \frac{h^3w}{12} \quad (4.2b)
\]
Figure 4.3 shows the displacement converge analysis as number of elements vs. displacement in $mm$ with elements ranging from 5 to 320. The number of elements are increased using h-refinement method, i.e. knot insertion on $u$ and $w$ parametric directions.

Using Equation 4.2, we found that the theoretical deflection of the beam at the free end is $\delta_B = 1.90 mm$. From Figure 4.3, we observe that it is possible to achieve the theoretical deflection value using 80 elements. Figure 4.4 shows the displacement gradient on the control points of the 3-dimensional beam and it supports the analysis results.

4.4.2 Cantilever bending of a 3-dimensional doubly-curved plate

The doubly-curved NURBS surface, which is obtained from NURBS-Python examples repository\(^1\), is used as the mold to generate the layer. For this example, a laminate with a thickness of $d = 0.1 mm$ is generated via our proposed method explained in Section 4.3.1. The material properties used for the analysis are follows: Modulus of elasticity $E = 210 \, GPa$, Poisson’s ratio $\nu = 0.3$. Similar to the previous example, a uniform load vector $f = 1.0 \, N/mm^3$ on the $-z$ direc-

\(^1\)https://github.com/orbingol/NURBS-Python_Examples
Figure 4.5 Cantilever bending of the bowl-shaped plate. Green is the initial geometry and blue is the final geometry after deformation.

tion is applied to the plate. Figure 4.5 illustrates the initial and the final geometries of the plate. Figure 4.6 displays the displacement of the control points at the free end of the doubly-curved plate.

It is not possible to compute a theoretical value for the displacement of the free-end of the doubly-curved plate. However, considering the results on Figure 4.6, we observe a uniform deflection on the free end of the plate, which is the expected result.

For this geometry, we also tested the layer generation methods via surface interpolation and approximation methods and we were unable to solve for the displacements. This issue was also observed by Brakhage and Lamby (2008), supporting our results. We explain this issue in detail in Section 4.4.3.
4.4.3 Effect of layer generation methods on IGA

Previous research (Cohen et al., 2010; Xu et al., 2011, 2013a) show that isogeometric analysis is heavily dependent on the parameterization of the NURBS domain by means of knot vector refinement and position of the control points in the volume and, traditional reconstruction methods, such as interpolation and approximation, might not be enough on their own (Brakhage and Lamby, 2008). In this section, we compare the control points grid generated via our proposed method, surface interpolation and surface approximation methods.

We observed that the control points grid of the offset surface generated by surface interpolation and approximation methods are skewed with respect to the control points grid of the progenitor surface. On the other hand, our proposed method generates a control grid similar to the progenitor surface, as expected due to the way the method offsets the progenitor surface to create the layer. We discovered that the control points grid generated via surface interpolation and approximation methods are not convenient for isogeometric analysis and subsequently, the matrix solvers failed even for the analysis of a single layer. These results are also coherent with the previous research related to generating geometries for isogeometric analysis.
4.5 Conclusion

In this paper, we proposed a new method for generating 3-dimensional NURBS volumes from surfaces for structural analysis of the composite laminates. We followed the composite manufacturing process during generation of the 3-dimensional models as proposed in our previous study (Bingol et al., 2019). We also introduced our new isogeometric analysis library, gIGA, which is built on our NURBS library, NURBS-Python (Bingol and Krishnamurthy, 2019).

For the layer generation, we compared 2 existing surface reconstruction methods, interpolation and approximation, with our proposed method. We found that our proposed method could generate more suitable geometries for isogeometric analysis.

4.6 References


CHAPTER 5. SUMMARY, DISCUSSION AND FUTURE WORK

5.1 Summary and Discussion

This study focuses on development of an automated composite laminate modeling and structural analysis software framework. The software framework is designed to generate 3-dimensional models of the composite laminates with or without defects and structural inclusions, such as stiffeners. The 3-dimensional models of the composite laminates are generated by following the laminate manufacturing process, referred as layer-by-layer modeling. After generation of the 3-dimensional model with desired properties, the structural analysis can be run via finite element software packages. The outputs of the framework can be used for virtual manufacturing, testing and estimating residual life of the composite structures.

Chapter 2 discusses the 1st version of the framework (Bingol et al., 2019) developed using commercial off-the-shelf (COTS) software, i.e. 3D ACIS Modeler (Spatial Corporation, 2017) and SIMULIA Abaqus (Dassault Systèmes Abaqus SIMULIA, 2017). This version of the framework can generate 3-dimensional models of the composite structures via layer-by-layer modeling approach. It can incorporate defects, i.e. delaminations and fiber/matrix breakage, in addition to the structural elements, such as stiffeners. This version of the framework has an innovative feature that computes the boundary conditions by identifying the faces of the 3-dimensional model to promote the automation process intended. The face identification process gets complicated to achieve manually when multiple defects and structural elements are introduced to the model, as these operations can generate a number of faces that are split from the initial single face. Such process gets extremely tedious if it is done manually on the user interface and prone to missing boundary condition assignments to some faces leading to wrong results. After generation of the 3-dimensional model and computation of the boundary conditions, the data passed to the finite element software SIMULIA Abaqus for mechanical analysis.
The initial version of the framework is developed using C++ and Python programming languages due to having only C++ programming interface for 3D ACIS Modeler and preferred programming language being Python as SIMULIA Abaqus provides a fully-featured Python programming interface. Choosing a scripted language like Python as the preferred programming language throughout the framework also reduces the learning steps and provides an option to run quick tests without any complicated compilation steps. SWIG\(^1\) is used to wrap C++ code into Python modules (Cottom, 2003) to maintain seamless integration between modeling and analysis software.

During the development of the initial framework, some limitations of the COTS software used required some workarounds by means of software engineering, i.e. bugs observed in SWIG. In addition, geometric compatibility problems are observed within 3D ACIS Modeler in conversion of faces represented via B-Rep to NURBS surfaces, as most operations are done on the NURBS component of the framework. Therefore, a new and improved framework is designed to only operate with NURBS geometries, and isogeometric analysis method, which is designed to integrate NURBS with finite element analysis, replaced the traditional finite element method used by SIMULA Abaqus.

Chapter 3 and Chapter 4 together describe the new and improved automated framework based on the NURBS geometries for the modeling component and isogeometric analysis for the analysis component. Chapter 3 focuses on the open-source NURBS library, NURBS-Python (Bingol and Krishnamurthy, 2019). Chapter 4 focuses on the composite laminate modeling extension, based on NURBS-Python and, the structural analysis library, gIGA, based on isogeometric analysis method (Hughes et al., 2005). Both NURBS-Python and gIGA are self-contained, object-oriented, pure Python libraries. More details, links and examples regarding to these libraries can be found on the Appendix.

Chapter 4 also discusses a new method for generating NURBS volumes from surfaces to generate layers that build the composite laminate. Since isogeometric analysis method is used for the structural analysis of the laminates, generating a analysis-aware model, a term coined by Cohen et al. (2010), is a very crucial step for obtaining correct results from the isogeometric analysis.

\(^1\)http://swig.org/
To investigate further, 3 surface reconstruction methods are compared in the context of generating composite layers: surface interpolation, surface approximation and a new proposed method, control point translation. It is observed that it is possible to obtain a result using the proposed method; whereas, the other 2 surface reconstruction methods fail during the isogeometric analysis stage.

In conclusion, a framework for modeling and structural analysis of composite laminates has been developed and its software components are publicly released on GitHub as open-source projects. A new approach for layer-by-layer modeling of the composite laminates has been proposed and its applicability is proved by using state-of-the-art modeling and analysis tools. This approach has been extended to directly operate with NURBS geometries and its analysis counterpart, isogeometric analysis method.

5.2 Future Work

Future work on the automated framework will involve extension of the NURBS library to handle defects like delaminations. These type of defects are incorporated via face splitting which can be represented using trimmed NURBS. Although NURBS-Python supports surface trimming via its traditional definition, i.e. closed NURBS curves defined in the parametric domain of the surface are attached to the data structure of the surface and the surface is tessellated without the regions defined by the closed curves, IGA cannot handle these type of surfaces directly. Truncated Hierarchical B-Splines or THB-Splines are an alternative, which can be used to define the trimmed geometries to perform IGA. NURBS-Python will be extended to support THB-Splines; and therefore, gIGA can be used to analyze delaminated composite structures.

gIGA is an object-oriented, pure Python isogeometric analysis library at its early beta stages of development. Another plan involves extending gIGA to support multiple geometries with constitutive models suitable for analysis of composite laminates. In addition, gIGA will be extended to take advantage of modern graphical processing units (GPUs) to perform parallel data processing during numerical integration and matrix assembly stages. Following the GPU extension of numer-
ical integration and matrix assembly, a GPU-based high-performance linear equation solver will be integrated with gIGA.

5.3 References


APPENDIX. PUBLISHED SOFTWARE

Although software is a very important part of the today’s scientific advancements, most of the time, proper software engineering principles are overlooked during the software development due to researchers' focus on developing a working code. It is a fact that taking such an approach in software development causes a disconnect between scientific developments and leaves a trail of impossible-to-maintain software which leads to unnecessary new development of existing software. This leads to a huge loss of time and budget, with increased effort.

A modern scientific software should

- Follow the software engineering principles and implements the best practices
- Provide an easy way to operate for researchers coming from different fields
- Allow extensions of the existing code base without blocking the new features
- Have a good documentation with working and clearly explained examples
- Have unit, function or system tests covering the majority of the code base
- Integrate with the modern DevOps tools for automation and integration

and, more importantly, a modern scientific software should be free and open-source, allowing research and development for people around the world. It shouldn’t stay behind the paywalls or unaffordable license fees , and therefore; it should follow the open-source software publication principles.

As a result of this study, several modeling and analysis libraries and modules are developed and (to be) released as free and open-source on GitHub. The following list shows the URLs of the software repositories released as free and open-source:

- geomdl: https://github.com/orbingol/NURBS-Python
- geomdl-examples: https://github.com/orbingol/NURBS-Python_Examples
- geomdl-cli: https://github.com/orbingol/geomdl-cli
• geomdl-shapes: https://github.com/orbingol/geomdl-shapes
• ACIS-Python3: https://github.com/orbingol/ACIS-Python3
• de-la-mo: https://github.com/idealab-isu/de-la-mo
• rwsat: https://github.com/orbingol/rwsat
• rw3dm: https://github.com/orbingol/rw3dm

The following list shows the names of the software to be released as open-source:

• geomdl-evaluators: Accelerated NURBS evaluation for geomdl
• gIGA: Isogeometric Analysis library for structural analysis