2010-03-10

Methods to Streamline Laminate Composite Design, Analysis, and Optimization

Ammon Ikaika No Kapono Hepworth
Brigham Young University - Provo

Methods to Streamline Laminate Composite Design, Analysis, and Optimization

Ammon Hepworth

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

C. Greg Jensen, Chair
Christopher Mattson
David T. Fullwood

Department of Mechanical Engineering

Brigham Young University

April 2010

ABSTRACT

Methods to Streamline Laminate Composite

Design, Analysis and Optimization

Ammon Hepworth

Department of Mechanical Engineering

Master of Science

Advanced composite materials have seen major market growth in recent years due to their high strength and low weight properties. These materials are often made using a process that creates a composite laminate by stacking several composite layers together. However, the design, analysis and optimization of laminate composite materials are often a labor intensive process when done manually. This thesis discusses CAD independent algorithms that are integrated into commercial CAD tools to streamline these processes. Methods have been developed to automatically create 3D ply geometry for a laminate composite lay-up, streamline the creation of a laminate composite finite element model and optimize the composite lay-up for a multi-layered laminate composite part.

Integrating a CAD independent geometry kernel into the NX laminate composite design automation application significantly improves the run time of that application. In addition, the automated composite finite element tool creates laminate composite finite element models that are more detailed than those made with zone based methods. This tool will save engineers, who are making laminate composite finite element models manually, dozens of hours of work per model. The automated composite finite element tool can also be integrated into an optimization framework, used in conjunction with a method to automatically apply boundary conditions, to create an effective optimization of a laminate composite part.

Keywords: laminate composite design, analysis, optimization, automation

ACKNOWLEDGEMENTS

I would like to thank Dr. C. Greg Jensen from Brigham Young University as well as Ryan Cox and James Roach from Pratt & Whitney. Each of them has been invaluable to the success of this research and subsequent thesis. They have been a great mentors as well as friends. Additional thanks to Dr. Mattson and Dr. Fullwood for their input and willingness to be on my committee. A special thanks to Pratt & Whitney for funding this research. Also thanks to my wife Monica for her support and love through all of this.

TABLE OF CONTENTS

LIST OF TABLES ..... **vii**

LIST OF FIGURES ..... **ix**

1 Introduction **1**

1.1 Problem Overview 1

1.2 Thesis Objective 3

1.3 Problem Delimitation 4

1.4 Thesis Organization 4

2 BACKGROUND **7**

2.1 Laminate Composite Materials 7

2.2 Composite Automation Programs 8

2.3 CAD APIs and the CAD Independent Approach 10

2.4 Geometry Mathematics and NURBS 11

3 Method **15**

3.1 Input Data from a User Familiar Source 16

3.2 CAD Independent Algorithms 17

3.2.1 Geometry Creation 17

3.2.2 Generation of Finite Element Model 18

3.3 Output Data into a Useable Source 21

3.4 Optimization 22

3.4.1 Parametric Boundary Conditions 22

3.4.2 Optimization Algorithm 23

3.4.3 Fitness Factor 24

4 Implementation **25**

4.1 Input Data from a User Familiar Source 26
4.2 CAD Independent Algorithms 29
4.2.1 Automatic Geometry Creation 29
4.2.2 Automatic Generation of Finite Element Model 31
4.3 Output Data into a Useable Source 35
4.4 Optimization 41
4.4.1 Automatic Boundary Conditions for the Finite Element Model 41
4.4.2 Optimization using Isight 43
5 Results 45
5.1 Composite Design Automation 45
5.2 Composite Analysis Automation 49
5.3 Composite Analysis Optimization 52
6 Conclusions 55
6.1 Recommendations 57
References 59

LIST OF TABLES

Table 4-1: An example of a ply table.....28

Table 5-1: The ply property table from MS Excel.....47

Table 5-2: Test results from speed comparison of NX, GSNLIB and CAA RADE 48

Table 5-3: Test results of the speed of the automated composite finite element tool.....52

LIST OF FIGURES

Figure 3-1: General approach to streamline design and analysis 16

Figure 3-2: A 2D representation of the normal vectors taken along the mid-surface.....20

Figure 3-3: Finding the thickness at each element center.....21

Figure 4-1: A representation of the approach of the automation program 26

Figure 4-2: Offset from outer surfaces and trim 30

Figure 4-3: A side view of a generic NX ply geometry creation.....31

Figure 4-4: Laminate FEA model of a vertical stabilizer 34

Figure 4-5: An ANSYS shell model of a cantilever beam.....40

Figure 4-6: Isight process flow for the laminate composite optimization 44

Figure 5-1: A test case showing simple ply geometry created in NX 44

Figure 5-2: Vertical stabilizer loaded in NX.....45

Figure 5-3: Vertical stabilizer ply lay-up.....45

Figure 5-4: NX Wing, FE shell mesh and FE shell mesh with lamina thicknesses shown ...50

Figure 5-5: Complex NX Surface, FE shell mesh, FE mesh with thicknesses shown 51

Figure 5-6: A graph of the genetic algorithm's progression.....53

1 INTRODUCTION

Advanced composite materials have seen a major market growth in recent years and are becoming more widespread in industry today [28]. This is largely due to the fact that these materials have such a high strength to weight ratio and that their material properties can be tailored to meet specific design criteria. Despite the benefits, it is often a "laborious and painstaking process" to design, analyze and optimize these materials manually [19]. Because of this, there has been an effort to develop automation methods to streamline the design, analysis and optimize of composite materials. This thesis will discuss methods to streamline laminate composite design, analysis and optimization using CAD-independent algorithms in a CAD centric way.

1.1 Problem Overview

One method of manufacturing advanced composite materials is by stacking several fiber dense layers and bonding them together. Each layer is made of high-modulus fibers that are imbedded in a lower modulus resin. Composite materials created using this stacking process are called laminate composites. Designing a composite laminate is time consuming for two major reasons. First, since the material properties of a part are largely determined by the fiber orientation of each layer in the laminate, there are many design combinations to explore. Second, when designing geometrically complex parts, the geometry of each layer is difficult to predict and time consuming to create [9, 25].

Due to the large number of design parameters, the design of composite parts naturally calls for optimization algorithms [4]. Utilizing computer integrated optimization algorithms allow designers to more efficiently explore the design space and find the optimum design of a laminate composite part. However, to make optimization possible, a fast and accurate analysis method must be employed. Finite element analysis is the preferred analysis method for geometrically complex laminate composite parts; but it is time consuming to manually build composite models having several layers with unique fiber orientation angles [9]. The finite element analysis process must be streamlined to make using optimization practical.

The time consumption in creating the composite layer geometry may be overcome through the use of an automated method to create the geometry and allow users to visualize and interact with it. The traditional method to automate and streamline the modeling phase of the design process (called the CAD centric approach) is through the use of a commercial CAD application programming interface (API). A CAD API allows users to programmatically create custom methods in existing commercial CAD applications. Creating a custom application using the CAD centric approach provides the following advantages:

- Provides for geometry visualization
- Easy for companies to adopt with a pre-existing CAD system in place
- Simplifies CAD data storage and transfer

Regardless of the advantages, there are two main weaknesses to the CAD centric approach. The first weakness is that CAD APIs are often computationally slow when compared to CAD independent algorithms because they often exist a level or two above the geometry kernel. This makes them impractical to use for computationally intense problems such as the creation of laminate composite layer geometry. The second weakness is that the CAD-centric

approach keeps the application on a single CAD system thus severely limiting its use within large companies that have and use multiple CAD platforms.

1.2 Thesis Objective

In 2003, a method was developed by Travis Astle that overcame the weaknesses of the CAD-centric approach while still incorporating its benefits. He applied CAD independent algorithms to the creation of flank milling data and integrated it into NX using NX's API. Simply stated, the method takes geometry from a CAD package, translates it into mathematical geometry data, uses CAD independent algorithms to create new geometry and perform math intensive geometric operations of the data, and then translates the new data back into the CAD package [2]. This method allowed the application to be portable enough to be integrated into several CAD applications and still provided simple visualization, company adaptation and data storage/transfer.

The objective of this thesis is to present a method that applies CAD independent algorithms to streamline the design, analysis and optimization of laminate composites. Doing this incorporates the benefits, discovered by Astle, of being fast and portable while still keeping the benefits of simplified visualization, company adaptation and data storage/transfer. The method specifically does the following:

- Automatically create 3D geometry for individual plies in a laminate composite lay-up for a composite part with complex geometry
- Streamline the creation of detailed laminate composite finite element models
- Optimize the composite lay-up for a composite part made of several layers

This method has been proved through the implementation of the method into computer programs that accomplish the above mentioned tasks.

1.3 Problem Delimitation

In writing software to accomplish the above mentioned automation tasks, it must be understood that the author has not produced a commercially available application. The purpose of writing the computer programs is only to show that the methods are implemented in a practical way. In addition, it is not the intent of the computer programs to be able to create a composite lay-up on any geometric object, but only those that can be represented by two opposing NURBS surfaces in Siemens NX and Dassault Systemes CATIA. The program that streamlines the creation of the composite finite element model is implemented into ANSYS, MSC NASTRAN and LS-DYNA only. No other analysis packages or mesh types are examined in this research effort. In regards to optimization, only a few test cases are examined and each is integrated into SIMULIA's Isight optimization framework. The purpose of this thesis is to present a general method to optimize laminate composites and not to prove necessarily that one algorithm is better than any other. This said, Isight's genetic algorithm has worked well in this study and so it will be the only optimization algorithm employed here.

1.4 Thesis Organization

A large portion of the content of this thesis comes from two papers published by the author in the Computer-Aided Design & Applications Journal in 2009 and 2010 [12, 13]. The thesis is organized into six chapters. Chapter 2 is a literature review that introduces the reader to the most relevant literature in relation to this thesis. This will include a brief introduction on

composite materials, a discussion about existing composite automation methods, CAD application programming interfaces and NURBS mathematics. The third chapter discusses the general methods used to streamline laminate composite design, analysis and automation. The fourth chapter discusses the implementation of those methods into existing commercial CAD and FEA packages. Chapter 5 shows the results of the composite design and analysis automation tools and the results from an optimization that utilizes the analysis automation tool. The sixth and concluding chapter discusses the conclusions and future work of this research.

2 BACKGROUND

The intent of this chapter is to give the reader the background necessary to understand the significance of this research. First there will be a general discussion of laminate composite materials in section 2.1 to give the reader a basic understanding of the design process used to optimize a laminate composite material for a given application. If the reader is completely unfamiliar with laminate composites the author recommends the following references [17, 8, and 14]. Section 2.2 is a review of the most significant research in the area of composite automation tool development. The purpose of this section is to show what has already been done in this area and to note what they are lacking. Section 2.3 is a discussion of CAD API programming to allow the reader insight into what others have accomplished using CAD APIs. Finally, in section 2.4, there is an introduction on NURBS surface mathematics and a brief discussion on the advantages of using NURBS mathematics in this research.

2.1 Laminate Composite Materials

Generally speaking, a composite material is any heterogeneous material made-up of more than one material. However, composite materials most commonly refer to a material having strong fibers imbedded into a weaker matrix material. The fibers serve to carry the load, while the matrix distributes the load to the fibers [8]. Advanced composites refer to composite materials containing long fibers and resins with mechanically superior properties and have an extremely high strength to weight ratio [25]. Traditionally, advanced composite materials are

manufactured by stacking several fiber dense layers and bonding them together to form a laminate. This process allows the properties of a composite laminate to be tailored to a specific loading environment by orienting the layers of unidirectional material to satisfy the loading requirements [17].

2.2 Composite Automation Programs

A number of automated laminate composite design tools have been developed and will be discussed herein, but there are two points to note before discussing each tool in detail. The first point to note is although each of these applications presents viable methods to automate design, analysis and optimization of laminate composites, all of them use a zone based approach. This means that the part is divided into zones of constant thickness allowing for simplified analysis and laminate definition. Although this simplification suffices for relatively smooth geometry with large regions of constant thickness, this does not provide sufficient accuracy in parts with rapidly changing thicknesses. In addition, the zone based approach does not create 3-dimensional ply geometry from the part geometry alone. All ply geometry is either created based on the zone boundaries or from user input.

AUTOLAY is a design tool developed in 1999 to automate laminate composite design, analysis and manufacturing using a GUI-based approach. This tool assists the user in ply modeling, substructure design and the generation of ply stacking sequences, engineering and manufacturing drawings. The tool has the user identify two surfaces that represent the engineering and manufacturing geometry. On the engineering surface, a user defines areas of constant thickness (zones) to be used by the tool to automatically determine the number of plies and the stacking sequence in those areas. The stacking sequence of the zones is determined

using design rules and analytical solutions which find the delamination and buckling strengths. The ply shapes are determined either automatically from the zone definitions or interactively by the user. Finally, engineering and manufacturing drawings are automatically produced using general design practices. The resulting benefit of AUTOLAY is that composite designers and manufacturers end up with a substantial cost and time savings [19].

Another method to automate composite design was developed by Vasey-Glandon et al. They patented a computer program called the Parametric Composite Knowledge System (PACKS) that generates an optimized 3-D ply definition for a laminate composite part. The program divides the plies into zones and uses FEA analysis software as well as predetermined design rules to optimize these zones to perform better in manufacturing [27]. PACKS reduced the cost of many aircraft composite development programs by over 60% and was licensed to Unigraphics Solutions, Inc (now Siemens) for commercial use in 2001 [11].

Another composites optimization computer program, developed by Menayo et al., optimizes a ply lay-up for a composite part to form a preliminary ply model. This is done by first running a stress analysis program (called ARPA) to obtain a stacking sequence for divided regions of the part and then automatically generating a ply stacking table that organizes the plies into zones. The stacking table is then optimized using design stacking rules and manufacturing constraints. Next, a ply drop-off distribution is performed to determine which plies end-up in a given zone and they are ordered according to design rules [15].

Siemen's NX has an integrated, zone-based laminate composites application that allows for ply modeling, draping, and optimization. In ply modeling, it allows one to associate ply lay-ups with NX surfaces, but it does not provide for 3D visualization of actual ply geometry.

Draping predicts ply orientations along doubly-curved surfaces. The optimization piece makes it possible to optimize laminate properties to meet specific design criteria [23].

These tools represent a good foundation of laminate composite automation, however all of them use the zone based approach. The method presented in this thesis will divert from this approach by automatically creating design and analysis data without having to define zones. The method automatically creates ply geometry and finite elements based on a ply table definition and the part geometry input by the user. No zone definitions are necessary.

2.3 CAD APIs and the CAD Independent Approach

Application programming interfaces (API) are used to interface with commercial software applications programmatically (as opposed to interactively). Most commercial CAD systems have APIs and they are often used in industry to automate design processes. The method of automating a design process using only the CAD system's API is called the CAD centric approach. There have been several theses written explaining methods that streamline design processes using APIs. A few recent theses that discussed using CAD APIs to generate custom CAD applications include the following: Delap used a CAD API to create a parametric model of a gas turbine flow path that can be optimized based on user rules and objectives [5]. Elliott programmatically created a reusable parametric model of a complex surface including over 3000 parameters also using a CAD API [6]. Scott also utilized a CAD API to streamline the CAD assembly and component level design process [21].

Automating a design process using the CAD centric approach provides several advantages: it provides a simple way to visualize geometry, it is simple to adopt for companies with existing CAD system in place and it simplifies data storage and transfer. However, there

are two major weaknesses to the CAD centric approach. The first weakness is that CAD APIs are often slow in performing geometrical computation when compared with CAD independent algorithms because they often exist at a level above the geometry kernel. When an application requires large amounts of geometry computation, the CAD-centric approach becomes impractical. The second major weakness is that the CAD-centric approach limits the use of the application to a single CAD package. Since there are many CAD packages in circulation today, limiting an application to a single CAD package may prevent extensive use among designers.

As stated in the introduction Travis Astle developed a method that overcame the weaknesses of the CAD-centric approach while still incorporating its benefits. Recall that he applied CAD independent algorithms to the creation of flank milling data and integrated it into NX using NX's API. In summary, the method takes geometry from a CAD package, translates it into mathematical geometry data, uses CAD independent algorithms to create new geometry, and translates the new data back into the CAD package [2]. This method allowed the application to be portable enough to be integrated into multiple CAD applications and kept the benefits of simple visualization, company adaptation and data storage/transfer.

2.4 Geometry Mathematics and NURBS

Non uniform rational B-splines (NURBS) are the mathematical surface representation used to represent geometry in this research. NURBS are a common parametric curve and surface representation used within many of the leading commercial CAD systems. NURBS mathematically represent a surface using a given number of control points that form a bidirectional control net. These control points each influence the topology of surface defined by the B-spline basis function. Each control point is associated with a weight that adjusts the

amount of influence that each control point has on the surface. NURBS employs a knot vector in each parametric direction u and v that also affect the influence of the control points. A NURBS surface of degree p in the u direction and q in the v direction is represented by the following equation:

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \qquad 0 \le u,v \le 1 \tag{2-1}$$

The $\{P_{i,j}\}$ are the control points, the $\{w_{i,j}\}$ are the weights, n and m represent the number of control points in the u and v directions respectively, and the $\{N_{j,q}(u)\}$ and $\{N_{j,q}(v)\}$ are the NURBS basis functions which are defined on the knot vectors:

$$U = \{\underbrace{0,\ldots,0}_{p+1}, u_{p+1},\ldots,u_{r-p-1},\underbrace{1,\ldots,1}_{p+1}\} \qquad u_i \le u_{i+1} \tag{2-2}$$

$$V = \{\underbrace{0,\ldots,0}_{q+1}, v_{q+1},\ldots,v_{s-q-1},\underbrace{1,\ldots,1}_{q+1}\} \qquad v_i \le v_{i+1} \tag{2-3}$$

r and s are defined by $r = n + p + 1$ and $s = n + p + 1$ [18]. To learn more about NURBS mathematics, see the following sources: [7, 18, 20, and 22].

A major advantage of using NURBS surfaces to represent geometry is that complex surfaces can be represented with relatively small amounts of data. Also, since NURBS are parametric in nature, they have the advantage of being able to quickly compute the coordinates of any point on a surface, making the parsing of a surface quick and easy [22]. In addition, many NURBS based, CAD independent algorithms have been published, including a C++ based

geometry library called GSNLIB. This library has several functions that create, manipulate and store NURBS curves and surfaces [24].

3 METHOD

This chapter discusses methods to streamline laminate composite design and analysis following these steps (see Figure 3-1 for a graphical representation):

1. Take surface geometry from a user familiar source (i.e. a commercial CAD system), translate it into the mathematical NURBS representation of that surface and store it in NURBS based, CAD independent data structures
2. Apply CAD independent algorithms to create the ply lay-up geometry and the laminate composite finite element model
3. Translate the NURBS surfaces back into a user familiar source for visualization and analysis

Methods will also be discussed to streamline laminate composite optimization and include the following:

- Applying parametric boundary conditions
- The optimization algorithm
- The fitness factor

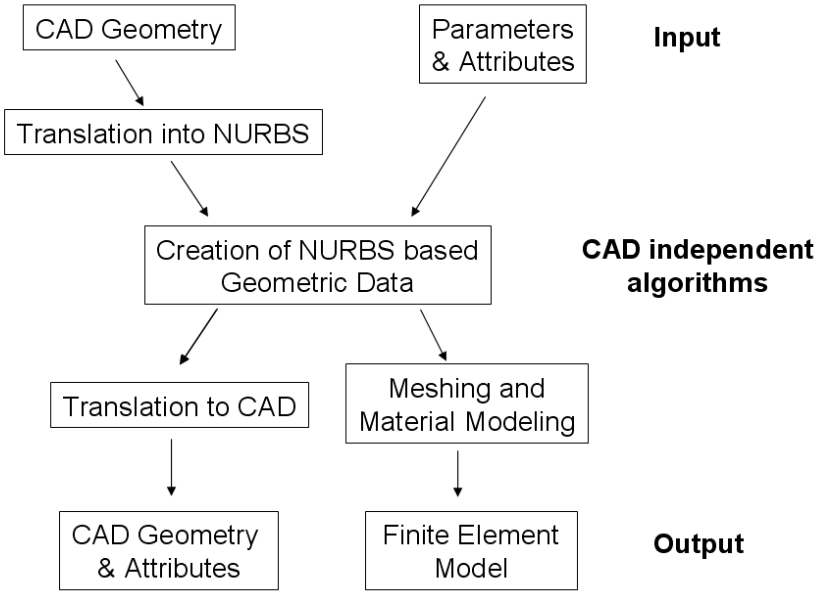

Figure 3-1: General approach to streamline design and analysis

3.1 Input Data from a User Familiar Source

A CAD API function prompts a user to select *l* number of surfaces within the CAD environment. These surfaces are translated into CAD independent form by using an API call to extract the NURBS data from the surface and storing it in CAD independent NURBS based data structures. This can be shown as follows:

$$\sum_{h=0}^{l}\sum_{i=0}^{n}\sum_{j=0}^{m} w_{h,i,j}P_{h,i,j}{}_{CADIndep} = \sum_{h=0}^{l}\sum_{i=0}^{n}\sum_{j=0}^{m} w_{h,i,j}P_{h,i,j}{}_{CAD} \tag{3-1}$$

$$\sum_{h=0}^{l} U_h{}_{CADIndep} = \sum_{h=0}^{l} U_h{}_{CAD} \tag{3-2}$$

$$\sum_{h=0}^{l} V_h{}_{CADIndep} = \sum_{h=0}^{l} V_h{}_{CAD} \tag{3-3}$$

In addition to CAD geometry data, the laminate specific parameters and attributes are read in from a spreadsheet and stored in CAD independent data structures.

3.2 CAD Independent Algorithms

Once the CAD data is brought into CAD independent data structures, CAD independent algorithms are applied to create the ply lay-up geometry based on the input parameters. This is done by using methods to offset the original surface geometry and trim it to the appropriate shape. At the same time the laminate composite finite element model is created using the method discussed herein.

3.2.1 Geometry Creation

In order to create the ply geometry, the original surfaces that came from the CAD system are offset to create new surfaces that represent the ply lay-up geometry. Some of the parameters that came from the spreadsheet are used to control the distance of each offset. The following explains a mathematical method to create an offset of a NURBS surface. The NURBS surface $\{ S(u,v) \}$ is queried for a grid of points $\{ G_{i,j} \}$ which lie on that surface with some tolerance $\{ t \}$ in *u* and *v*.

$$G_{i,j} = \sum_{i=0}^{n} \sum_{j=0}^{n} S(i/n, j/n) \tag{3-4}$$

In the above equation $n = 1/(t-1)$ and *t* is defined such that $1/(t-1)$ must be an integer.

In addition to the grid of points, the normal vectors $\{ H_{i,j} \}$ are found at the same parameter values along *u* and *v* on the surface as used in finding the grid of the points:

$$H_{i,j} = \sum_{i=0}^{n} \sum_{j=0}^{n} \hat{n}(i/n, j/n) \tag{3-5}$$

Note that the unitized surface normal vector { $\hat{n}(u,v)$ } in the above equation is found using the traditional normal vector formula:

$$\hat{n}(u,v) = \frac{\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v}}{\left\| \frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \right\|} \tag{3-6}$$

Where " $\times$ " denotes the cross product [7].

Next, an offset grid of points { $K_{i,j}$ } that lie some distance { d } from the original surface are found using this equation [29]:

$$K_{i,j} = \sum_{i=0}^{n} \sum_{j=0}^{n} G_{i,j} + H_{i,j} \cdot d \tag{3-7}$$

This grid of points { $K_{i,j}$ } is used to create the offset surface by utilizing an interpolating function to fit a NURBS surface to the point data.

Once an offset surface is created, a proprietary trimming algorithm is utilized to trim the offset surface against an opposing offset surface trimming the ply the appropriate shape.

3.2.2 Generation of Finite Element Model

The creation of the laminate finite element model starts with the two NURBS surfaces read in from the CAD system that define the outer geometry of the laminate. These are called { $S_1(u,v)$ } and { $S_2(u,v)$ }. A mid-surface { $M_{i,j}$ } of the two outer surfaces is found by first

discretizing each surface by incrementing through the $\{u,v\}$ parameters of each surface $m \times n$ times and then averaging the opposing points on each surface as shown here:

$$M_{i,j} = \sum_{i=0}^{m} \sum_{j=0}^{n} \frac{S_1(i/m, j/n) + S_2(i/m, j/n)}{2} \tag{3-8}$$

The mid-surface is then created by fitting a NURBS surface through $\{M_{i,j}\}$ by using an interpolating function that finds the NURBS surface control points and knot vector. Once the mid-surface is found, a grid of points $\{G_{i,j}\}$ parameterized by $p \times q$ is found on that surface by incrementing through the $\{u,v\}$ parameters $p \times q$ times as shown:

$$G_{i,j} = \sum_{i=0}^{p} \sum_{j=0}^{q} M(i/p, j/q) \tag{3-9}$$

To form the quadrilateral shell elements $\{E_k\}$, the grid of points $\{G_{i,j}\}$ are associated to the elements in the following way:

$$E_k = \sum_{k=0}^{(a-1)(b-1)} \begin{bmatrix} e_{1k} \\ e_{2k} \\ e_{3k} \\ e_{4k} \end{bmatrix} = \sum_{i=0}^{a-1} \sum_{j=0}^{b-1} \begin{bmatrix} G_{i,j} \\ G_{i+1,j} \\ G_{i+1,j+1} \\ G_{i,j+1} \end{bmatrix} \tag{3-10}$$

Next, by querying the points on the mid-surface halfway between each element step, the center points of each element $\{C_{i,j}\}$ are found:

$$C_{i,j} = \sum_{i=0}^{m} \sum_{j=0}^{n} M(i/2m, j/2n) \tag{3-11}$$

In addition to the center point of the elements, the positive and negative unit normal vectors are also found at each of the element centers. This is done by solving the traditional

normal vector formal (Equation 3-12) at each element center point on the mid-surface. Figure 3-2 is a representation of a composite part with the top and bottom curves representing the opposing surfaces and the middle curve representing the mid-surface. The black arrows represent the positive and negative normal vectors taken at the element centers along the mid-surface.

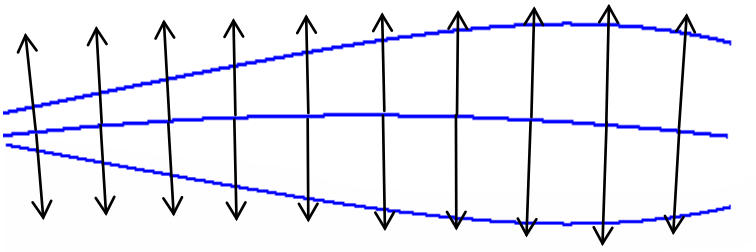

Figure 3-2: A 2D representation of the normal vectors taken along the mid-surface

The next step is to find the intersection points $\{ I_{1_{i,j}} \}$ and $\{ I_{2_{i,j}} \}$ of each outer surface and the normal vectors that pass through them.

$$I_{1_{i,j}} = \sum_{i=0}^{m} \sum_{j=0}^{n} S_1 \bigcap \hat{n}(i/2m, j/2n) \tag{3-13}$$

$$I_{2_{i,j}} = \sum_{i=0}^{m} \sum_{j=0}^{n} S_2 \bigcap \hat{n}(i/2m, j/2n) \tag{3-14}$$

The thickness at each element is then approximated to be the distance between the intersection points on the opposing surfaces (see Figure 3-3).

$$t_{i,j} = \sum_{i=0}^{m} \sum_{j=0}^{n} I_{1_{i,j}} + (I_{2_{i,j}} - I_{1_{i,j}}) \tag{3-15}$$

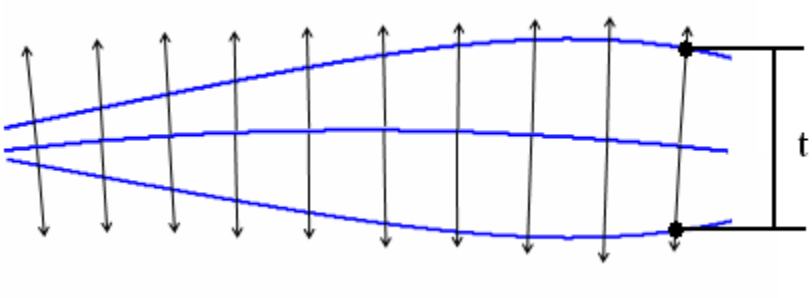

Figure 3-3: Finding the thickness at each element center

Once the element thicknesses are found the laminate definition is determined for each element by adding up the thicknesses of the lamina in the ply table. The laminas are added up starting with the one having the highest priority and going through until the element thickness is reached. Each element is then associated with the laminas that belong in that region.

3.3 Output Data into a Useable Source

Once the laminate geometry is created, the NURBS surface data that represents each lamina is translated back into the CAD system for visualization. This is done by using the CAD API to create CAD based NURBS surfaces from the control points, weights and knot vectors of the NURBS mathematical surfaces. This is shown mathematically as follows:

$$\sum_{h=0}^{l}\sum_{i=0}^{n}\sum_{j=0}^{m} w_{h,i,j} P_{h,i,j}\Big|_{CAD} = \sum_{h=0}^{l}\sum_{i=0}^{n}\sum_{j=0}^{m} w_{h,i,j} P_{h,i,j}\Big|_{CADIndep} \tag{3-16}$$

$$\sum_{h=0}^{l} U_h\Big|_{CAD} = \sum_{h=0}^{l} U_h\Big|_{CADIndep} \tag{3-17}$$

$$\sum_{h=0}^{l} V_h\Big|_{CAD} = \sum_{h=0}^{l} V_h\Big|_{CADIndep} \tag{3-18}$$

The finite element data found using the CAD independent algorithms is also translated into a form that can be read by a finite element package.

3.4 Optimization

In order to establish a working optimization the first critical component is a method to automatically generate a finite element model. This method was discussed in subsection 3.2.2. Another component needed for the optimization is a method to automatically assign boundary conditions (loads and constraints) to the finite element model. In addition, a robust optimization algorithm and fitness function needs to be established to drive the optimization to an optimum solution.

3.4.1 Parametric Boundary Conditions

Parametric boundary conditions are used to assign boundary conditions to a model that will update according to rules regardless of how many nodes are in the model. This is done by pre-determining an algorithm that assigns the boundary conditions to specific nodes based on where the conditions are with respect to the part. For example, say the desired location of a boundary condition is along the bottom end of the part. In this case the boundary conditions $\{B_i\}$ are assigned to mid-surface nodes $\{G_{i,j}\}$ of size $m \times n$ in the following way:

$$B_i = \sum_{i=0}^{m} G_{i,0} \tag{3-19}$$

However in another example, the desired location of the boundary conditions are along the left side of the part; the boundary conditions in this case are:

$$B_i = \sum_{i=0}^{m} G_{0,m*i} \tag{3-20}$$

Although these two examples above do not cover every loading case, they are enough to demonstrate the feasibility of applying parametric boundary conditions when necessary. It is left to the reader to explore other algorithms for more specific cases, if desired.

3.4.2 Optimization Algorithm

The purpose of the optimization algorithm in a laminate composite optimization is to drive the lamina properties (thickness or angle) to values that determine the best design for a specific load case. The algorithm that was chosen to do this is a genetic algorithm which is an algorithm that is based on Darwin's theory of evolution [16]. The main reason why a genetic algorithm was chosen over other gradient based methods is because gradient based methods frequently were not able to find an optimum due to the large number of design variables in a laminate composite optimization. The gradient based methods slowly changed one or two values at a time trying unsuccessfully to find a pattern. The genetic algorithm, on the other hand, was able to much better explore the design space because it moved randomly in all directions until it evolved into an optimum. My observations were similar to what the literature says in that genetic algorithms are not guaranteed to find a global optimum, but they will generally find an "acceptably good" answer "acceptably quickly" [3]. This is why a genetic algorithm was chosen to be used in the optimization method of this research.

3.4.3 Fitness Factor

When using a genetic algorithm, one critical component is a criterion that determines how well a specific design performs as compared to other designs. This is called a fitness factor. "The fitness of an individual is a measure of how 'good' the solution represented by the individual is. The better the solution, the higher the fitness…" [26]. A simple example would be the case of minimizing stress in a part. In this case the fitness factor would likely be the resulting stress of the analysis. In other more complicated cases, where several outputs drive the optimum, a fitness factor is determined in a different way. Possibly the best solution is to take the average of all the outputs. Or maybe it's more effective to take the minimum or maximum of all the output.

The purpose of this subsection is to explain that a fitness factor is used in the optimization but there are various ways of computing it depending on the specific analysis being run. Since this thesis does not explore specific analyses, but presents a general approach to laminate composite optimization, a specific method to find a fitness factor will not be shown here. Instead, the implementation of a specific fitness factor algorithm will be discussed in the implementation chapter (specifically subsection 4.4.2), where a specific optimization and analysis is shown.

4 IMPLEMENTATION

The methodology discussed above was implemented into computer programs that automate laminate composite design and analysis. One of these computer programs automatically creates the ply lay-up within a commercial CAD program given the outer surface geometry of a composite part and a spreadsheet that defines the parameters of the laminate. Another computer program automatically creates a laminate composite finite element model for commercial finite element packages given the same inputs. Since the base code for the applications are independent of commercial CAD software, it can be implemented into several commercial CAD and FEA packages.

The commercial CAD systems that are integrated with the automation program are: Siemens – NX and Dassault Systemes – CATIA. The CAD independent software used is the General Surface NURBS Library (GSNLib) distributed by Solid Modeling Solutions Inc. GSNLib is a software toolkit that provides methods for creation, storage and manipulation of NURBS surfaces [24]. This toolkit lends itself to be a good option to integrate with NX and CATIA because GSNLib, CATIA's API and NX's APIs are all based in the C/C++ programming language. This allowed the entire application to be to be written in a single programming language. The NX API used is called NX Open C and is a C based programming library that has several functions that allow for the creation and manipulation of NX geometry. The CATIA API used is called CAA RADE, and is a C++ based programming library that allows programmatic control of all interactive functions in CATIA.

The automation program was written within the framework of the CAD API and run from within the interactive CAD system. Two separate pieces of code are written for NX and CATIA to perform the translation to and from the specified CAD system. However, both NX and CATIA share the same base CAD independent code. This code utilizes many of the GSNLib function calls to handle the NURBS mathematics, but many of the other methods are implemented using custom code.

Figure 4-1 Figure 3-1shows a graphical representation of the approach of this program. It shows that the program is built upon the foundation of CAD independent algorithms. The CAD API framework is an interface between the interactive CAD system and the CAD-independent algorithms. The commercial CAD system is used for user interaction, the API framework is used to translate data to and from the CAD independent algorithms and the CAD-independent algorithms are used to create geometric and analytical data.

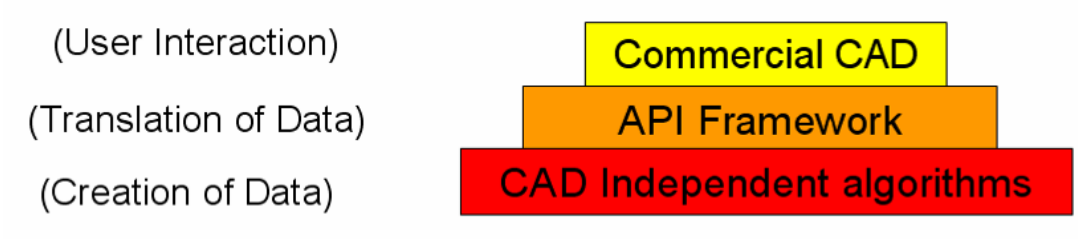

Figure 4-1: A representation of the approach of the automation program

4.1 Input Data from a User Familiar Source

After running the program from within the CAD system, an API function is called to prompt the user to select the surfaces that make up the outer shell of the composite part. In NX this surface definition can be defined in the CAD system or in a text file as a set of surface points that make up the surface. For CATIA the surface definition is only defined as a set of surface

points in a text file. Using another CAD API function call, the selected surfaces are queried for their NURBS surface control points, weights and knot vectors. These values are passed into a function that constructs an IwBSplineSurface object that is predefined in GSNLib. This object contains all the data to store a NURBS surface in CAD independent form. The translation code from NX to GSNLib is shown below.

```
	tag_t face;
	IwContext pContext;
	UF_MODL_bsurface_t nx_surf;
	UF_MODL_ask_bsurf(face,&nx_surf);
	gw_CPOINT   **Pw;
	ULONG    n, m, r, s;
	short   p, q;
	double *U, *V;
	//Getting num poles from NX NURBS surface
	n = nx_surf.num_poles_v -1;
	m = nx_surf.num_poles_u - 1;
	p = nx_surf.order_v - 1;
	q = nx_surf.order_u - 1;
	r = n+p+1;
	s = m+q+1;
	U = new double[r+1];
	V = new double[s+1];
	//Getting knot vector from NX NURBS surface
	for(int i=0; i<r+1; i++)
	{
		U[i] = nx_surf.knots_v[i];
	}
	for(int i=0; i<s+1; i++)
	{
		V[i] = nx_surf.knots_u[i];
	}
	//Getting control points from NX NURBS surface
	Pw = new gw_CPOINT*[(n+1)*(m+1)];
	for (int i=0;i<(n+1);i++)
	{
		Pw[i]=new gw_CPOINT[m+1];
	}
	for (int i=0, k=0;i<n+1;i++)
	{
		for (int j = 0; j < m+1; j++,k++)
		{
			Pw[i][j].x = nx_surf.poles[k][0];
			Pw[i][j].y = nx_surf.poles[k][1];
			Pw[i][j].z = nx_surf.poles[k][2];
			Pw[i][j].w = nx_surf.poles[k][3];
		}
	}
	//Creating IwBSplineSurface
	IwBSplineSurface *GSNLIB_surf = new (pContext)IwBSplineSurface(n, m, p,
q, r, s, Pw, U, V);
```

```
//This frees the memory when you're done with the surface
UF_MODL_free_bsurf_data(&nx_surf);
```

Another call to a CAD API function prompts the user to select a Microsoft Excel spreadsheet containing the ply parameters for the composite part, called a ply table. The ply table is a spreadsheet that describes each lamina in the composite lay-up. Each line of the spreadsheet expresses that lamina's fiber angle, thickness, material, and priority. The priority describes how prevalent that lamina is used throughout the laminate. For example, a ply with a priority of 1 will likely be used more frequently than a ply with a priority of 7. An example of a ply table is shown in table 4-1.

Table 4-2: An example of a ply table

Ply	Priority	Material ID	Thickness	Angle	Material
1	1	1	0.1	0	graphite
2	3	1	0.1	30	graphite
3	5	2	0.1	60	glass
4	7	2	0.1	90	glass
5	6	2	0.1	-60	glass
6	4	1	0.1	-30	graphite
7	2	1	0.1	0	graphite

Next, the user is prompted to select a spreadsheet that contains material properties for the materials used in the first spreadsheet. The material table is a spreadsheet that contains all the specific material properties for all of the composite materials that are used in the ply table.

After selecting these documents, they are opened and parsed programmatically to get out the data and store it in data structures within the program. The user also has the option instead to enter only the ply thickness into a GUI prompt to be stored. However, if this option is chosen,

additional pre-processing will be required. After being prompted for the ply and material properties, the user is also prompted to input the number of elements in the *u* and *v* directions to be used to create a mesh for finite element analysis of the composite part.

4.2 CAD Independent Algorithms

Once the geometry and parameter data is translated from the CAD system, the CAD independent algorithms programmed into C++ are used to automatically create the ply geometry and the laminate finite element model outside of the CAD system.

4.2.1 Automatic Geometry Creation

To create the 3-dimensional ply geometry, surfaces are offset from the two outer surfaces using the ply thickness parameters that were obtained from the Excel spreadsheet. Each offset surface is created using an offset function (shown below) that offsets each surface by the ply thickness amount. The function shown below follows the method outlined in equations 3-4 through 3-7. The function takes in an IwBSplineSurface object, a distance to offset and the grid density as the number of points in *u* and *v*.

```
IwBSplineSurface *create_GSNLIB_offset_surface(IwBSplineSurface *Surf, double
distance, IwContext &pContext, int &Upnts, int &Vpnts)
{
      IwBSplineSurface* newSurf;
      double coords[3], unit_norm[3];
      IwTArray<IwPoint3d> Curve_pnts;
      IwPoint3d tmpPoint;
      IwBSplineCurve *ChordCurve;
      IwContext CurveContext;
      IwTArray<IwBSplineCurve*> offset_splines;
      IwExtent1d newInterval;
      //intializing surface pnts for the new surface
      double ***surf_pnts=new double**[Upnts];
      for (int i=0; i<Upnts;i++)
      {
            surf_pnts[i] = new double*[Vpnts];
            for (int j=0;j<Vpnts;j++)
            {
```

```
                surf_pnts[i][j] = new double[3];
            }
        }
        // CYCLE U SURFACE PARAMETER (STRINGER)
        for (int i=0;i<Upnts;i++)
        {
            // CYCLE V SURFACE PARAMETER (CROSS-CURVE)
            for (int j=0;j<Vpnts;j++)
            {
                ask_face_props_GSNLIB(Surf, i/double(Upnts-1),
                j/double(Vpnts-1), coords);
                ask_face_norm_GSNLIB(Surf, i/double(Upnts-1),
                j/double(Vpnts-1), unit_norm);
                //Offsets the points in the normal direction
                add_mult_array( coords, unit_norm, distance,
                surf_pnts[i][j]);
            }
        }
        //CYCLE IN SPANWISE DIRECTION, CREATING CHORDWISE CURVES
        for (int i=0;i<Upnts;i++)
        {
            for (int j=0;j<Vpnts;j++)
            {
                tmpPoint.x = surf_pnts[i][j][0];
                tmpPoint.y = surf_pnts[i][j][1];
                tmpPoint.z = surf_pnts[i][j][2];
                Curve_pnts.Add(tmpPoint);
            }
            create_GSNLIB_spline(Curve_pnts, 3, ChordCurve, CurveContext);
            Curve_pnts.RemoveAll();
            offset_splines.Add(ChordCurve);
        }
        create_GSNLIB_thru_curve(offset_splines, 0, 1, 3, newSurf, pContext);
        return newSurf;
}
```

Once offset, these surfaces are trimmed to intersecting offset surfaces using the proprietary trimming method mentioned in chapter 3 (see Figure 4-2).

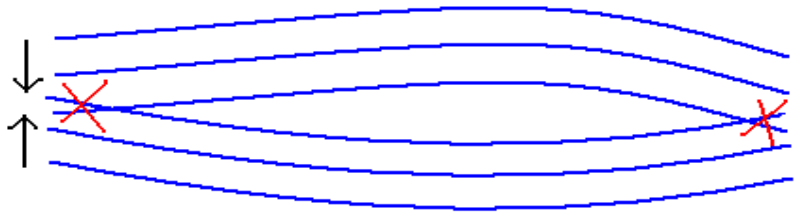

Figure 4-2: Offset from outer surfaces and trim

After the new surfaces are created and trimmed, they are stored in GSNLib IwBSpline-Surface objects to await translation back into the CAD system. The parameters and attributes that were read in from the Excel spreadsheet are stored in a data structure vector that contains all the IwBSplineSurface objects and the attributes that are associated with them. Figure 4-3 shows a side view of the offset and trimmed ply geometry for a generic part after it is translated back into NX.

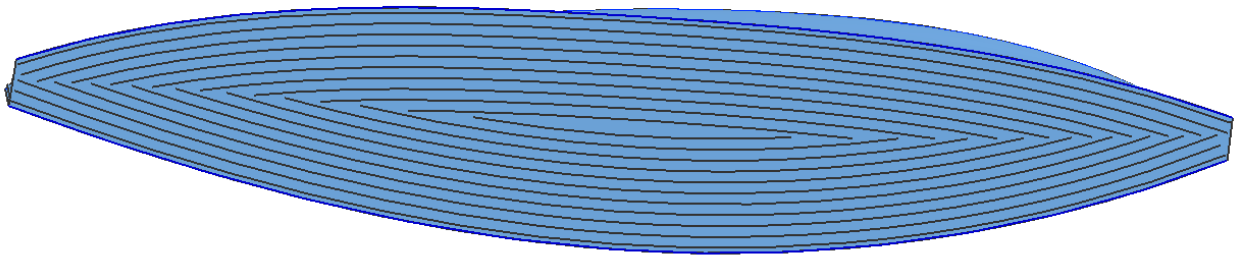

Figure 4-3: A side view of a generic NX ply geometry creation

4.2.2 Automatic Generation of Finite Element Model

The finite element model is created using equations 3-8 through 3-14 discussed in chapter 3. First, a mid-surface of the part is automatically created by using equation 3-8 in chapter 3.

```
for (int i=0;i<Unum;i++)
{
      for (int j=0;j<Vnum;j++)
      {
            GSNLIB_geom::ask_face_props_GSNLIB(Plys.at(0).NURBSSurface.
            GetAt(0), i/double(Unum-1), j/double(Vnum-1), pnt1);
            GSNLIB_geom::ask_face_props_GSNLIB(Plys.at(1).NURBSSurface.
            GetAt(0), i/double(Unum-1), j/double(Vnum-1), pnt2);
            if (pnt1[0]-pnt2[0]<.0001 && pnt1[1]-pnt2[1]<.0001 &&
            pnt1[2]-pnt2[2]<.0001)
            {
                  newPnt[0] = pnt1[0];
                  newPnt[1] = pnt1[1];
                  newPnt[2] = pnt1[2];
            }
```

```
                else
                {
                        math_func::average(pnt1, pnt2, newPnt);
                }
                Curve_pnts.Add(newPnt);
        }
        //creating chordwise curves
        GSNLIB_geom::create_GSNLIB_spline(Curve_pnts, 3, ChordCurve,
        CurveContext);
        Curve_pnts.RemoveAll();
        midSurfSplines.Add(ChordCurve);
}
GSNLIB_geom::create_GSNLIB_thru_curve(midSurfSplines, 0, 0, 3, midSurf,
midSurfContext);
```

The mid-surface is then automatically meshed to form quadrilateral laminated shell elements as shown in equations 3-9 and 3-10. After that, following the procedure discussed in equations 3-11 through 3-14 of chapter 3, each element is associated with the corresponding plies that make up the 3-dimensional component of the shell element.

```
//Gets the nodes and normal lines
//Finds the thickness for every element
int k=0;
for(int i=0; i<Unum; i++)
{
        for(int j=0; j<Vnum; j++)
        {
                //Getting the node locations
                GSNLIB_geom::ask_face_props_GSNLIB(midSurf, i/double(Unum-
                1), j/double(Vnum-1), pnt_on_surf);
                nodes.Add(pnt_on_surf);
                //Getting the normal lines at the center of the elements
                if(i>0 && j>0)
                {
                        Uparm = (double(i-1)/double(Unum-
                        1)+double(i)/double(Unum-1))/2;
                        Vparm = (double(j-1)/double(Vnum-
                        1)+double(j)/double(Vnum-1))/2;
                        GSNLIB_geom::ask_face_props_GSNLIB(midSurf, Uparm,
                        Vparm, pnt_on_surf);
                        GSNLIB_geom::ask_face_norm_GSNLIB(midSurf, Uparm,
                        Vparm, unitNorm);
                        math_func::add_mult_array(pnt_on_surf, unitNorm, 100,
                        norm_pos_pnt);
                        math_func::sub_mult_array(pnt_on_surf, unitNorm, 100,
                        norm_neg_pnt);
                        IwLine::CreateLineSegment(lineContext, 3,
                        pnt_on_surf, norm_pos_pnt, normLine1);
                        crInterval = normLine1->GetNaturalInterval();
                        intersectMe = (IwCurve*)normLine1;
```

```
Plys.at(0).NURBSSurface.GetAt(0)-
>GlobalCurveIntersect(Plys.at(1).NURBSSurface.GetAt(0
)->GetNaturalUVDomain(), *intersectMe, crInterval,
.000001, rSolutions);
if (rSolutions.GetSize() > 0)
{
      IwSolution *sol = rSolutions.GetDataArray();
      uParameter = sol->m_vStart.m_adParameters[1];
      vParameter = sol->m_vStart.m_adParameters[2];
      GSNLIB_geom::ask_face_props_GSNLIB(Plys.at(1).N
      URBSSurface.GetAt(0), uParameter, vParameter,
      pnt2);
      dist1 =
      math_func::distance_between_points(pnt_on_surf,
      pnt2);
}

IwLine::CreateLineSegment(lineContext, 3,
pnt_on_surf, norm_neg_pnt, normLine2);
crInterval = normLine2->GetNaturalInterval();
intersectMe = (IwCurve*)normLine2;
Plys.at(1).NURBSSurface.GetAt(0)-
>GlobalCurveIntersect(Plys.at(0).NURBSSurface.GetAt(0
)->GetNaturalUVDomain(), *intersectMe, crInterval,
.000001, rSolutions);
if (rSolutions.GetSize() > 0)
{
      IwSolution *sol = rSolutions.GetDataArray();
      uParameter = sol->m_vStart.m_adParameters[1];
      vParameter = sol->m_vStart.m_adParameters[2];
      GSNLIB_geom::ask_face_props_GSNLIB(Plys.at(0).N
      URBSSurface.GetAt(0), uParameter, vParameter,
      pnt2);
      dist2 =
      math_func::distance_between_points(pnt_on_surf,
      pnt2);
}
//Pre-loading the intersection vector
sheetIntersections.push_back(temp);
for (int a=0; a<Plys.size();a++)
{
      sheetIntersections[k].push_back(0);
}
//Adding the plies based on priority
totDist = dist1 + dist2;
elemThk.push_back(totDist);
numPlies = 0;
currentDist = 0;
//Finding the number of plies in each element
while(totDist > currentDist && numPlies<Plys.size())
{
      currentDist = currentDist +
      priorityVector.at(numPlies).PlyProperties.Thick
      ness;
      int me =
      priorityVector.at(numPlies).PlyProperties.PlyNu
      m;
```

```
                        sheetIntersections[k].at(priorityVector.at(numP
                        lies).PlyProperties.PlyNum-1) = 1;
                        numPlies++;
                    }
                    k++;
                }
            }
        }
```

The image shown in Figure 4-4 is a laminate finite element model of a vertical stabilizer that is brought into Altair's Hyper Mesh for visualization. The various colors shown represent the unique ply definitions for each element.

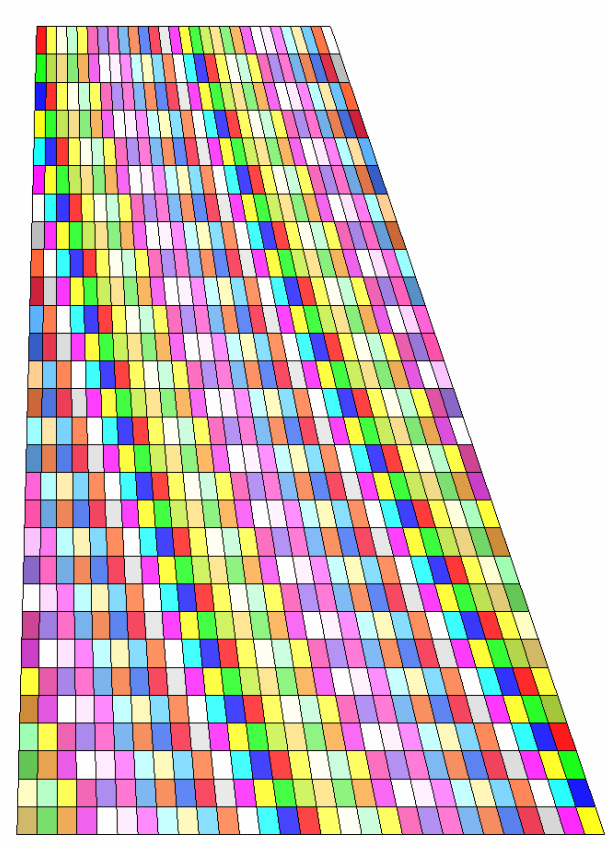

Figure 4-4: Laminate FEA model of a vertical stabilizer

4.3 Output Data into a Useable Source

The surfaces that were stored in the IwBSplineSurface objects are translated back into CAD surfaces for visualization. In NX this is simply done using an NX Open C function that takes as input the NURBS control points, weights and knot vectors and outputs a surface directly into NX. The code below is function that takes in an IwBSplineSurface object, extracts the data from it and puts it into a format to be read by the NX API function UF_MODL_create_bsurface.

```
tag_t create_NX_bsurface_from_GSNLIB_surface(IwBSplineSurface *GSNLIB_Surf)
     {
      tag_t nx_bsurf;
      int num_states;
      UF_MODL_state_p_t states;
      UF_MODL_bsurface_t nx_bsurf_info;

      double *uKnots;
      double *vKnots;
      ULONG KnotCountU;
      ULONG KnotCountV;
      IwTArray<ULONG> uKnotMultiplicities;
      IwTArray<ULONG> vKnotMultiplicities;
      IwTArray<IwPoint3d> ControlPoints;
      IwTArray<double> Weights;
      ULONG UCount;
      ULONG VCount;

      GSNLIB_Surf->GetKnotsPointers(KnotCountU, KnotCountV, uKnots, vKnots);
      GSNLIB_Surf->GetControlPointNet(UCount, VCount, ControlPoints,
      Weights);
      nx_bsurf_info.is_rational = GSNLIB_Surf->IsRational();
      nx_bsurf_info.order_u = GSNLIB_Surf->GetDegree(IW_SP_U)+1;
      nx_bsurf_info.order_v = GSNLIB_Surf->GetDegree(IW_SP_V)+1;
      nx_bsurf_info.num_poles_u = UCount;
      nx_bsurf_info.num_poles_v = VCount;
      nx_bsurf_info.knots_u = new double[KnotCountU];
      nx_bsurf_info.knots_v = new double[KnotCountV];
      nx_bsurf_info.poles = new double[UCount*VCount][4];
      for(int i=0; i<KnotCountU; i++)
      {
            nx_bsurf_info.knots_u[i] = uKnots[i];
      }
      for(int i=0; i<KnotCountV; i++)
      {
            nx_bsurf_info.knots_v[i] = vKnots[i];
      }

      for(int i=0; i< UCount*VCount; i++)
      {
            nx_bsurf_info.poles[i][0]= ControlPoints[i].x;
```

```
            nx_bsurf_info.poles[i][1]= ControlPoints[i].y;
            nx_bsurf_info.poles[i][2]= ControlPoints[i].z;
            if(nx_bsurf_info.is_rational == 0)nx_bsurf_info.poles[i][3]= 1;
            else nx_bsurf_info.poles[i][3]= Weights[i];
      }
      //Creating the NX Bsurface
      UF_MODL_fix_bsurface_data (.00001, &nx_bsurf_info, &num_states,
&states);
      UF_MODL_create_bsurface (&nx_bsurf_info, &nx_bsurf, &num_states,
&states);
      UF_MODL_update ( );
      return nx_bsurf;
}
```

The translation from GSNLib back to CATIA has a couple more steps to than the GSNLib to NX process. In CAA RADE there is a function that takes the same inputs as the NX Open C function and outputs a geometrical NURBS object. This object can only be viewed in CATIA after it is converted into a skin and then into a datum feature. Therefore it must be added to the procedural view for actual visualization in CATIA. The function below takes in an IwBSplineSurface object , extracts the NURBS data from it, creates a CATNurbsSurface and does all the necessary steps for CATIA visualization.

```
void create_CATSurface_from_GSNLIB_surface(IwBSplineSurface *GSNLIB_Surf)
{
      double *uKnots;
      double *vKnots;
      ULONG KnotCountU;
      ULONG KnotCountV;
      IwTArray<ULONG> uKnotMultiplicities;
      IwTArray<ULONG> vKnotMultiplicities;
      IwTArray<IwPoint3d> ControlPoints;
      IwTArray<double> Weights;
      ULONG UCount;
      ULONG VCount;
      //Creation of the grid points to be passed as the knot vector argument
      GSNLIB_Surf->GetKnotsPointers(KnotCountU, KnotCountV, uKnots, vKnots);
      GSNLIB_Surf->GetControlPointNet(UCount,VCount,ControlPoints,Weights);
      int nbPoleU = UCount;
      int nbPoleV = VCount;
      CATMathGridOfPoints gridOfPoints(nbPoleU,nbPoleV);
      CATMathPoint controlPoint;
      int k=0;
      for (int i=0; i< nbPoleU;i++)
      {
            for (int j=0; j< nbPoleV;j++)
```

```
      {
            controlPoint.SetX( ControlPoints[k].x);
            controlPoint.SetY( ControlPoints[k].y);
            controlPoint.SetZ( ControlPoints[k].z);
            gridOfPoints.SetPoint(controlPoint,i,j);
            k++;
      }
}
//Creation of the knot vectors
CATLONG32 IsPeriodic= 0;
CATLONG32 UDegree= GSNLIB_Surf->GetDegree(IW_SP_U), VDegree=
GSNLIB_Surf->GetDegree(IW_SP_V);
CATLONG32 UKnotsCount = KnotCountU - 2*UDegree;
CATLONG32 VKnotsCount = KnotCountV - 2*VDegree;
//Knot vector
double *UKnots;
UKnots = new double[UKnotsCount];
double *VKnots;
VKnots = new double[VKnotsCount];
for(int i=0; i<UKnotsCount ;i++)
{
      UKnots[i] = uKnots[i+UDegree];
}
for(int i=0; i<VKnotsCount ;i++)
{
      VKnots[i] = vKnots[i+VDegree];
}
//Multiplities
CATLONG32 *UMultiplicities;
UMultiplicities = new CATLONG32[UKnotsCount];
CATLONG32 *VMultiplicities;
VMultiplicities = new CATLONG32[VKnotsCount];
//U
UMultiplicities[0] = UDegree+1;
for(int i=1; i<UKnotsCount-1 ;i++)
{
      UMultiplicities[i] = 1;
}
UMultiplicities[UKnotsCount-1] = UDegree+1;
//V
VMultiplicities[0] = VDegree+1;
for(int i=1; i<VKnotsCount-1 ;i++)
{
      VMultiplicities[i] = 1;
}
VMultiplicities[VKnotsCount-1] = VDegree+1;
CATLONG32 IndexOffset= 0;
CATKnotVector NonUniformU(UDegree,IsPeriodic,UKnotsCount,UKnots,
  UMultiplicities,IndexOffset);
CATKnotVector NonUniformV(VDegree,IsPeriodic,VKnotsCount,VKnots,
  VMultiplicities,IndexOffset);
//Creation of a rational NURBS surface
CATLONG32 isRational=1;
double * aWeights=new double[nbPoleU*nbPoleV];
for (int i = 0; i < nbPoleU*nbPoleV; i++)
{
    aWeights[i] = 1.;
```

```
        }
        // NURBS Surface creation
        CATNurbsSurface * piSurf1 = piGeomFactory->
        CATCreateNurbsSurface(NonUniformU,
        NonUniformV,isRational,gridOfPoints,aWeights);
        if (NULL==piSurf1)
        {
            printf( "NURBS surface could not be created");
            return;
        }
        delete [] aWeights;
        aWeights = NULL;
        //Creation of the skin
        CATSurLimits surMaxLimits ;
        piSurf1->GetMaxLimits(surMaxLimits) ;
        CATSoftwareConfiguration * pConfig = new CATSoftwareConfiguration();
        CATTopData topdata(pConfig);
        CATTopSkin * pSkinOpe =::CATCreateTopSkin(piGeomFactory,
                &topdata,
                piSurf1,
                &surMaxLimits);
        if (NULL==pSkinOpe)
        {
            return;
        }
        pSkinOpe->Run();
        // Gets the resulting body
        CATBody * piSkinBody = pSkinOpe->GetResult();
        if (NULL==piSkinBody)
        {
            return;
        }
        // Deletes the operator
        delete pSkinOpe;
        pSkinOpe=NULL;

        CATIDatumFactory_var spDatumFactory;
        spDatumFactory = pSpecContainer;
        CATISpecObject* oDatumFeature;
        spDatumFactory->InstanciateDatum(piSkinBody, oDatumFeature);
        oDatumFeature->Update();
        spCurObj = oDatumFeature;
        spCurObj->InsertInProceduralView();
        pConfig->Release();

}
```

The attributes that are associated with each IwBSplineSurface are translated into NX using a function that takes as input a string of text and numerical values and outputs an NX attribute that is associated with the NX surface. Since CATIA has no such “attribute” feature available to associate general attributes with surfaces, this step was not implemented in CATIA.

The finite element analysis input file is created programmatically from the nodes and elements as well as the ply materials and properties. Based on what the user selects as the analysis package to be output to, the data is organized in a way that conforms to that software's input file format. The result is an analysis data file formatted for use by a wide variety of commercial Finite Element processing packages including ANSYS, LS-DYNA and NATRAN. The following code creates an ANSYS input file from the data created in subsection 4.2.2. Note that the code follows equation 3-10 for the organization of the elements. Inputs include the model nodes, sheet intersections and an output file.

```
void printANSYS(IwTArray<IwPoint3d> nodes, vector <vector<int>>
&sheetIntersections, ofstream &outfile)
{
	double angle;
	double thickness;
	double pnt_on_surf[3];

	int numElements = (Unum-1)*(Vnum-1);

	outfile<<fixed;
	outfile<<"/PREP7"<<"!"<<endl;

	//Printing Nodes
	outfile<<"! Nodes"<<endl;
	outfile<<"CSYS,0"<<endl;
	for(int i=0; i<nodes.GetSize(); i++)
	{
		outfile<<setprecision(4);
		outfile<<"N, "<<i+1<<", "<<nodes.GetAt(i).x<<",
		"<<nodes.GetAt(i).y<<", "<<nodes.GetAt(i).z<<endl;
	}

	//Printing Elements
	outfile<<"!"<<endl<<"! Specify Element type and options"<<endl;
	outfile<<"ET,1,"<<"SHELL181"<<endl;
	outfile<<"KEYOPT,1,3,2"<<endl;
	outfile<<"KEYOPT,1,8,2"<<endl;
	outfile<<"!"<<endl<<"! Elements"<<endl;
	for(int i=0,int j=0,int k=0; i<numElements+Unum-2; i++,j++,k++)
	{
		if(j > Vnum-2)
		{
			i++;
			j=0;
		}
```

```
            outfile<<"SECNUM, "<<k+1<<endl;
            outfile<<"E, "<<i+1<<", "<<i+2<<", "<<i+Vnum+2<<",
            "<<i+Vnum+1<<endl;
            //Printing properties for that ply
            outfile<<"!"<<endl<<"! Properties"<<endl;
            outfile<<"SECTYPE,"<<k+1<<",SHELL"<<endl;
            outfile<<"! thickness, material and angle"<<endl;
            for(int j=0; j<Plys.size(); j++)
            {
                  //only print ply if is a one
                  if(sheetIntersections[k][j]==1)
                  {
                        outfile<<"SECDATA,
                        "<<Plys.at(j).PlyProperties.Thickness<<",
                        "<<Plys.at(j).PlyProperties.MaterialID<<", "
                        <<Plys.at(j).PlyProperties.ply_angle<<endl;
                  }
            }
            outfile<<"!"<<endl;
      }

      //Printing material properties
      outfile<<"!"<<endl<<"! Materials"<<endl<<"!"<<endl;
      for(int i=0; i<VecMatProps.size(); i++)
      {
            outfile<<"! "<<VecMatProps.at(i).MaterialName.c_str()<<endl;
            outfile<<"MP,EX,"<<VecMatProps.at(i).MaterialID<<","<<VecMatProps
            .at(i).E1<<endl;
            outfile<<"MP,EY,"<<VecMatProps.at(i).MaterialID<<","<<VecMatProps
            .at(i).E2<<endl;
            outfile<<"MP,PRXY,"<<VecMatProps.at(i).MaterialID<<","<<VecMatPro
            ps.at(i).v12<<endl;
            outfile<<"MP,GXY,"<<VecMatProps.at(i).MaterialID<<","<<VecMatProp
            s.at(i).G12<<endl;
            outfile<<"MP,GYZ,"<<VecMatProps.at(i).MaterialID<<","<<VecMatProp
            s.at(i).G12<<endl;
            outfile<<"MP,GXZ,"<<VecMatProps.at(i).MaterialID<<","<<VecMatProp
            s.at(i).G12<<endl;
            outfile<<"MP,DENS,"<<VecMatProps.at(i).MaterialID<<","<<VecMatPro
            ps.at(i).Density<<endl;
      }
     return;
}
```

Similar code was written for the implementation of LS-DYNA and NASTRAN but is left to the reader to determine.

4.4 Optimization

To establishing a working optimization, programs are needed to automatically generate a finite element model and assign boundary conditions to the finite element model. The program to generate the finite element model has already been discussed. Subsection 4.4.1 will discuss the automatic assignment of boundary conditions. To link the programs together and drive the optimization to an optimum, an optimization framework is used and discussed in subsection 4.4.2. In addition, a modal analysis optimization of a composite wing is discussed with the implemented fitness function.

4.4.1 Automatic Boundary Conditions for the Finite Element Model

ANSYS was one of the FEA packages that were used to demonstrate the program's effectiveness. Applying automatic boundary conditions to the ANSYS finite element model was implemented using the ANSYS Parametric Design Language (APDL). APDL is a scripting language that allows users to automate ANSYS with the use of do-loops, if/else statements, and other automated ANSYS commands [1]. Scripts can be easily written to automatically load, constrain and solve analyses. An ADPL script was written to automatically set up a modal analysis on a cantilevered wing. This script tells ANSYS what analysis to be performed (in this case a modal analysis), automatically constrain one end of the wing in all 6 degrees of freedom and specifies which modes to output to a file. The following example is a script that automatically solves a modal analysis of a cantilevered model with a 20x20 mesh density as shown in Figure 4-5:

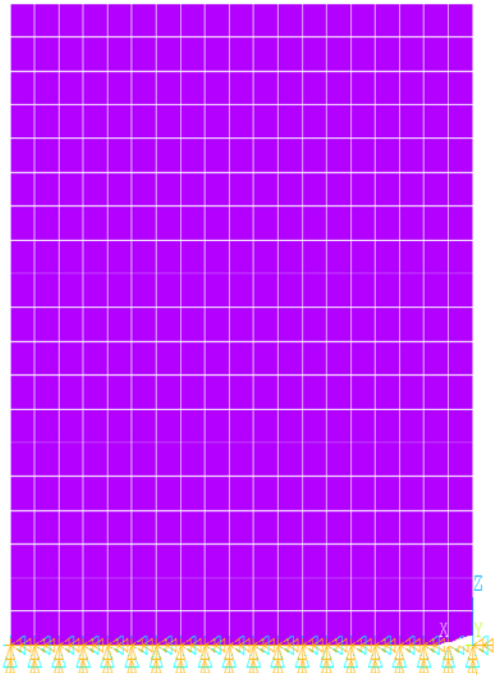

Figure 4-5: An ANSYS shell model of a cantilever beam

```
/solu
ANTYPE,2
MODOPT,SUBSP,10
EQSLV,FRONT
MXPAND,10, , ,0
LUMPM,0
PSTRES,0
MODOPT,SUBSP,10,0,0, ,OFF
RIGID,
SUBOPT,8,4,14,0,0,ALL
*SET,NodeNum,1
*SET, numInc,1
*SET,NumNodesU,20
*SET,NumNodesV,20
*DO, inc, 1, NumNodesV * NumNodesU, numInc
        d,NodeNum,ux,,,,,uy,uz,rotx,roty,rotz
        NodeNum = NodeNum + numInc
*ENDDO
SOLVE
FINISH

/POST1
*DO, i, 1, 10, 1
```

```
        *GET, mode, MODE, i,FREQ
        /OUTPUT,Modes,txt,,APPEND
        *VWRITE,i, mode
        Mode %I: %14.7G
        /OUTPUT
        /show,close
*ENDDO
```

4.4.2 Optimization using Isight

A laminate composite optimization was set up using commercial optimization and process flow software called Isight (by SIMULIA). The objective of the optimization is to drive the natural frequencies of the wing away from known keep out frequencies in order to avoid resonant conditions. The design variables of the optimization are the fiber angles of each of the plies in the lay-up. These design variables are put into the Isight software along with the objective. A genetic algorithm is configured to drive the optimum away from the keep out frequencies by adjusting the fiber angles of each ply. The genetic algorithm first assigns random values from -90 to 90 degrees to the fiber angles. Over time the genetic algorithm evolutionarily drives these angles to values that make the wing's natural frequencies as far from the keep out frequencies as possible.

The fitness factor of this optimization is somewhat more complex than just the output from the analysis. Each modal frequency needs to stay as far away from the keep out frequencies as possible. Often when a modal frequency moves away from one frequency range, it starts to get closer to another keep-out frequency. The compromise in this case is to center the natural frequency between the two keep-out frequencies. To do this, the algorithm to determine the fitness first finds the distance between each natural frequency and the keep-out frequency. The algorithm then finds the difference between each natural frequency and its closest keep-out frequency. The fitness factor becomes the minimum of all these. The objective of the

optimization is to maximize the minimum of all the minimum distances. The result is the optimum design's modal frequencies will be as far away to any of the keep out frequencies as possible.

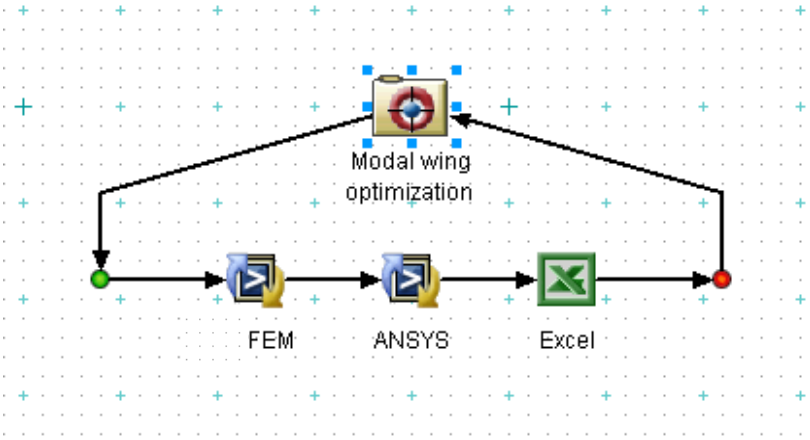

Figure 4-6: Isight process flow for the laminate composite optimization

Figure 4-6 shows the process flow for the modal analysis optimization of the laminate composite wing described above. The modal optimization loop is run as many times as is needed to get the optimization to converge (possibly hundreds or even thousands of times). The first task in the optimization runs the code to set up the mesh and laminate properties of the finite element model. The next task runs ANSYS with the APDL script to automatically set up the constraints, solve the model and output results to a text file. The third step, Isight takes the results from the text file and puts them into Excel. Excel is pre-programmed to determine the goodness of fit of the solution from the results of the optimization. The fitness result is then automatically taken from Excel and put back into the Isight optimization.

5 RESULTS

As stated in chapter 1, the objective of this thesis is to propose and demonstrate methods that apply CAD independent algorithms to streamline laminate composite design, analysis and optimization in a CAD centric way by accomplishing the following:

- Automatically create 3D geometry for individual plies in a laminate composite lay-up for a composite part with complex geometry
- Streamline the creation of detailed laminate composite finite element models
- Optimize the composite lay-up for a composite part made of several layers

Section 5.1 shows the results of the composite design automation tool that automatically creates 3D geometry for individual plies in a laminate part with complex geometry. Section 5.2 discusses the results of the composite analysis automation tool that automatically creates laminate finite element models. Section 5.3 discusses the results of the modal optimization of the composite wing.

5.1 Composite Design Automation

Laminate visualizations have been successfully created in NX and CATIA using the composite automation program described above. Figure 5-1 shows the results of a simple test of the ply geometry creation tool. These results show the ply geometry that was created automatically given this simple outer geometry definition. This can be seen as the offset surfaces shown in Figure 5-1 between the upper and lower surfaces. This model's outer surface definition

only varies in two dimensions showing that the program can handle offsetting and trimming capability in 2D space.

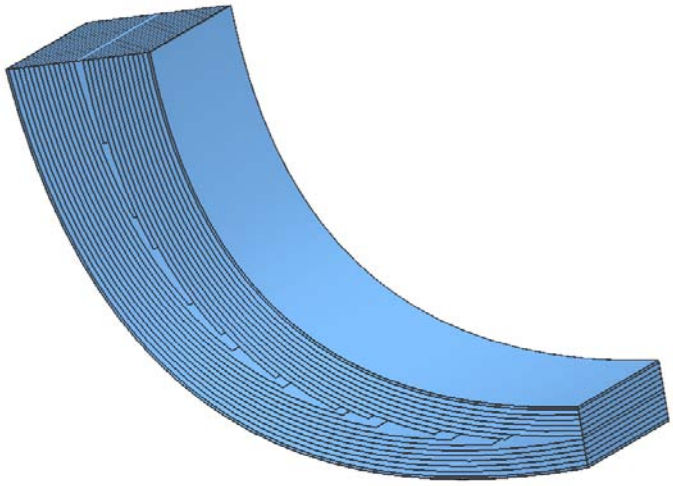

Figure 5-1: A test case showing simple ply geometry created in NX

The program was also used to successfully create ply lay-up geometry for a part varying in 3 dimensions. This is shown by making the lay-up geometry for a model airplane composite vertical stabilizer given an outer surface definition in NX. Figure 5-2 shows the surface definition of the vertical stabilizer loaded in NX and Figure 5-3 shows the NX ply lay-up created by the program shown in wireframe view. This definition is much more difficult to create than the first example due to the fact that the outer surface definition varies in all three dimensions. Table 5-1 shows the ply table created in Excel that defines the ply properties used create the geometry for the ply lay-up.

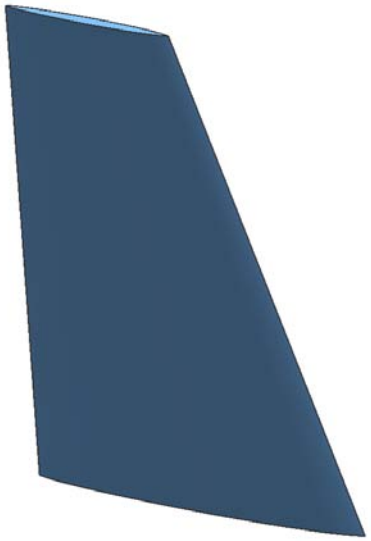

Figure 5-2: Vertical stabilizer loaded in NX

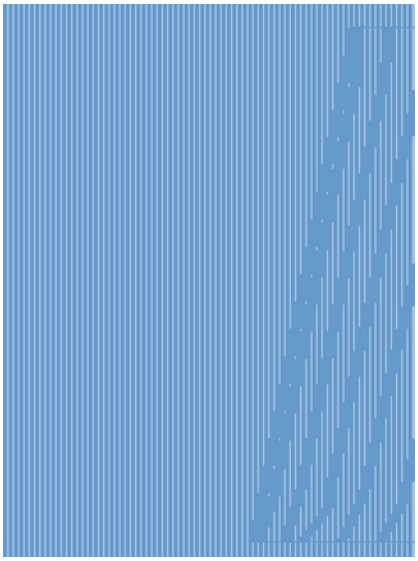

Figure 5-3: Vertical stabilizer ply lay-up

Table 5-1: The ply property table from MS Excel

Ply Properties						
			(mm)	(degrees)		
Ply	Material ID	Pre-form	Thickness	Angle	Material	Weave
1	1	B	0.5	0	graphite	Plain
2	1	B	0.5	45	graphite	Plain
3	1	B	0.5	-45	graphite	Plain
4	1	B	0.5	90	graphite	Plain
5	1	B	0.5	0	graphite	Plain
6	1	B	0.5	45	graphite	Plain
7	1	B	0.5	-45	graphite	Plain
8	1	B	0.5	90	graphite	Plain
9	1	B	0.5	0	graphite	Plain
10	1	B	0.5	45	graphite	Plain
11	1	B	0.5	-45	graphite	Plain
12	1	B	0.5	90	graphite	Plain
13	1	B	0.5	0	graphite	Plain
14	1	B	0.5	45	graphite	Plain
15	1	B	0.5	-45	graphite	Plain
16	1	B	0.5	90	graphite	Plain

To compare the speed of one composite design automation program using CAD independent algorithms vs. one using only a CAD API, a significant function is tested for speed in NX Open C, CAA RADE and GSNLIB. The function tested is one of the primary functions used in the automated laminate composite design program and is used hundreds of thousands of times throughout. Although this function may not represent the actual computational speed of the entire program, it does represent a large portion of the program, and therefore it is a good measure of the speed of the entire program. This function takes in *u* and *v* surface parameters as input and outputs the Cartesian point on the surface associated with it. The speed test performed calls the function a total of 1,000,000 times on each surface tested. The test calls the function to parse the surface making 1/1000 unit steps in the *u* direction and 1/1000 steps in the *v* direction. Mathematically this can be shown as parsing through the surface $\{ S(u,v) \}$ to get a grid of points $\{ G_{i,j} \}$:

$$G_{i,j} = \sum_{i=1}^{1000} \sum_{j=1}^{1000} S(i/1000, j/1000) \tag{5-1}$$

The speed is clocked for each surface test for NX Open C, GSNLib and CAA RADE. Table 5-2 shows the resulting times recorded for the each of the tests.

Table 5-2: Test results from speed comparison of NX, GSNLIB and CAA RADE

Surface	**Time in NX Open C**	**Time in GSNLIB**	**Time in CAA RADE**
Surface 1	46.766 sec.	1.516 sec.	.375 sec.
Surface 2	47.953 sec.	1.516 sec.	.375 sec.
Surface 3	47.187 sec.	1.578 sec.	.390 sec.
Surface 4	47.719 sec.	1.516 sec.	.375 sec.

When comparing NX Open C to GSNLib, the test in GSNLib ran about 30 times faster than NX Open C. This is dramatic increase in computational time using GSNLib over NX Open C. The performance of GSNLib and CAA RADE, however, are on the same order of magnitude. This is because CAA RADE allows for direct access to CATIA's geometry kernel, where NX Open C operates at a level or two above the geometry kernel.

The ply definition and analysis files created automatically using the automation program written takes a designer less than one minute to run. In contrast, it takes the designer an estimated 40 hours to create the same ply definition if done interactively in a CAD system. Therefore, when using this automation program, companies that design laminate composite parts will save a significant amount of time and money. In addition, this program allows designers to evaluate several design configurations in the same time it would take to create a single design. This allows for superior designs to be created in less time.

Although this automation process could be programmed using NX Open C, without utilizing the CAD independent functions in GSNLib, there are several reasons why using these functions in an automation program makes the program superior. First, the advantages of the CAD centric approach including simplified viewing and data storage/transfer are held intact. Second, the application is portable enough to "plug" into any commercial CAD system with an existing API. Third, the use of GSNLib functions allows for a 30 times speed increase over the NX Open C API.

5.2 Composite Analysis Automation

The automated composite finite element tool has been successful in automatically creating laminate composite finite element models for ANSYS, NASTRAN and LS-DYNA.

Figure 5-4 shows three models of a wing. The first model is a wing created in NX. The second is a mid-surface shell element model of the wing created by the automated composite finite element tool and loaded into ANSYS. The model has a mesh density of 15x30 and contains thirteen, 0.1" lamina at the thickest area. The third model is the same ANSYS finite element model as the second, but it has the lamina thicknesses of the shell elements shown in 3-dimensions for better visualization.

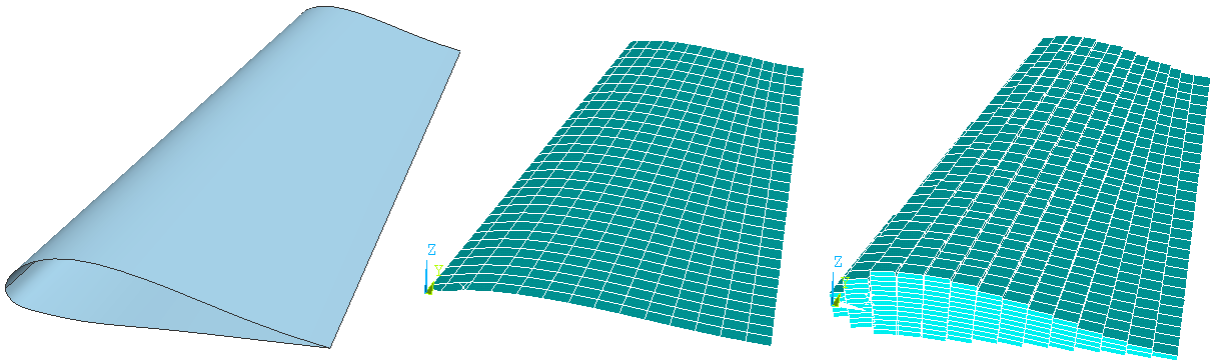

Figure 5-2: NX Wing, FE shell mesh and FE shell mesh with lamina thicknesses shown

The major advantage of using the automated composite finite element tool over other zone based, automated methods is that the finite element models that it creates more accurately represent the laminate lay-up for models with complex topology. As shown in the far right image of Figure 5-4, each element contains its own lay-up. Each ply in this lay-up is defined with some thickness (shown in figure 5-4), fiber angle (not shown) and material properties (also not shown). Because each element has its own ply lay-up, the finite element models can represent areas with large variation with more detail than those that are divided into a just a few zones, as done in previous methods. Except for the highly unlikely case where the number of zones matches the number of elements in a finite element model, geometrically complex models (possibly as simple as the one shown in Figure 5-4 or as complex as Figure 5-5) will be more

accurately represented than the models created using a zone based method. The increased detail produced from this tool will yield more accurate analysis results in models with rapidly varying geometry (i.e. the model in Figure 5-5) compared to other zone based methods.

Figure 5-5 shows an example of a more complex topology that was modeled using the automated composite finite element tool. The mesh density is 40x40 nodes and the model contains 36 plies that are .01" thick. Although this example is not an actual part, it is useful to show here to demonstrate the complexity of models that can be made using this tool. Other zone based methods would not come close to being able to model this part with as much detail.

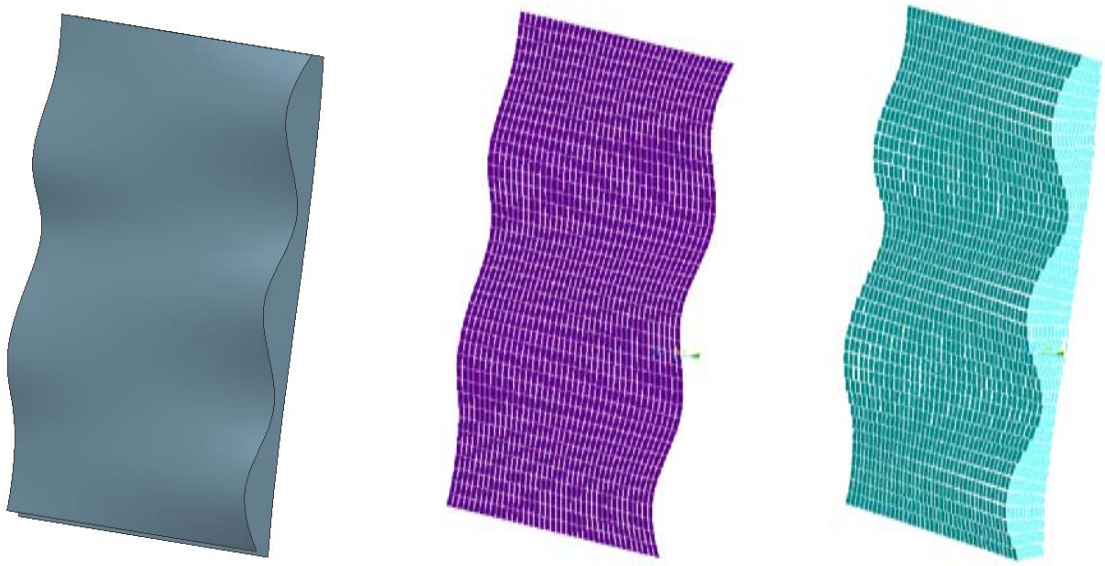

Figure 5-3: Complex NX Surface, FE shell mesh, FE mesh with thicknesses shown

There is also an obvious time savings advantage of using this automated approach instead of creating a laminate composite model directly in the FEA package. If a finite element model such as the one shown in Figure 5-4 were done manually, it would take a proficient user an estimated 40 hours of work. This is because each of the 450 elements in that model needs to

have a unique ply lay-up assigned to it. Using the automated composite finite element tool, the same model takes only a few seconds to create.

Table 5-3 shows run times of the automated composite finite element tool to create the wing model shown in figure 4. The run time increases approximately linearly with the number of total nodes in the model. As the number of plies in the model increases, the time increases by around 50%. This study shows that even finely meshed models with large numbers of plies can be created extremely quickly (under 30 seconds) as compared to manual methods.

Table 5-3: Test results of the speed of the automated composite finite element tool

Num plies	10x20 nodes	15x30 nodes	30x60 nodes	50x100 nodes
13 plies	0.281 sec.	0.687 sec.	2.766 sec.	7.609 sec.
26 plies	0.328 sec.	0.781 sec.	3.172 sec.	8.562 sec.
52 plies	0.422 sec.	0.969 sec.	4.047 sec.	10.844 sec.
104 plies	0.609 sec.	1.375 sec.	5.563 sec.	15.281 sec.
208 plies	0.906 sec.	2.171 sec.	8.67 sec.	24.172 sec.

5.3 Composite Analysis Optimization

The modal analysis optimization of a laminate composite wing as discussed in subsection 4.4.2 was successful in driving the modal frequencies away from certain given keep out frequencies. In this optimization the fiber angles of each of the 13 plies were used as the design variables and were free to move between -90 and 90 degrees. The genetic algorithm evolutionarily drove these angles to values that resulted in a maximum distance of the closest modal frequency to a keep out frequency. The result was that the first ten modal frequencies of the wing were at least 100 Hz away from any of the ten keep out frequencies. Figure 5-6 shows the progression of this optimization as it drove to an optimum design.

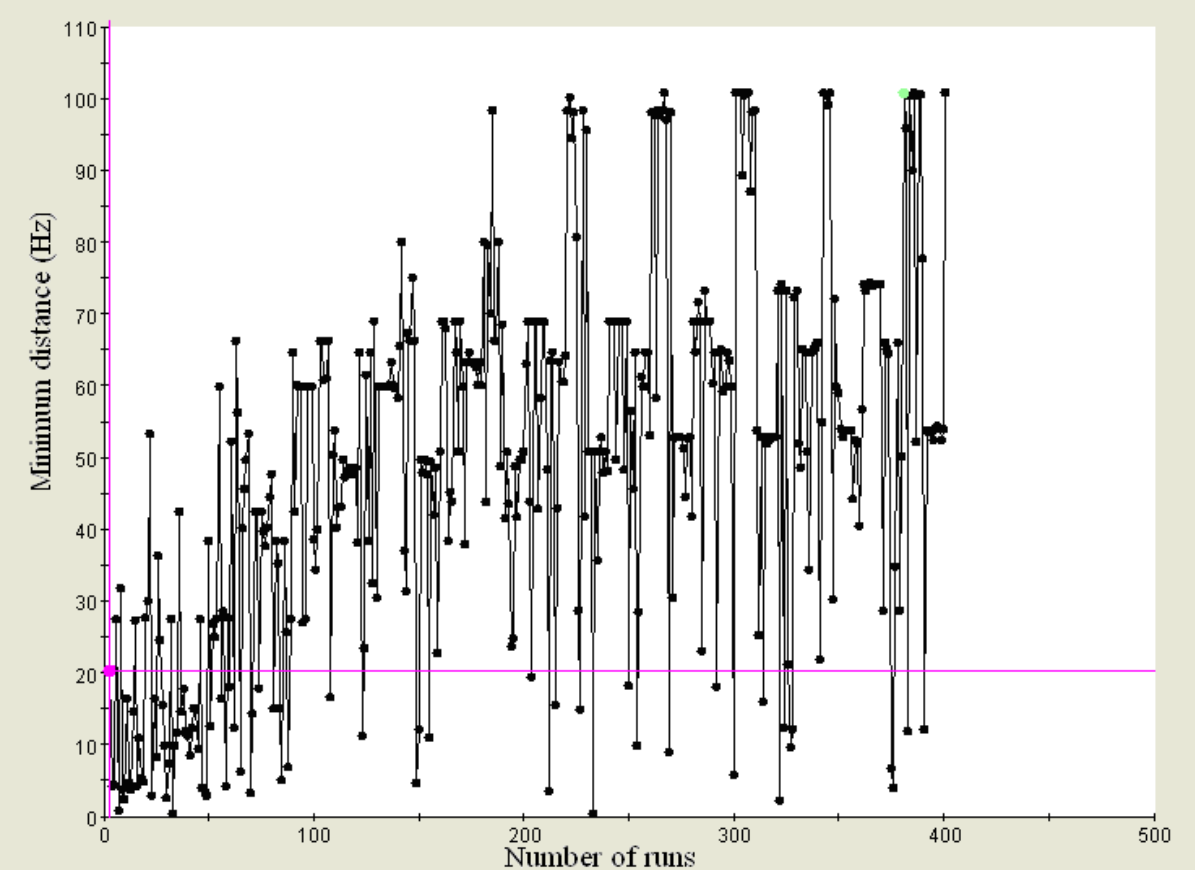

Figure 5-4: A graph of the genetic algorithm's progression

6 CONCLUSIONS

Custom applications that directly access a geometry kernel to perform mathematical computations will run faster than custom applications that call API functions that operate a level or two above the geometry kernel. This is shown by the fact that applying CAD independent algorithms in the creation of custom NX applications allows for the application to be faster (roughly 30X) than similar programs written solely in the NX Open C API. The reason for this is the fact that the NX Open C API functions operate at a level or two above the geometry kernel and are therefore slower in performing geometrical computation when compared with GSNLib and CAA RADE which interface directly with the geometry. Therefore, integrating a CAD independent geometry kernel such as GSNLib with a custom NX application to perform geometry calculations can significantly improve run time. In the same light, making direct calls to the NX Parasolid kernel should also significantly improve run time. In addition, it was shown that integrating a CAD independent geometry kernel with a custom CATIA application will not generally improve run time because CAA RADE allows direct access to the geometry kernel. Therefore, custom applications that directly access a geometry kernel to perform mathematical computations will run faster than custom applications that call API functions that operate a level above the geometry kernel.

Utilizing a generic geometry kernel to create custom geometry applications allow those applications to be portable between several dissimilar software packages. Doing this allows the custom applications to be run within multiple software packages allowing the application to be

written only once, and not rewritten for every individual software package. It also allows custom applications to stay within user familiar software packages, thus keeping the benefits of the original software package. The portability of this method is proven by the fact that the same CAD independent code was used in both NX and CATIA for geometry creation, utilizing their individual CAD APIs merely for translational purposes.

The method presented in this thesis to automate laminate composite finite element models creates models that are more detailed than those made with zone based methods. Because of the greater detail in the model, this method will likely yield more accurate analytical results in models with rapidly changing geometry (such as the model shown in Figure 5-5) than other similar models created with zones based methods. This method also allows for the creation of a tool that creates laminate composite finite element models within seconds and will save engineers, who are doing this manually, dozens of hours of work per model. In addition, the automated composite finite element tool can be integrated into an optimization framework, used in conjunction with a method to automatically apply boundary conditions, to create an effective optimization of a laminate composite part.

6.1 Recommendations

One current limitation to the automated laminate composite design and analysis methods is that the input part geometry is limited to two opposing NURBS surfaces. A major improvement would be to automate the creation of ply lay-ups for parts that are defined with more than two NURBS surfaces or with even with a solid model. This improvement would allow for the design and analysis automation to be applied to a larger variety of composite parts.

Another limitation specifically in the automated laminate composite analysis method is that the current method does not account for the drape of layers in the laminate. For surfaces with geometry whose curvature changes in *u* and *v* directions, local lamina fiber angles change when the lamina is laid down. This, in turn, changes the directional strength properties of the material. Therefore accounting for the drape of each lamina in the stack improves the analytical results. An improvement to the current method would be the addition of a method that calculates the new local fiber angles for each lamina and applies these calculations to the finite element model.

REFERENCES

[1] ANSYS APDL Programmer's Guide. http://uic.edu/depts/accc/software/ansys/html/prog_55/g-apdl/AS1.htm

[2] Astle, T. L.: System Architecture and Development of CAD Independent Algorithms for Integration with Commercial CAD Software, M.S. Thesis, Brigham Young University, Provo, UT, 2003.

[3] Beasly, D.; Bull, D.; Martin, R.: An Overview of Genetic Algorithms: Part 1, Fundamentals, University Computing, volume 15(2), pages 58-69, 1993.

[4] Bruyneel, M.: A General and Effective Approach for the Optimal Design of Fiber Reinforced Composite Structures, Composites Science and Technology, Volume 66, Issue 10, pp 1303-1314, 2005.

[5] Delap, D.: CAD-based Creation and Optimization of a Gas Turbine Flowpath Module with Multiple Parameterizations, M.S. Thesis, Brigham Young University, Provo, UT, 2003.

[6] Elliott, J.: An Automated Approach to Feature-Based Design for Reusable Parameter-Rich Surface Models, Computer-Aided Design & Applications, Vol. 4, Nos. 1-4, 2009, pp 497-507

[7] Farin, G.: Curves and Surfaces for Computer Aided Geometric Design, Academic Press, Inc., San Diego, CA, 1988.

[8] Gay D.; Hoa, S.: Composite Materials Design and Applications, Taylor and Francis Group, Boca Raton, FL, 2007.

[9] Gibson, R. F.: Principles of Composite Material Mechanics, CRC Press, Boca Rotan, FL, 2007.

[10] Gurdal, Z.; Haftka, H.; Hajela, P.: Design and optimization of laminated composite materials, John Wiley & Sons, Canada, 1999.

[11] Hale, R; Schueler K.: Knowledge-Based Software Systems for Composite Design, Analysis and Manufacturing, Society of Automotive Engineers, 2001.

[12] Hepworth, A. I.; Jensen, C. G.; Roach, J. T.: A CAD Independent Approach to Automate Laminate Composite Design and Analysis, Computer-Aided Design & Applications, 6(2), 147-156, 2009.

[13] Hepworth, A. I.; Jensen, C. G.; Roach, J. T.: Methods to Streamline Laminate Composite Analysis and Optimization, Accepted for publication in Computer-Aided Design & Applications, 2010.

[14] Hull, D.; Clyne, T.: An Introduction to Composite Materials, Cambridge University Press, Australia, 1996.

[15] Menayo, G.; Quero, A.: Computer-Aided Method of Obtaining a Ply Model of a Composite Component. U.S. Patent App. 731,794, 2007.

[16] Mitchell, Melanie: An Introduction to Genetic Algorithms, MIT Press, 1998.

[17] Peters, S.: Handbook of Composites, Chapman & Hall, Great Britain, 1998.

[18] Piegl, L.; Tiller, W.: The NURBS Book, Springer-Verlag, Berlin, Heidelberg, New York, 1997.

[19] Prakash, B.: AUTOLAY – A GUI Based Design and Development Software for Laminated Composite Components, Computers & Graphics 23 95-110, 1999.

[20] Rogers, David F.; An Introduction to NURBS: with Historical Perspective, Academic Press, San Francisco, 2001.

[21] Scott, N.: High-Level, Product Type-Specific Programmatic Operations for Streamlining Associative Computer-Aided Design, M.S. Thesis, Brigham Young University, Provo, UT, 2008.

[22] Sederberg, T.: BYU NURBS, http://cagd.cs.byu.edu/~557/text/ch1.pdf, 2007.

[23] Siemens Corp., http://www.plm.automation.siemens.com/en_us/Images/10651_tcm1023-4449.pdf, 2008.

[24] Solid Modeling Solutions, http://www.smlib.com/gsnlib.html, 2009.

[25] Strong, A. B.: Fundamentals of Composites Manufacturing Materials, Methods, and Applications, Society of Manufacturing Engineers, USA, 2008.

[26] Thede, S.: An Introduction to Genetic Algorithms, Consortium for Computing Sciences in Colleges, pages 115-123, 2004.

[27] Vasey-Glandon, VM, & Kunkee, D.: Knowledge driven composite design optimization process and system therefore. U.S. Patent No. 7,010,472. 2006.

[28] Weiss, Pete: Long Island eyes growth in composite materials market, Long Island Business News, January 25, 2008.

[29] Zeid, I.: Mastering CAD/CAM, McGraw-Hill, New York, NY, 2005.