2010-12-08

Automated Tool Design for Complex Free-Form Components

Kevin G. Foster
Brigham Young University - Provo

Automated Tool Design for Complex Free-Form Components

Kevin G. Foster

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 A. Mattson
Michael P. Miles

Department of Mechanical Engineering

Brigham Young University

April 2011

ABSTRACT

Automated Tool Design for Complex Free-Form Components

Kevin G. Foster

Department of Mechanical Engineering

Master of Science

In today's competitive manufacturing industries, companies strive to reduce manufacturing development costs and lead times in hopes of reducing costs and capturing more market share from early release of their new or redesigned products. Tooling lead time constraints are some of the more significant challenges facing product development of advanced free-form components. This is especially true for complex designs in which large dies, molds or other large forming tools are required. The lead time for tooling, in general, consists of three main components; material acquisition, tool design and engineering, and tool manufacturing. Lead times for material acquisition and tool manufacture are normally a function of vendor/outsourcing constraints, manufacturing techniques and complexity of tooling being produced. The tool design and engineering component is a function of available manpower, engineering expertise, type of design problem (initial design or redesign of tooling), and complexity of the design problem.

To reduce the tool design/engineering lead time, many engineering groups have implemented Computer-Aided Design, Engineering, and Manufacturing (CAD/CAE/CAM or CAx) tools as their standard practice for the design and analysis of their products. Although the predictive capabilities are efficient, using CAx tools to expedite advanced die design is time consuming due to the free-form nature and complexity of the desired part geometry. Design iterations can consume large quantities of time and money, thus driving profit margins down or even being infeasible from a cost and schedule standpoint. Any savings based on a reduction in time are desired so long as quality is not sacrificed.

This thesis presents an automated tool design methodology that integrates state-of-the-art numerical surface fitting methods with commercially available CAD/CAE/CAM technologies and optimization software. The intent is to virtually create tooling wherein work-piece geometries have been optimized producing products that capture accurate design intent. Results show a significant reduction in design/engineering tool development time. This is due to the

integration and automation of associative tooling surfaces automatically derived from the known final design intent geometry. Because this approach extends commercially available CAx tools, this thesis can be used as a blueprint for any automotive or aerospace tooling need to eliminate significant time and costs from the manufacture of complex free-form components.

Keywords: complex free form surfaces, multidisciplinary optimization, generative parametrics, automated tool design, response surface methodology

ACKNOWLEDGMENTS

I want to express my thanks to Dr. C. Greg Jensen for the professional guidance and direction that he has offered me over the years as both my undergraduate and graduate advisor He is one of those rare individuals that drives you to be a better person and accomplish more than you thought previously possible. Were it not for him I probably would not be where I am today. I'd also like to thank my friends and fellow graduate students with whom I worked with in the BYU ParaCAD lab for all the fun memories and assistance in developing this research. Many hours were spent in discussion with them as to how I can develop efficient algorithms and overcome the challenges encountered in developing this research. I'd like to thank Pratt & Whitney in East Hartford Connecticut for funding this research and making it all possible. I'd also like to thank Jason Elliott and Michael Weiss of the Pratt & Whitney Hollow Fan Blade engineering staff for their engineering expertise and guidance. I would also like to thank Scientific Forming Technologies (Ms. Misty Engelbrecht and Dr. Wu specifically) for their generosity in granting me many licenses of their DEFORM 2D forging simulation software to complete the research. I couldn't have chosen a better company to promote their software. I'd like to thank all the family and friends for their support and constant reminders to finish this research. They have provided me with more than I can provide in return.

I would like to express my profound thanks, appreciation, and love to my beautiful wife, Lisa. I thank her for all her understanding and support throughout the duration of my project. She has truly stood by my side and provided me with the needed motivation to see this research to completion. I love her and am grateful to have her as my eternal companion.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

To conceptualize, design, and manufacture today's complex products in increasingly competitive and efficient markets there is a push causing today's technology companies to explore new ways to improve designs, increase productivity, and reduce costs. The engineering tools available to engineers are always being improved to help satisfy these demands. Today software such as parametric CAD systems, finite element analysis (FEA), computation fluid dynamics (CFD), and other CAE software systems have been developed to meet the needs of companies striving to improve designs, increase productivity, and reduce costs.

Today, for example, designing products such as an aircraft jet engine or the Reusable Solid Rocket Motors (RSRM) for NASA's Space Shuttle or future heavy lift vehicle require detailed input and collaboration from several different disciplines**.** Those disciplines encompass aerodynamics, structural design, thermo sciences, and manufacturing to name a few. In the past, experts in these disciplines worked in small teams specialized in handling the integration process. Today engineering groups are working to more fully implement what is called "concurrent engineering." Concurrent engineering moves different engineering groups away from the "over the wall" or sequential design approach to a multi-disciplinary design approach centered on Integrated Product Design (IPT's) teams (Hogge 2002). IPT's in industry strive to discover and design the optimal scenario within the allotted time (schedule) and budget (cost). Using a multi-disciplinary design optimization (MDO) approach to aid in convergence, an

optimal solution can be achieved by implementing a software concurrent-engineering approach paradigm.

1.1 Problem Statement

In today's manufacturing markets, companies strive to reduce development, design, and manufacturing lead times in hopes of capturing more market share from early release of their new or redesigned products. One of the more significant and time consuming challenges of product development is tool design. This is especially true for large stamping dies, forging dies, and forming molds.

The lead time for tooling, in general, is comprised of three components; material acquisition, tool design/engineering and tool manufacturing. Lead times for the material acquisition and tool manufacture component are typically fixed due to vendor/outsourcing constraints, manufacturing techniques and complexity of tooling being produced. However, the design/engineering component is a function of available manpower, engineering expertise, type of design problem (initial design or redesign of tooling), design methods employed, and complexity. To reduce the tool design/engineering lead time many engineering groups have implemented Computer-Aided Design, Engineering, and Manufacturing (CAD/CAM/CAE or CAx) tools as their standard practice to assist the design and analysis of their tooling products.

Although the predictive capabilities are efficient, using CAx tools to expedite finished forming die surface design is an inherently time consuming process due to the cyclical nature of the design iterations that must be made to meet and deliver original design intent. These design iterations can consume enormous amounts of time and money. Any time saving measures are typically welcome so long as sufficient quality is achieved.

This thesis research will explore the issues and limitations described above by answering the following questions:

1. Can the integration of parametric CAD, meshing capabilities, bulk forming simulation, and optimization accomplish a realistic part/die models prediction when the parts being formed are defined using complex free-form surfaces?
2. Can numerical surface construction and interpolation techniques be used to accurately allow for an objective rating on the accuracy of predictive results compared to the original design intent?
3. Can the implementation of this methodology (the integration and automation of associative tooling surfaces automatically derived from the original part geometry) produce a significant reduction in design/engineering tool development lead time?
4. Can this methodology be used as a blue print for any automotive or aerospace tooling industry to eliminate significant time and costs from the manufacture/design of complex free-form components?

1.2 Thesis Objective

The objective of this thesis is to develop a conceptual automated tooling design scheme methodology that integrates commercial CAD/CAM/CAE technologies. These technologies will be coupled with numerical free-form surface construction, interpolation methods, and an optimization engine. The intent is to obtain tooling and pre-formed work piece geometries optimized to produce near-to-design intent products for in-process manufacturing geometry.

1.3 Delimitations of the Problem

The objective of this research is to validate the integration of CAD/CAM/CAE tools for the purpose of developing an automated tool design scheme that is integrated with an optimization engine. This allows for determination of the optimal parameter values for the desired topology. This method was tested by implementing two test cases. Test cases 1 and 2 will be high fidelity complex free-form die-work piece surface components. The major difference between the test cases is the manufacturing methods used to obtain the desired geometry (i.e. double vs. single sided machining). The differences are explained in detail in Section 5.1. Both test cases will implement the aforementioned methodology, for a jet turbine engine compressor systems' first stage shroudless hollow fan blade. Determination of the original design intent is not deduced rather provided as a reference as to the validity of the automated tool design method output. Micro shape (specific design feature) optimization of the forging dies and work pieces will not be performed rather the questions posed in Section 1.1 will be addressed.

CHAPTER 2: LITERATURE REVIEW

This chapter provides a review of past and currently accepted engineering and computer science oriented literature that provides a basis for the methodology. The surveyed literature is intended to establish a background for the reader regarding what research has been conducted in fields closely related to this thesis, and how previous research has been built upon and implemented in this thesis. The following literature review will also exploit the voids wherein this thesis conducts pertinent research.

From the problems discussed in the previous chapter, this research requires an investigation of the following topics:

- Computer Aided Design (CAD) (Section 2.1)
- CAD Application Program Interfaces (API's) (Section 2.2)
- Computer-Aided Engineering (CAE) (Section 2.3)
- Surface Interpolation Methods (Section 2.4)
- Multidisciplinary Optimization (Section 2.5)
- Statistical Response Surface Modeling (Section 2.6)

2.1 Computer Aided Design (CAD) - Parametrics

The use of CAD began in the mid 1960's as an aid for drafting. It has since become a large contributor to shortened design times and increased product quality due to its ability to assist in the creation, modification, analysis and optimization of a design (Lee 1999, Hsu et al.

1992). The ability to virtually conduct concept generation to concept finalization has led to large advancements compared to the products of the pre-CAD era.

From its initiation, CAD has promised five important benefits to the engineering design process (Dieter 2000, Liker et al. 1992).

- Automation of routine design tasks
- Ability to design in three dimensions
- Design by solid modeling to create digital geometric databases which permit downstream analysis and simulation
- Electronic transfer of design database to manufacturing (CAD/CAM) where it is used to create NC tool paths for machining and quick digital transfer for rapid prototyping
- A paperless design process

As the fundamental tool for modeling geometric systems, CAD has progressed significantly by developing squarely on top of wireframe modeling, surface modeling, solid modeling, and the current feature-based parametric modeling systems while enabling the successful implementation of the benefits listed above as envisioned by its creators. This research relies heavily upon all the different types of CAD.

In today's CAD tools, capabilities exist to provide geometric modeling systems for manipulating and constructing shapes, to specialized application programs used for analysis and optimization (CAD-based Master Models). The following sub-sections provide background information for the methods implemented in this research.

2.1.1 Wireframe Modeling Systems

Wireframe modeling systems display visual representation of geometric entities by displaying a wireframe (points and line) drawing of the shape. These systems store the wireframe representation as mathematical descriptions in the form of lists of curve equations, coordinates of the points, and the connectivity information for the shape's curves and points. The connectivity information describes which points belong to which curves and to which entities the curve/point sets are adjacent to.

These systems lent themselves for ease of use due to the simple inputs required to create a shape and the relative ease for generating such a system independent of a commercial vendor. Wireframe modeling systems do have their pitfalls. Since the visual model is composed of lines and points, it is oftentimes ambiguous to determine the orientation and overall visual description (no features for depth perception). These systems also contain no information pertaining to the inside and outside boundary faces of the objects thus rendering it impossible to determine mass properties, derive tool path automation possibilities, or generate a mesh for finite element analysis (Lee 1999).

For this research, wireframe modeling is the foundation for all the higher level modeling that is required. All the construction curves for the dies and workpiece a wireframe based. For more information on wireframe modeling see Mäntylä (1988) and Lee (1999).

2.1.2 Surface Modeling Systems

Surface modeling systems build upon the technology of a wireframe system and add the capability to store surface equation and connectivity information. Surfaces can be created by interpolating points, interpolating a mesh of curves, or by translating/rotating a curve.

Surface modeling systems are typically used to create complex structured and free-form surfaces for visualization and machining purposes. Surfaces can also be used by finite element analysis packages for determining proper mesh generation. By using the underlying mathematical descriptions of surfaces, numerically controlled (NC) tool paths can be generated for the surface objects. Although surface modeling systems overcome many weaknesses of wireframe systems there still exist weaknesses such as the inability to calculate material volume information due to ambiguity in determining if a surface grouping forms a closed volume (Lee 1999).

This research uses surface modeling as the main technique for generating the die surfaces and the workpiece geometry. The surfaces provide the optimal starting point for the FE mesh to be applied for simulation. For more information on surface modeling techniques see Choi (1991), Mortenson (1985), Su et al. (1989), and Hosaka (1992).

2.1.3 Solid Modeling Systems

Solid modeling systems build on top of wireframe and surface modeling capabilities. Solid modeling systems provide the possibility to provide information about what is inside the 3D model as well as information about the surface of the object. Solid modeling systems also have the capability to calculate mass properties, generate Finite Element (FE) meshes, model kinematic motion, model collision detection, and model 3D NC tool paths.

To define the interior and exterior of a solid object mathematically, a point set Q in 3D Euclidean space (E^3) is defined. The interior of an object and associated boundary can be defined as the set iQ and bQ, respectively, then we can write.

$$Q = iQ \cup bQ \tag{1}$$

If the exterior of Q is defined to be cQ, then

$$W = iQ \cup bQ \cup cQ \tag{2}$$

where W becomes all possible points in the E^3 3D space (Zied 2005). Figure 1 shows a visual depiction of the above mathematical descriptions.

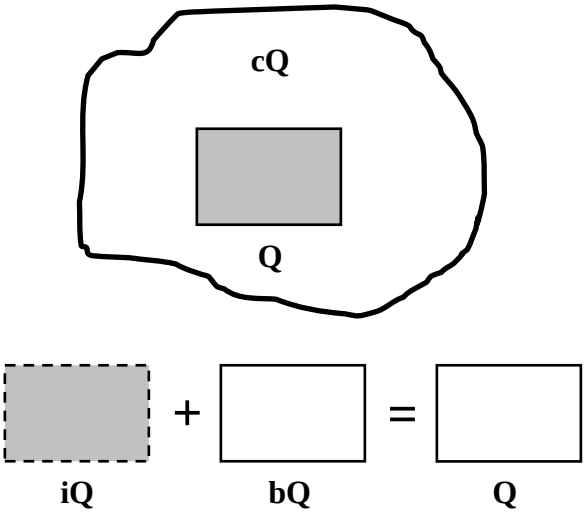


Figure 1: Mathematical Definition of a Closed Solid

There are two basic types of solid modeling techniques, Constructive Solid Geometry (CSG) and Boundary Representation (B-Rep). With CSG the solid is constructed in a building-block fashion by combining primitive shapes such as a sphere, cube, cylinder, cone or sphere. These primitives can be combined by use of Boolean operations such as subtraction, union, and intersection. Figure 2 shows the various combinatorial methods for different primitives.

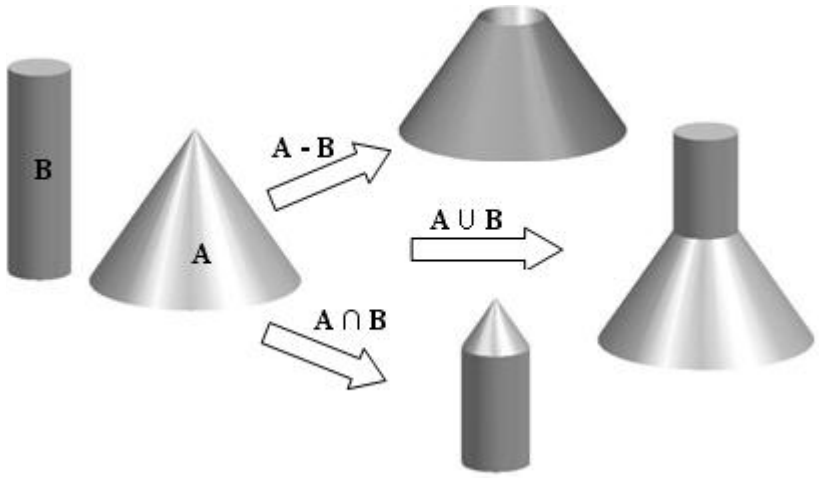


Figure 2: Example of Subtraction, Union, and Intersection Boolean Operations

In B-rep models, solids are represented by sets of faces that form an air tight volume as shown in Figure 3.

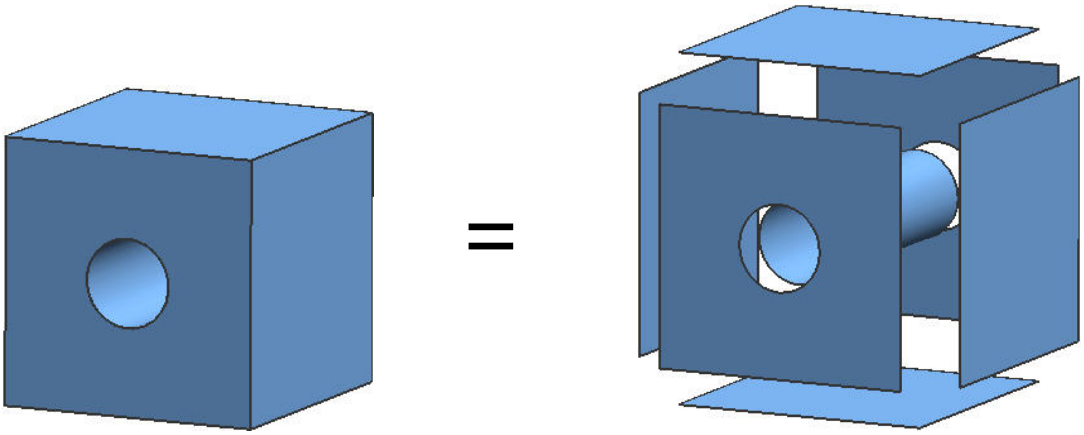


Figure 3: Boundary Representation (B-rep)

These faces are regions or subsets of closed and orientable faces (possible to distinguish between surface normals pointing inside or outside). To ensure that a B-rep model is valid

topologically, it needs to satisfy the condition known as Euler-Poincaré relationship which is defined as the relationship

$$F - E + V - L = 2(B - G) \tag{3}$$

where the number of faces (F), edges (E), vertices (V), inner loops of faces (L), bodies (B) and through holes (G) must satisfy the condition. Since B-rep modeling creates topologically valid geometry, the CAD software database must store information about the connectivity of faces and the equations defining he geometry faces. A B-rep model is useful for complex parts that cannot be modeled conveniently with primitive shapes. It has also be shown in today's B-rep modeling capable CAD packages that once topology has been defined, many different operations can be performed to adjust the geometry without changing the basic topology because it defines the part topology and geometry separately (Lee 1999, Dieter 2000, Zied 2005). B-rep modeling will be used in this research to construct the workpiece geometry's closed volume.

For more detailed references about solid modeling and associated techniques see Mäntylä (1988), Requicha et al. (1992), Rossignac et al.(1991), and Mortenson (1985).

2.1.4 Feature-based Parametric Modeling

It has been said that commercial CAD systems would become the soul source for geometry construction and manipulation for multidisciplinary optimization applications (Samareh 2001). In order for that capability to become a reality, the development of parametrics was needed. Parametrics enable easy modification and reuse of CAD models. Parametric systems were first introduced by Parametric Technology's Pro/Engineer in the late 1980's and have since then been embraced by all commercially available CAD software (Hoffman et al. 2001). Parametrics permit efficient testing of "what if" scenarios during product design in

search of the best design. Parametrics also allow the models to be defined by parameters that are linked to shapes or operations such as constraint relationships, geometry constructs, dimensions, and various design features. These associations allow for the parametric information to flow downstream in the models. This allows the geometry to be updatable when a parameter must be altered. This parametric modeling ability negates the complete reconstruction of the model when change is necessary. These parameters can be stored either internally in the CAD package or externally in the form of text files or spreadsheets. Parametrics also support the notion of model reuse by allowing designers to create a family of parts. For a through, comprehensive summary of the impact that parametrics has had on solid modeling techniques see Anderl and Mendgen (1995)

Another advantage of feature-based parametrics is the capability to have the information on the existence, size, and location of manufacturing features enabling automatic generation of process plans from a model (Lee 1999).

For parametric modeling to be a success, a proper parameterization scheme must be chosen such that during the morphing and regeneration of the model to the desired parameters values, model failure and corruption does not occur (Hoffman et al. 2001). Elliott (2004) stated that for complex surface geometry with large number of parameters, no solution exists to allow ease of parameterization. Figure 4 shows the basic principle of parametrics and the potential flexibility of the model being designed.

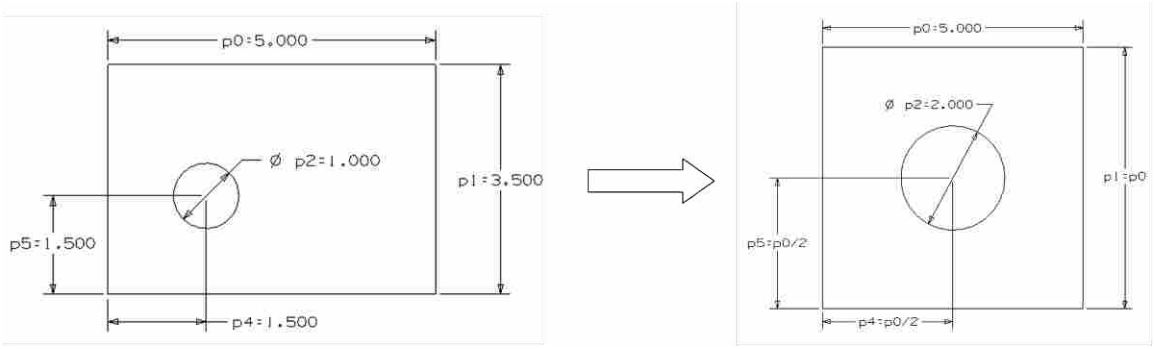


Figure 4: An Example of Parametrics (King 2004)

Failures in the regeneration of a CAD model can be traced to several factors such as kernel errors, bugs in the CAD software, or poor parameterization practices (Hogge 2002). Where update failures and part corruption has a large rate of occurrence, an alternative solution for parametrics is needed, such as generative parametrics. Generative parametric modeling is the automatic reconstruction of a complex model based on the altered parameter values avoiding update errors and file corruption. Parametric and generative parametric modeling schemes can be used in conjunction with wireframe, surface, solid, and feature-based modeling and are used as the modeling schemes of choice for this research. For a more in depth look at feature-based parametric modeling see Shah et al. (1995), Hoffman (2005), and Allada et al. (1995).

2.1.5 CAD-based Master Models

When designers are faced with the task of building models that will flow downstream to various differing disciplines, a daunting task suddenly looms that requires proper model fidelity and construction. This model becomes the center of what is called a "CAD-centric" design process. It is a model that stands at the center of the design process and directs product

definition and management (Elliott 2004). This model is frequently called a product "master model." Hoffman et al. (1998) defined a product master model as follows:

> "The master model is an object-oriented repository that provides essential mechanisms for maintaining the integrity and consistency of the deposited information structures."

Since companies typically have many departments (including design, manufacturing, marketing, and subcontractors) it becomes a necessity to have a template file that provides CAD data in a consistent way for all groups to use.

The concept of a master model can be applied by an assembly, or used for a single part. The characteristics of master models have wide spread use in product design. It has been shown by Hogge (2002) and Delap (2003) that CAD-based design can have profound success in multidisciplinary optimization schemes by imbedding all the necessary attributes, model smoothness parameters, continuity parameters, and data for generation/modification of downstream applications or disciplines. It has also been stated that any defeaturing or massaging of the CAD master models to suit needs of analysis, visualization or manufacturing is facilitated within the domain of the master model (Elliott 2004, King 2004).

The advantages of using a CAD-based master model in CAD-centric design, analysis and optimization have been seen in solid modeling applications. However, when using high fidelity parameter rich surface models as the master model for design and analysis loops, this method has not yet been explored. If this capability existed and were widely implemented, an "Intelligent Product Data Management" system would exist that could explicitly identify and maintain discipline model interdependencies, provide immediate feedback on change impacts, and support multidisciplinary view of requirements, function allocation and behavior of the systems involved (Waterbury 1999).

2.2 CAD Application Program Interfaces (API)

Recognized as a powerful design tool in the late 1970's (Requicha 1980), CAD API's were developed to facilitate programmatic creation and manipulation of CAD models using high level programming languages (i.e. C,C++, Java). An API is a library of functions that is utilized in a programming environment, and allows access to the core functionality of the software (King 2004).

Zied (2000) stated that traditional parametrics create opportunities to describe flexible designs but tend to allow parameters to conflict with each other at some time during the modeling process. Zied also said that the designer must resolve the conflicts before moving on. Using API's allows for program specific error handling, debugging, access to object oriented structures, file I/O, request of user input, and ability to handle more complex designs (Ramsaswamy 1993, Magalhaes 2004).

It has been recognized that there exists three types of API's: macros, program specific languages, high-level programming language with specific functions (Hogge 2002). The combination of the various types of API's enable users to develop customized programs to dynamically create and analyze models in batch operations such that process automation can be achieved.

Macros are used in several programs ranging from word processors to CAD/CAE/CAM software. A macro is a simple string of commands that record a sequence of actions that the user desires be performed. Macros are among the easiest of the types of API's to learn. Macros are good to use to automate repetitive tasks. Some macros are recorded automatically as key stokes and mouse clicks are used with graphical user interfaces (GUI's).

Program specific languages are typically only native to a specific program for which it has been developed. A common example of a program specific language is that developed specifically for ANSYS, the ANSYS Parametric Design Language (APDL). The APDL allows access to most everything available interactively but with access to the core functionality of the ANSYS environment in a programmatic sense (Rohm 2000).

The most robust level for an API is the high-level programming language with specific functions. This type of API incorporates high-level programming languages such as C/C++, Visual Basic, C# or Java as the programming languages of choice. High-level languages are typically used in third generation CAD systems such as Dassault' CATIA or Siemens' NX. These APIs are the most difficult to use since the user must have a working knowledge of not only the API toolkit and its interactive functions but also the programming language and associated syntax necessary to communicate with the program and outside resources. One of the benefits of high-level languages is that object-oriented concepts can easily be incorporated into the custom API programs. Also, APIs from multiple software programs can be utilized simultaneously, which assists in integrating multiple programs.

Rohm (2001) mentioned two disadvantages of creating custom CAD API programs: 1) the development time is considerably larger than it would be to create interactive models 2) the developer must know the API of the system. Though there are disadvantages, the advantages outweigh them when families of parts need to be created. First, multiple models can be created in very little time using a common model parameterization scheme. Secondly, because the models are created programmatically, they lend themselves to use in optimization routines with error checking. Finally, the API allows users to create highly customized applications within commercial CAD software.

Examples of the usage of API's can be found in many areas of industry. Rohm et al. (2000) used API languages show that instead of recreating various models for jet engine components when parameters are manipulated, that a program can be written that would recreate the models every time it is executed. The main benefit advertised was that model creation cycle time was taken from "months to minutes".

Ardalan (2000) developed a spacecraft design system using SolidWorks API. An interface was created that allowed a user that was familiar with spacecraft design but unfamiliar with the CAD system to easily create a 3-D model within the CAD package. Wilson (2004) successfully used the Siemens NX API (UG/Open) to perform direct slicing of CAD models for use in the rapid prototyping industry. This allowed for a more parametric approach to be implemented and enabled rapid prototyping to occur from the CAD system. Astle (2003) developed CAD system-independent geometric algorithms for flank milling impellers for turbo machinery. This allowed simplified portability between systems for connecting to various CAD systems.

Of the works cited on API CAD programming, Elliott (2004) found that only few articles show work relating to Parameter Rich Surface Model (PRSM) features inside a parametric CAD-based environment. He showed that research had been done to automatically generate and manipulate spline points, curves, and surfaces to reduce model parameter count. He also found that along with looking at an increased number of parameters within a programmatic environment for surface applications, the number of design variables used was impressive but it still fell short of typical PRSM parameter sets (Tang et al. 2001, Haimes et al. 2003). Elliott's research showed that when working with PRSM's he was able to accomplish the following six points:

- CAD models from large data sets can be created in a seamless integration of text file data and different part models.
- Programmatic control of the variation of input parameters is possible to successfully create feasible models within a pre-described design envelope.
- Complex free-form surface geometry with different parameterization schemes and multiple constraints were generated.
- Reusable surface models with high-fidelity, mesh-worthy geometry were created.
- Current practices were improved upon by overcoming obstacles present in industry design processes.
- Two part files were controlled simultaneously to programmatically transfer data.

As can be seen in the available literature, the benefits of using API's in design, analysis, and optimization will be demonstrated in this thesis. The API's from NX, ANSYS, and DEFORM are implemented in this research. The NX API is of the high-level programming language type and is used to generate all the model geometry and also used in the evaluation process to determine surface deviations. The API's from ANSYS and DEFORM are of the program specific type. The ANSYS API is used to develop a program that can logically develop a mesh suitable for simulation. The DEFORM API is used to automate the simulation process. These API methods are combined to handle a high number of parameter based PRSM's for the CAD-centric methodologies researched.

2.3 Computer-Aided Engineering (CAE)

Computer-aided engineering is the process of allowing a computer to analyze geometry created by CAD or an analysis software package (Lee 1999). CAE allows engineers and

designers to virtually design and analyze products before manufacture is begun to aid in the determination of an optimal design. It uses empirical and analytical based methods to predict physical phenomena. CAE software is based on the finite element method (FEM), known also as finite element analysis (FEA). CAE software generally exists as either one or a combination of the following portions of the FEM : preprocessing, solving, and postprocessing CAE software exists for many disciplines such as thermodynamics, heat transfer, kinematics, mechanical design, fluid dynamics, and structural dynamics to name a few.

Combining the three main portions of FEA for CAE allows users to develop models for predicting reality. The predictive capability of CAE tools has progressed to the point where much of the design verification is now done using computer simulations rather than physical prototype testing (Raphael et al. 2003).

2.3.1 Finite Element Analysis

Finite element analysis (FEA) is the most dominant method available today for simulating model behavior for CAE. FEA approximates governing differential equations into a large set of linear algebraic equations that can be solved on a computer (Balling 2001). The FE method began to gain popularity in the 1960's in the field of structural engineering. It has since expanded into other such areas as those mentioned in section 2.3. Most industries use FEA software on a regular basis to calculate everything from effective stresses and strains to pressures experienced during forming processes. FEA plays an important role in the successful implementation of this research. With FEA the forming simulations can be executed and results obtained to determine pertinent parameter settings for final model topology. For additional information on how the finite element method has been developed and implemented see Balling (2006).

2.3.1.1 FEA Preprocessing

Mesh generation software aids in the first step of FEA by dividing the geometry under investigation into a collection of geometrically simple subdomains called finite elements or elements. A collection of these elements is called a grid or mesh (Reddy 1985, King 2004). The discretization of the domain provides the necessary input to numerical solver software such as nodal coordinates, element connectivity, element properties, and support data. In order to simulate reality, load data, and constraints need to be in place to allow the object simulations to be represented as they occur in nature. Loadings can be represented as forces (point and distributed), pressures, prestressed loading conditions, and moment loading to name a few. Constraints can take on the form of restricting translation, rotation, temperature, velocities, and voltage, etc. For a more comprehensive list of load and constraint types see the ANSYS documentation.

2.3.1.2 FEA Solving

Once geometry has been preprocessed, the software generates systems of equations that relate the boundary conditions to the unknowns such as displacement, temperatures, velocities, etc. Using a numerical solver the systems of equation are solved for the unknowns (Balling 2006).

2.3.1.3 FEA Postprocessing

The final step when using FEA is calculating the element results from the results of the systems for equations previously solved. An important process for those that are more visual by

nature is visualizing and reviewing the output from the solver which is numerical in nature. This has been coined postprocessing of the FEA solution (Balling 2006).

2.4 Surface Interpolation Methods

Surface interpolation can be understood to mean more than one thing. To delineate between the two meanings used in this thesis the following terminology will be used: surface construction interpolation and surface deviation interpolation.

When dealing with surface construction, surface interpolation is typically referred to as surface fitting (Piegel 1997). Fitting with an interpolation construction scheme, curves or surfaces are fitted precisely through given data, i.e. XYZ point coordinates, with assumed derivatives of the curves at the data point locations. This data is typically structured such that a numerical method or CAD package can develop a control net of curves that can be fitted with a surface (Farin 1988). Surfaces, typical in 3rd generation CAD products, are constructed from uniform B-splines, non-uniform rational B-spline (NURBS) or Bezier curves to name a few. For more information on the background and development of these curves and surface types, see Piegel et al. (1997), Elliott (2004), and Sederberg (2007).

The second type of surface interpolation can be referred to as calculating surface distance deviations between two arbitrary surfaces. To determine the deviations from a numerical standpoint requires knowledge of surface normals (or desired direction vector) at known data point locations that exist on the surface. If surface normals are desired, knowledge of two surface tangent vectors must be known at the known data point. These tangent vectors must be at some angle θ to each other. Taking the cross-product (see Figure 5) of the two vectors will yield the normal belonging to the surface at the known point.

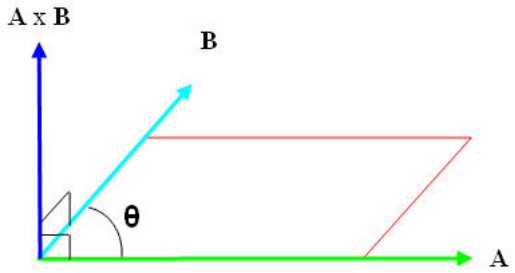


Figure 5: Surface Normal Calculation with Cross-product

With these surface normals, the distance to the desired surface could be calculated by interpolating the intersection point of the surface normal and surface to which the deviation calculation is desired.

Surface construction interpolation and surface deviation interpolation are both used heavily with this research. Knowing how to calculate surface normal vectors is used to allow intersection algorithms to properly identify the point data locations for deviation calculations. These methods are critical to the success of this research and allow for the evaluation of the simulation results and testing their validity.

2.4.1 Non-Uniform Rational B-splines (NURBS)

In parametric feature-based design, it is common place to use B-splines or NURBS to define the free-form surfaces using surface fitting techniques developed in third generation CAD systems. The definition of a NURBS as defined by Piegel (1997) is given as

$$C(u) = \frac{\sum_{i=0}^{n} w_i N_{i,p}(u) P_i}{\sum_{i=0}^{n} w_i N_{i,p}(u)}, \tag{4}$$

with

P_i control points,
$N_{i,p}$ basis functions,
w_i weights,
u parameters, with $0 \leq u \leq 1$, and
n number of control points - 1.

The number of control points set by n dictates the number of weights, w_i, associated with the same number of control points P_i. A NURBS curve has a knot vector containing n+p+2 knots for the curve where p is equal to the degree of the NURBS curve desired. A knot vector is a list of parameterization values, or knots, that specify the spacing of the parameter values for the B-spline curve (Sederberg 2007). A *uniform B-spline* is a curve whose knot vector consists of evenly spaced parameter values. A non-uniform B-spline is a curve whose knot vector is not evenly spaced thus allowing a larger variety of curves to be represented. A Non-Uniform Rational B-spline (NURBS) is a curve that has weights associated with each control point that determine the amount of influence the control point has on the overall spline shape. For a NURBS curve, the *p*th-degree B-spline basis functions defined for a non-uniform knot vector are defined by the following:

$$N_{i,0}(t) = 1 \text{ if } t_i \leq t < t_{i+1}, 0 \text{ otherwise}$$

$$N_{i,p}(t) = \frac{(t - t_i) N_{i,p-1}(t)}{t_{i+p} - t_i} + \frac{(t_{i+p+1} - t) N_{i+1,p-1}(t)}{t_{i+p+1} - t_{i+1}} \tag{5}$$

The knot vector for a NURBS curve is shown below

$$U = \left\{ \underbrace{t_0,\ldots t_0}_{p+1}, t_1,\ldots t_{i-1}, \underbrace{t_i,\ldots,t_i}_{p+1} \right\} \tag{6}$$

where there are $p+1$ extra knots prepended and appended to the knot vector that control the end conditions of the B-spline.

To construct a B-spline or NURBS surface, for example, a bidirectional net of control polygon points, two knot vectors and the products of the B-spline basis functions are needed (Sederberg 2007). A NURBS surface of degree p in the u direction and degree q in the v direction is a piecewise rational function of the form

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} N_{i,p}(u) N_{j,q}(v) P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} N_{i,p}(u) N_{j,q}(v)}, \tag{7}$$

with

$P_{i,j}$	control points,
$N_{i,p}, N_{j,q}$	basis functions,
$w_{i,j}$	weights,
u, v	parameters with 0≤u,v≤1, and
n, m	number of control points – 1.

The control points P_{ij} form the bidirectional control net where the surface is defined from (n+1)*(m+1) control points. The surface also has two associate knot vectors of the size n+p+2 knots and m+q+2 knots when its degrees are p in the u-direction and q in the v-direction. The B-spline basis functions along with the knots vectors are of the same form as shown above in equations 5 and 6.

Using NURBS in models that are parameter rich and free-form in nature allow for surface definitions to be smooth, continuous, and easy to manipulate. NURBS allow proper placement of surface fitting data, sufficient parametric curve continuity, and allow proper geometry continuity to ensure correct curvature of the specified curve. These techniques are the basis for all the curves created in this thesis.

2.5 Multidisciplinary Optimization

Multidisciplinary optimization (MDO) is the process of connecting multiple CAD/CAE/CAM (CAx) tools into a seamless program that integrates with an optimization engine. The optimization strives to achieve the objectives of the design by iterating quickly and autonomously through various designs until an optimal design or set of designs is achieved. To meet the objectives in a design problem, constraints need to be created and design variables defined (Parkinson 2006). In CAD-centric shape optimization, design variables become the driving parameters of the parametric model. These are used in the optimization problem to allow the optimization process to control the geometry and shape definition (Hardee et al. 1999).

For MDO to be functional, the creation of a master program is required. This involves the file transfer (I/O) and linking of the chosen engineering software to create a dynamic process flow for all information in the loop. The method for implementing MDO into a design process involves several tasks, which are completed by various disciplines involved with the optimization routine. Hogge (2002) outlined a typical parametric scheme of events in a MDO situation (see Figure 6) as follows: parametric modeling, analysis, optimization, and design selection.

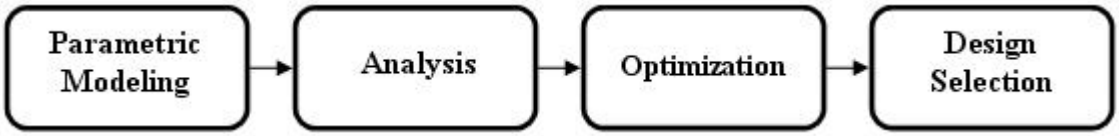


Figure 6: A Parametric Design Scheme

To make the master program run smoothly, API programs specific for each discipline need to be developed. Once the disciplines are integrated for MDO, the next step is to automate the handoff of information from one program to the next so that the sequence is allowed to run in a batch mode of operation. There are several methods for linking analysis codes to optimization software. One method is to use input and output files to link the optimizer and analysis code as illustrated in Figure 7.

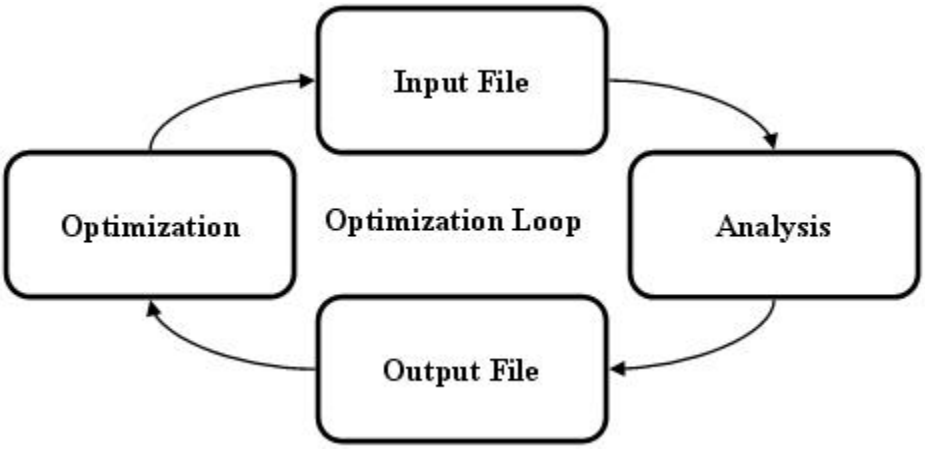


Figure 7: A General Design Optimization Loop

The optimization program changes the design variables and creates a new input file. The analysis code is then executed using the new input file and the results of the analysis are written to an output file. The optimization engine reads the outputted results file and the loop (design

iterations) continue until an optimal set of parameters is determined. This should be done with software that can handle process integration, design exploration, analysis and visualization. For this thesis, this will be handled with iSIGHT-FD, a suite of visual and flexible tools used to set up an automated plan to thoroughly explore the design space and find optimal solutions.

Within the last decade, CAD-centric shape optimization has been successfully implemented using commercially available CAD software (Hardee et al. 1999, He et al. 1998, Hogge 2002, Ou et al. 2002 and 2003, Delap 2003, Wu et al. 2004). Although shape optimization has been successful, there are some limitations and needs stated by researchers. A fundamental need that is becoming less of a concern as API's improve and software becomes more integrated is the inability to exchange data between CAD, analysis, and optimization tools seamlessly (Hogge 2002, Delap 2003, Elliott 2004, King 2004). Vanderplaats (1999) stated that except for robust, well-defined structures, full automation of CAD, analysis, and optimization processes is not possible and requires multiple intermediate preparatory steps to be employed. Elliott (2004) worked with parameter rich surface models (PRSM's) that had a high number of driving parameters and recommended that with the improvements in future technology that PRSM CAD-centric optimization methodologies could be incorporated into MDO loops to handle high fidelity complex models with faster convergence times.

Some of the limitations stated and described in the past are overcome by this research. First, the exchange of data between software tools can be automated even when dealing with less familiar simulation software that has a limited API. Second, recent developments in the API's of today's third generation CAD software make it possible to generate models robust enough for automation and integration with optimization. These models allow for repetitive procedures that took weeks to successfully execute to now execute programmatically in minutes. Lastly,

allowing PRSM's with the thousands of driving parameters to be integrated with optimization capabilities to allow for faster convergence than has been seen in the past by focusing on the critical driving parameters.

2.6 Statistical Response Surface Modeling

Response surface modeling methods originally were developed to analyze experimental data and to create empirical models of the observed response values (Gunita 1997). Response Surface Methodology (RSM) is an experimental technique invented to find the optimal response within specified ranges of the factors. These designs are capable of fitting a second-order prediction equation for the response (JMP 2005). The particular forte of RSM is its applicability to investigations where there are few observations because the experiment is both very expensive and very time consuming to perform (Gunita 1997).

The term "Response Surface Methodology" refers to a complete package of statistical design and analysis tools which are generally used for the following steps (Lawson et al. 2001):

1) Design and collection of experimental data (DOE)
2) Regression analysis to select the best equation for description of the data
3) Examination of the fitted surface contour plot to understand design space

2.6.1 Design of Experiments (Central Composite Designs)

To be able to collect data about the response of some model or simulation a structured approach must be taken to ensure "sensible" data can be obtained. Trial and error methods are time consuming and will not describe the design to allow prediction of a response based off of

various input combinations. Factorial designs consist of all combinations of a factor level or two or more factors (see Figure 8).

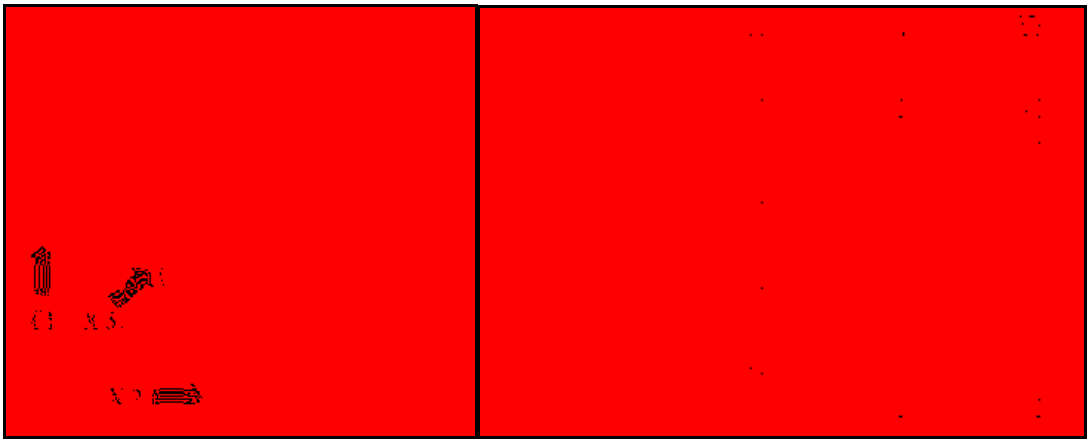


Figure 8: Two Level Three Factor Full Factorial DOE Example

The goal is to understand the impacts the factors (variables) have on the system of interest. Frequently this involved the use of replication (to understand variability), randomization of experiments (ensures no erroneous signals), and addition of center points. Using a full factorial with replicates and center points can be taxing (time and resource consuming) for a 3 factor experiment (~20+ experiments). Frugality becomes of primary importance since it can oftentimes be time consuming and expensive to obtain all the results needed for a full factorial design (Larsen et al 2001). To reduce the number of needed experiments, another experimental design method known as central composite design (CCD) may be used. CCDs build upon two level factorial designs. CCDs are typically used for unconstrained optimization but can be adapted for constrained optimization by modifying the CCD for discrete limits on the factors (design variables). The model for 2^k factorial design is:

$$\hat{Y} = b_0 + \sum_{i=1}^{k} b_i X_i + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} b_{ij} X_i X_j \qquad (8)$$

With the interactions terms of higher order usually neglected ($\sum_{i=1}^{k} b_{ii} X_i^2$). Equation 8 allows for the estimation of the coefficients (b_0, b_i, and b_{ij}) for the main effects and 2 factor interactions (Simpson 1998). The only missing terms from the full quadratic equation are the squared terms in each X_i. To estimate these terms a set of axial (star points) and center points are used. These extra points are essentially a set of one-at-a-time experiments. Using a central composite design permits the development of a reasonably accurate data representing the design response. Figure 9 shows an example of a 3 factor central composite design.

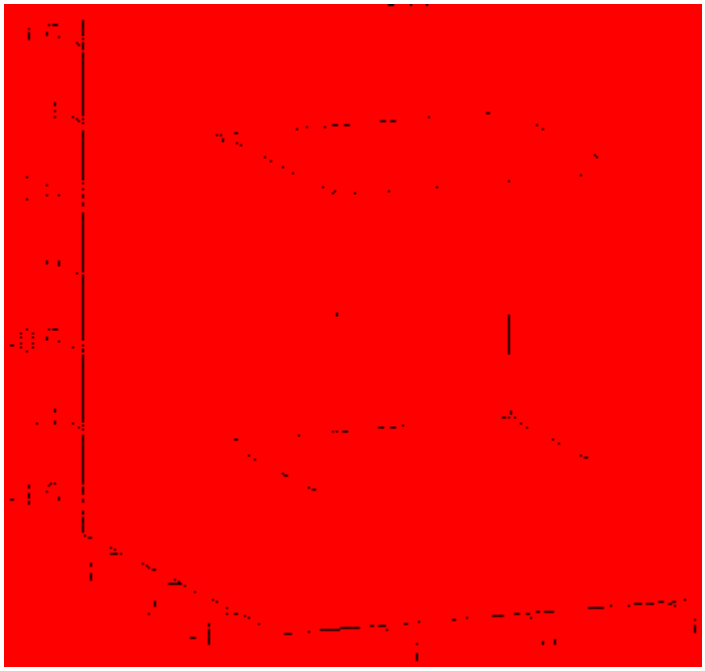


Figure 9: Example 3 Factor Central Composite Design

2.6.2 Regression Analysis

The goal of response surface methodology is to detail the relationship between the response and the independent variables of interest to seek an optimum set of system conditions (Biles 1984). To do so a polynomial approximation equation (quadratic equation) relating the factors to response is created. Using non-linear regression techniques a quadratic equation can be generated that shows which factors (variables) are significant and what the corresponding coefficients of the equation are. This is accomplished by minimizing iteratively the sum of the squares. The model for a general quadratic equation for k independent variables is:

$$\hat{Y} = b_0 + \sum_{i=1}^{k} b_i X_i + \sum_{i=1}^{k} b_{ii} X_i^2 + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} b_{ij} X_i X_j \qquad (9)$$

Using computer aided regression analysis tools, the significant factors and their interactions can be identified and their coefficients obtained forming the quadratic equation that represents the design space of interest. A more detailed description of RSM techniques and tools can be found in (Box and Draper, 1987).

2.6.3 RSM Surface Creation/Examination

Given a quadratic model, certain software can plot surfaces or contour plots over the factor ranges. When only 2 or 3 important variables exist, 3D surface plots can be created either manually or automatically in statistical analysis software. This provides an approximation of the true response surface over the region of interest, allows the optimum operating conditions to be chosen, and permits improved understanding of the estimated response the model provides for various design parameter combinations.

CHAPTER 3: METHOD

The method presented in this chapter involves the file transfer (I/O) and linking of chosen engineering software to create a dynamic process flow of all information to create an optimization loop. A Design of Experiments (DOE) is used to create a Response Surface Model (RSM) using results from the optimization scheme that will predict what the optimal design variables should be to produce the optimal results. This method achieves the objectives outlined in section 1.1. To accomplish the objectives, a commercially available iterative workflow environment handles the integration of the general steps shown in Figure 10

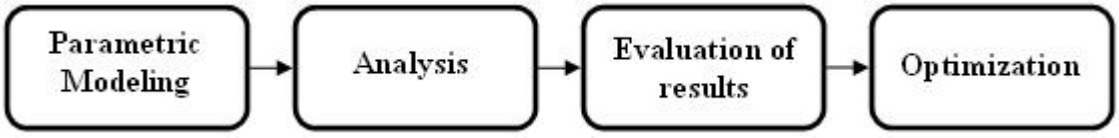


Figure 10: A Tool Design Optimization Scheme

This method enables significant time savings by integrating the outlined CAE operations into an autonomous CAD-centric tooling design optimization routine. For a description of the advantages behind CAD-centric models in optimization, refer back to Section 2.1.5. The interactive and programmatic implementation of this method is presented in Chapter 4.

3.1 Parametric Modeling

Complex free-form surfaces are generally constructed with a large amount of engineering knowledge (Elliott 2004). Due to the large amount of required knowledge and information their construction process is long and tedious. Surface modeling or remodeling time is encountered every time an update or change is required. To ease the additional work load due to possible design changes, CAD parametrics are used. Parametrics allow the control of complex geometries through the introduction of variables to the underlying mathematical equations. Parametrics avoid "hard coding" of parameter values for the underlying geometries. Changing the controlling variables, allows the geometry changes to be made without requiring an engineer to start over.

When large and complicated models such as free-form die and work piece geometries are created, updates are required to obtain the proper shapes. Updating a model can be undesirable due to the required time needed to recalculate every object and feature affected by the change. As model complexity increases the time required to make changes and updates increases. Likewise model complexity increased the probability of update errors and inconsistencies within the parametric model. To overcome the model update time and update error possibilities, a generative approach to CAD parametrics is used. The generative approach to modeling is the conduit for all of the geometric model construction in this research.

3.1.1 Generative Parametrics

A generative parametric model is one that is recreated automatically every time a modification is made to the parameterization scheme of the part. For large complex free-form surfaces this is desirable and thus eliminates the possibilities of encountering update errors,

which render the part geometries corrupt. This method requires a considerable amount of preparation and effort during the model construction phase but allows for significant time savings during execution of the design iterations.

To accomplish the level of automation necessary for complex free-form surfaces, object-oriented programming (OOP) coupled with application programming interfaces (API's) are used for generating the initial and modified models for the design cycle. The API's allow for automated model construction eliminating the man hours used to interactively construct the model surfaces.

Elliott (2004) determined that programmatic surface modeling for the PRSM object class requires study of three steps in order to develop successful automated parameterization schemes which are: planning, development, and evaluation. The research Elliott has performed is the backbone of the modeling methodology described in this research. This research builds squarely on top of the notion of reusable parameter rich surface models for the design and construction of forming die and workpiece geometry. This research implements applicable code snippets from Elliott's research for the development of this optimization methodology.

3.1.2 PSRM Planning

In order to successfully apply feature-based parametrics to complex free-form concepts, thorough planning is needed to properly scope the amount of work needed during concept development. The planning stage can be described by three different stages, similar to developing solid modeling schemes.

The first stage is identifying the necessary inputs to generate parametric models. This entails determining what format the input should assume (i.e. imported external part files, bulk point data sets, or individual parameter name/value input).

The second stage is designing a strategy for modeling the specific product. This step aids in creating a seamless automated design process. Stage two involves not only the CAD model geometry creation sequence but also the imposed rules required by various downstream disciplines. An equally important aspect of the design strategy involves determining what feature primitives and surface elements are needed at which location in the model.

The third and final stage of planning is parameterization of the surface modeling features. This entails determining the complete set of design parameters and which parameters can remain unchanged or subsequently derived from the master design parameters. During this stage it is important to determine the model hierarchy along with the appropriate constraint types such as tangency, continuity, parallelism, object size, shape, and location to name a few.

3.1.3 PRSM Development

Once the features, parameters, and relationships have been identified for the PRSM objects, the model can be developed. Developing the PRSMs can be accomplished by either an automated (programmatic) or manual (interactive) approach. An automatic approach to modeling is taken for the required complex free-form surfaces. It is recommended that an automatic approach be taken if many instances of a similar concept need to be analyzed. Developing a complex automated parameterization tool enables fast and efficient geometry creation within the PRSM domain. Developing an automated application for a complex class of parts allows the difficult and time consuming model characteristics to be transformation into non-cumbersome attributes. The attributes simply describe the design requirements, rather than presenting obstacles. By creating an automated model generation approach, the model becomes a reusable product model requiring little to no manual engineer intervention.

The modeling methodology developed for this thesis consists of algorithms written using a high-level programming language and third-generation CAD system APIs. As presented in Section 2.2, CAD API's allow a designer to programmatically model parts, query product models, and create assemblies and drawings by allowing access to all of the core interactive functionality in a programmatic form. User Functions (UF) or executables are the product of the implementation of CAD API's. Most third-generation CAD APIs, such as CAA RADE (CATIA) and UG/OPEN (NX), provide a C/C++ language interface that supports ANSI C standards. Using the object oriented data structures offered by C/C++ within the automated model parameterization tool the input data can be stored, implemented, and manipulated for this research.

3.1.4 PRSM Evaluation

Elliott (2004) outlined important criterion for deciding the effectiveness of product modeling schemes for complex free-form surfaces. The requirements set forth for the evaluation of these types of surfaces are the following necessary capabilities:

- Allows for large amounts of input data
- Creates reusable models by automatic parameterization of independent model features
- Generates high-fidelity geometry
- Builds and improves upon current design process

The models created by the methods developed not only need to function properly for the above capabilities but also for model variations of the product family when desired.

3.2 Analysis

With successful completion of a generative parametric model for the desired product, the model needs to be prepared and submitted to the downstream analysis applications. In many cases, several CAE software programs are needed to perform the desired analyses. Each analysis process requires automated execution of the disciplines used. In order to link all the analysis codes together, each of the programs (modules), are developed so as to allow easy connection to the master program.

When developing automated techniques for handling the analysis portion of the master program there are typically three stages of development namely: 1) manual interactive exploration of the desired function (Section 3.2.1), 2) development of API program modules that automatically analyze each discipline (Section 3.2.2), and 3) connecting the modules so as to achieve fluent communication between programs (Section 3.2.3). There are two separate analysis codes linked together for this research: ANSYS mesh generation and DEFORM forming simulation.

3.2.1 Interactive Development

When developing a new procedure for the automation of a certain process, it is recommended to carry out the procedure interactively with the graphical user interface. This enables the developer to determine all the necessary steps for complete automation and to know what portions of the development process exist as part of the software's core functionality and which portions require customized development. By establishing guidelines for creating the modules, all the required inputs and outputs needed for preprocessing through postprocessing

can be identified. This allows the modules to be developed in parallel with discipline experts and allows for ease of integration into the master program.

3.2.2 API Program Development

Separate API based automated programs are needed for each application desired for the analysis software programs of choice. The API programs can be developed using either a single API toolkit or a combination of many toolkits as described in Section 2.2. Figure 11 demonstrates a possible scenario for a program that requires analysis in three separate disciplines.

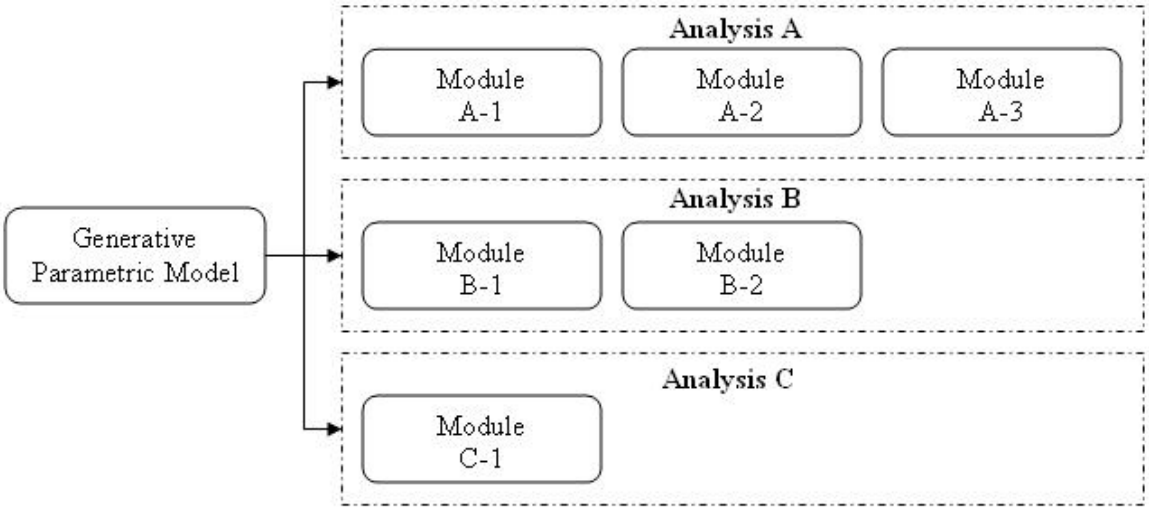


Figure 11: API Modules

Each discipline has the possibility of requiring development of multiple API programs as needed to fulfill the requirements of the application. The modules could be developed to run in parallel or in series based on software availability and the computational availability. This research runs the analysis modules sequentially due to licensing constraints in the software chosen for integration.

3.2.3 CAD/CAE Integration

As described in the section heading, to achieve automation of the proposed design cycle, a thorough integration of the necessary components must be developed. Modeling and analysis codes must be packaged such that integration is as simple as calling an executable with the proper inputs readily available. The framework for the project, iSIGHT-FD, will integrate NX CAD, ANSYS mesh generation capabilities, DEFORM forging simulation, and numerical surface construction for evaluation to form a complete loop. When direct connections do not exist between the engineering software of choice, custom scripts and links are developed to achieve seamless software integration. This includes the creation of the properly formatted input and output files along with all the necessary support files for proper execution.

A common step for overcoming the discontinuities between parameterized CAD models and downstream analysis (when analysis isn't performed in the CAD software) is to identify commercially available direct connections between CAE systems. If the direct connection does not exist, a standard neutral file format such as IGES or STEP can be used for translation.

Frequently program specific input format must be used to execute analysis code. Special formats must be generated as an output or translation prior to executing downstream analysis. In general, these preparatory actions can be executed in sub-routines developed in an API environment to properly format the data for the next program. Generation of unique input should be capable of batch execution once the parametric CAD models have been constructed.

CAE analysis typically occurs in three steps: preprocessing, numerical simulation, and postprocessing as described in Section 2.3. Preprocessing involves the preparation of a model for submittal to the solver. Examples of preprocessing could include import or creation of the geometry entities, generation of a FE mesh, applying boundary conditions (loads and

constraints), and application of any other system default settings. The numerical simulation step involves submitting the input deck, of proper format, to the desired numerical solver and allowing the results to be calculated and saved. Postprocessing is the process of extracting and viewing the results desired from the simulation either from a GUI or outputted results file.

Depending of the optimization objectives of the system being developed, the analysis can be constructed many various ways. One popular method CAE software providers are promoting is a complete package (whole analysis process occurs inside of the same software program). Oftentimes, all the needed programs cannot produce the proper fidelity results or multiple programs are required for analysis and must be shared among the better (or chosen) of the CAE tools available industry wide. That being the case for this research, automated methods for allowing the preprocessor, numerical simulator, and postprocessor need to be developed to ensure seamless communication between different programs (see Section 2.2). Given the high fidelity of the complex free-form surfaces necessary, a software program with full access to its core programmatic functionality would be needed to generate the desired structured mesh. This is accomplished inside of ANSYS by way of the ANSYS Parametric Design Language (APDL) API. The output from ANSYS contains the proper DEFORM input format. DEFORM, a forming/forging simulation solver, acts as a secondary preprocessor applying system specific default settings. DEFORM, once the simulation has been executed, performs initial postprocessing to extract the displaced nodal locations and associated stresses.

When communication between the various programs is not available, simple scripts, batch system routines, and simple C++ algorithms allow the modules to perform the required tasks. This enables the programs to communicate in series (or parallel) by passing the necessary information from one program to the next in a simple and smooth manor.

3.3 Numerical/Geometric Surface Interpolation

The simulation results are used in conjunction with numerical surface construction techniques to develop a mathematical model of the NX 3D surfaces. The displaced forming die surface data are compared to the original design intent (nominal surface profile). Surface fitting techniques are used to analyze 3D surfaces from the simulation results and the desired nominal surface contour (see Section 2.4). The nominal surface profile data is supplied to the program as an input. The program to construct the optimal surfaces can be developed using an OOP language in a strictly analytical manor or inside of a custom API program that can graphically display the results if desired. This research uses the latter (UG/OPEN API toolkit) to develop the means for interpolating the deviations between the simulated results and nominal surfaces using surface deviation interpolation. The nominal surface profile data is passed to the surface construction program as a bulk point data set that represents the final formed part and desired die surface geometry for the operation being simulated.

Two methods are employed to give a fitness value to the deformed surfaces. A fitness value is a method for determining the amount of validity the surfaces possess. This fitness value is used in the optimization to judge the closeness of the simulated vs. desired nominal surface representation. Method 1 uses the surface normals and develops an estimate for the surface-to-surface deviations. Method 2 uses a linear distance between the two surfaces in the direction of die movement. Both methods require the calculation of the intersection point between a direction vector and a complex free-form surface. The deviation values are compared with each other to ensure that the deviations are within family for the specified point. Method 2 is passed to the optimization engine for evaluation based on the methods described above.

3.4 Multidisciplinary Optimization

It has been stated by Hogge (2002) that there are two main challenges to performing MDO: computational expense and organizational complexity. Computational expense (time needed to run analysis) increases as the various disciplines required use more analysis code (software) and design variables. The organizational complexity increases when multiple analysis packages and various design approaches are brought together to form one method or design system capable of multiple cycles or iterations. The iSIGHT-FD framework is used to create a workflow of the model and analysis modules. The optimization is also handled by iSIGHT-FD. The chosen framework typically handles the integration of the optimization technique by way of input, optimization iteration and then output results. Using an optimization technique external to the integration framework, such as Microsoft Excel solver or OptdesX is possible but will not be implemented in this research.

By using iSIGHT-FD and its optimization capabilities, an iterative work flow environment is created. In order to perform the optimization, objectives and constraints must be determined. Constraints and ranges on CAD parameter values (the optimization scheme is CAD-centric, see Section 2.1.5) are constrained within the iSIGHT-FD framework.

There exist multiple algorithms for optimization. The algorithms used for this research must be able to explore continuous non-linear design spaces, be well suited for long running simulations, and exploit the local area around initial design point in order to locate the optimum. For these requirements, a gradient based or a sequential quadratic programming based algorithm could be used as the optimization technique. The choice for the optimization algorithm and reasoning for that selection is presented in section 4.4.2.

3.5 Statistical Response Surface Methodology

Response Surface Modeling (RSM) methods are used to characterize in detail the relationship between the factors (design variables) and the response (optimization results). The goal is to be able to predict the response to specific inputs. Response surface methodology refers to the use of three steps of statistical design and analysis, namely: Design (which includes the collection of experimental data which allow fitting a general quadratic equation for smoothing and prediction); Regression analysis to select the best quadratic equation for description of the data; and examination of the fitted surface via contour plots and other graphical/numerical means.

There are many types of RSM designs that can be employed depending on cost of model execution. Design methods such as full factorial, central composite, box behenken, and small composite designs are commonly used along with their variants.

Given that the methodology used in this thesis predicts long simulation run times, exploring all possible combinations of a 3^k factorial design (k being the number of design variables) would be very costly. Central Composite Designs (CCDs) are more frugal with experiments than full factorial designs and are commonly used in conjunction with computer simulations.

Using graphical and numerical methods internal to the RSM software, true optima for a particular design can be identified. Section 4.5 details the development of the specific modeling methods used to help identify the optimum design variable selection.

CHAPTER 4: DEVELOPMENT

This chapter discusses how the method outlined in Chapter 3 is applied using the selected CAD and CAE software. The test case for the method is applied to a 1st stage compression system's shroudless hollow fan blade for a turbine jet engine shown in Figure 12. The airfoil chosen for this study exemplifies all the required characteristics for parameter rich complex free-form surface models in that it requires free-form surface construction, large amounts of data as inputs, surfaces are representative of difficult to parameterize non-reusable features, requires high-fidelity mesh generation for optimal model description, and is a large portion of the complex and lengthy multidisciplinary optimization process.

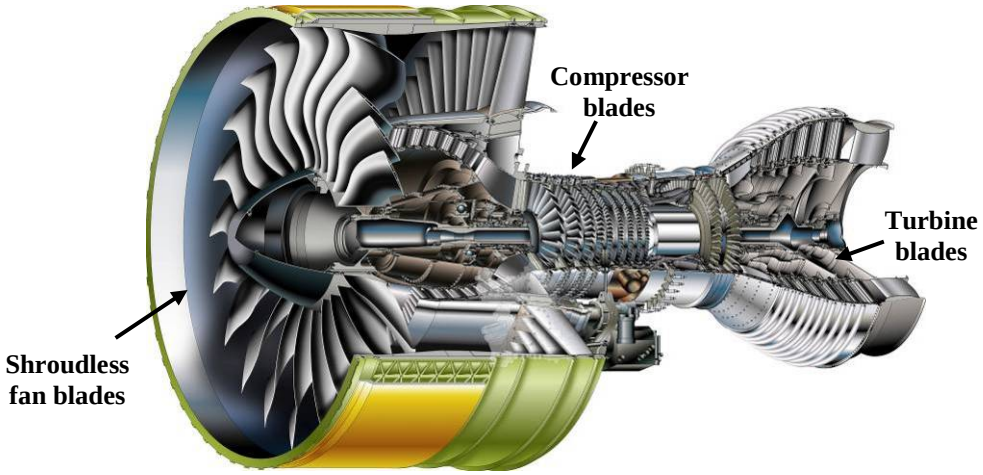


Figure 12: A Cut-away of the GP7000 Jet Engine Highlighting the Fan Blade

The development of the methodology is accomplished with the use of the NX UG/OPEN API, ANSYS ADPL API, DEFORM's macro scripts, iSIGHT-FD for integration, and JMP for Response Surface Modeling. The sections in this chapter are parallel to the sections in Chapter 3. The hardware and software used is listed in Table 1.

Table 1: Hardware and Software Used

CPU	HP xw4300 w/Intel Pentium 4, 3.4 GHz CPU & 3.25 GB RAM
Operating System	Microsoft Windows XP Professional
Compiler	Microsoft Visual Studio .NET C++
CAD Software	Siemens NX 4.0
Mesh Software	ANSYS 10.0
Analysis Software	DEFORM-2D v9.01
Integration Software	iSIGHT-FD 1.0
Statistical Analysis Software	JMP 8

4.1 Parametric Modeling

Since the parametric model functions as the catalyst for changes implemented by the optimization process, it must accurately represent the geometry necessary for all of the downstream analyses. The method for generating parameter rich complex surface models, as outlined in Chapter 3, must be followed to successfully plan, develop, and evaluate the generative modeling approach.

4.1.1 Planning

Planning for an automated generative modeling scheme for parameter rich complex surface models is extremely important. The genius of a most major projects stems from planning work done in advance. The planning for this implementation is presented in the following sections and includes:

1. Identifying and characterizing the necessary inputs for the modeling methodology.
2. Developing a modeling strategy that defines the parameterization and construction of the automated CAD models

4.1.1.1 Inputs

The starting point for all the shroudless Hollow Fan Blade (HFB) surfaces developed for use in this thesis require an airfoil definition in a bulk point aero source (AS) file, a NX root attachment part file, and a predefined bounding box defining the region for the hollow cavities of the airfoil in the form of a NX part file.

Aero Source (AS) File

An AS file is a text file generated by aerodynamicists that have determined the optimal shape an airfoil must have to obtain the desired performance characteristics for a jet engine. The text file contains Cartesian coordinates that describe the flow path for the airfoil shape (see Figure 13).

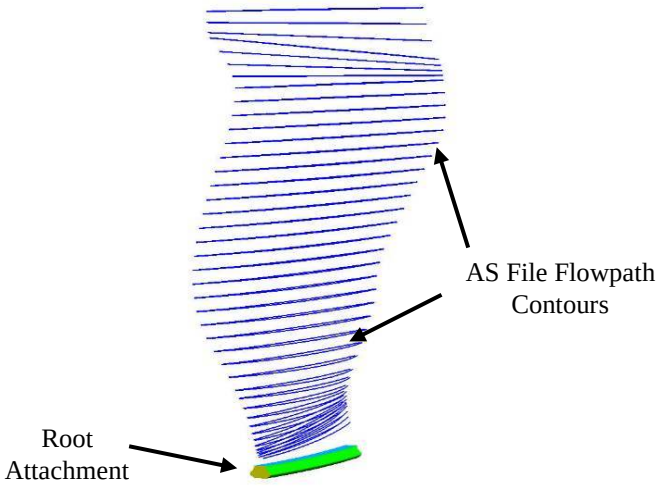


Figure 13: AS File with Root Attachment Properly Positioned

The coordinates are arranged such that there are a certain number of cross-sections and certain number of points per section defining the airfoil region. The format of the aero source files used in this research is similar to those typically found in industry.

The AS files used in this research are non-proprietary in nature but do provide a realistic starting point for the complex model generation. Although the AS files used are not those used in industry, the b-spline curves and surfaces developed under the applied methods are of the same mathematical form as those typically found throughout industry. AS files used in this research contain the same level of fidelity as industry airfoil surface models. Each AS file represents a unique airfoil design. The AS files include the Cartesian coordinates for the flow path and also contain other important parameters such as the number of cross sections and number of points per section.

In order to import the data, the AS files were scanned by the automated surface modeling program and the driving parameters and data coordinate points were stored as 3-dimensional floating point variables within the program database.

Root Attachment

The second input to the HFB automated surface modeling program is a NX part file containing the root attachment of the workpiece. This part is the portion of a completed airfoil that attaches the fan blade to the engine rotor or "hub". In industry, root attachment geometry does not vary significantly from engine-to-engine. Figure 13 shows a typical root part used for HFBs. This part file is solid geometry and has associative parameter names or labels associated with features and faces necessary for downstream parameterization of curves and surfaces. For the HFB automated surface modeling program, the solid root attachment has all of its surfaces

flagged with identifiers for later object recognition and positioning of die and workpiece surface constructs.

Hollow Window Bounding Box

For the purpose of developing a HFB automated surface modeling tool, a perimeter for the hollow cavities of the HFB must be defined. This is accomplished by using a NX part file that contains a single joined curve that describes the perimeter of the hollow cavities (see Figure 14).

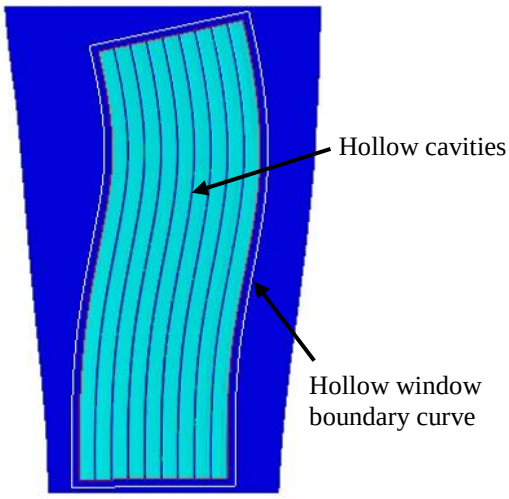


Figure 14: Hollow Window Boundary

The solitary closed curve defines the cavity region (whether in a flat or twisted state) where compensation is needed. This region ensures that cavity collapse does not occur during forming simulation.

4.1.1.2 Strategy

Developing a strategy for a program that automatically creates complex free-form surfaces is much different than interactively (using a graphical user interface) creating the same

models. When a programmatic approach is taken for such complex models, it must be capable of producing an exact model to that created interactively and do so repetitively. The design process must likewise be able to preserve design intent and process quality.

The goal of this research is to generate the forming die surfaces for a particular HFB that will produce the design intent once the forming simulation has been executed. The blade workpiece will also be created from the same starting point as the die surfaces. To generate the die and workpiece surfaces, the original design intent must be reverse engineered. To reverse engineer the design intent to produce die surface and workpiece geometry a specific strategy has been developed. This strategy allows all geometry pieces to use either all or a portion of the programmatic method developed for successful implementation. To automate the parameterization and generation of the surfaces, the design must be accomplished by way of four general steps, otherwise called modules as shown in Figure 15 to Figure 18.

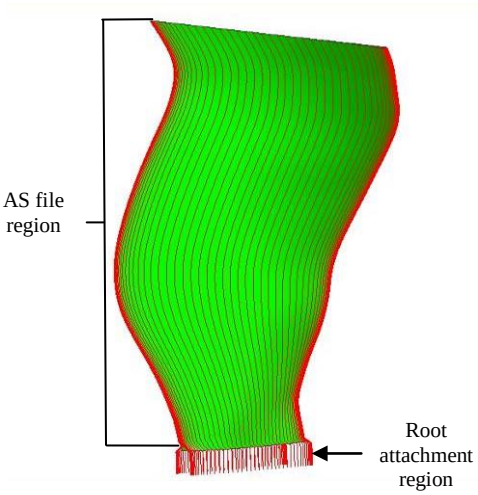


Figure 15: Module 1 Modeling

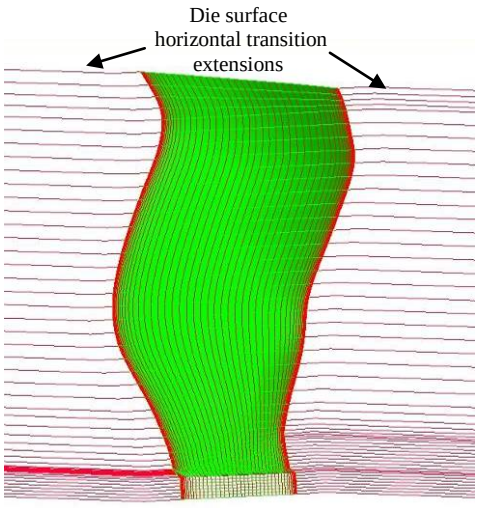


Figure 16: Module 2 Modeling

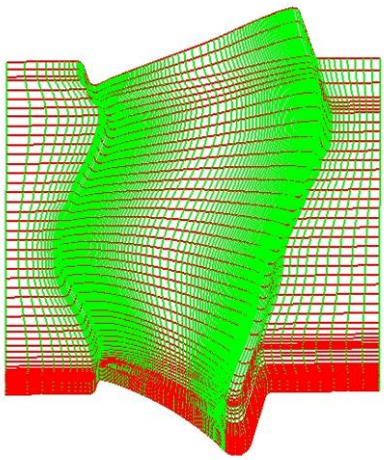


Figure 17: Module 3 Modeling

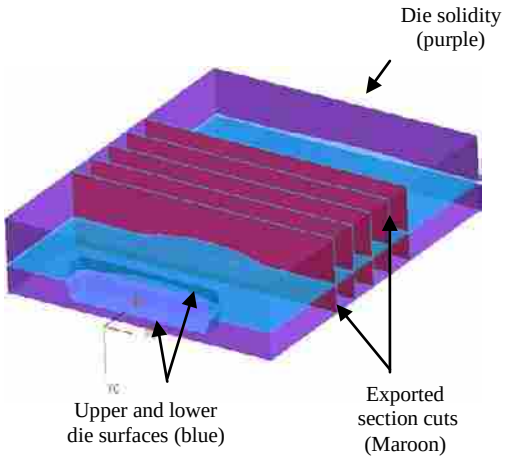


Figure 18: Module 4 Modeling

Module 1

Module 1 begins with the AS file and root attachment inputs. The two inputs are integrated together with vertical B-spline curves that pass through the AS file data points and intersect with a pre-final form root attachment. This creates vertical curves that follow the contour of the airfoil from the base of the root attachment to airfoil tip and successfully integrates the two inputs as shown in Figure 15.

Module 2

Module 2 has two capabilities to handle two completely different surface outputs. If generating the airfoil workpiece, module two is responsible for developing the leading and trailing edge curve transition to form a closed workpiece. If development of a forming die is required it handles the transition from the airfoil to the die plane for the forming surface scenario. This transition takes into account the die draft angles, die machining requirements,

forming flashing collection areas, and shape forming parameters to encourage proper die fill. This operation is performed for both the upper and lower die surfaces.

Module 3

Module 3 is used to generate the net of B-spline control curves through which a B-spline surface is created (see Figure 17). The control net of curves is developed by properly spacing curves that extend from the root attachment to tip of the airfoil along the span of the die surface area. Cross-section curves are created from the leading to trailing edges of the airfoil for the width of the airfoil surface. Surfaces from the control net of curves for both the die halves of the airfoil.

Module 4

Module 4 creates solid geometry from the Module 3 resultant surfaces and prepares the model for downstream analysis by applying defeaturing (where needed), model simplification, and any simulation assumptions that must originate from the CAD model. This entails taking the cross-sectional slices at the proper stations of the solid die and workpiece geometries. This provides the 2D slices of the simulation domain used by downstream programs for analysis. The NX parts representing the three objects are then saved individually to the proper location for later retrieval (see Figure 18). The three objects representing the die and workpiece geometry are later reunited for simulation purposes.

4.1.2 Development

This section explains the details involved in developing the four modules that parameterize the HFB die and workpiece geometric features and creates the generative design program executable. The automated design application has been developed using the high level

programming language C/C++ coupled with the native toolkit functions from the UG/Open API library of NX. The modeling method is a seamless NX application that automates the generation of HFB die/workpiece models and applies parameters and constraints to the detailed surface features. The automated design tool has been developed to run external to the interactive NX environment.

Separate modeling applications have been created for generating the die surfaces and workpiece geometry. The strategy described in section 4.1.1.2 was implemented for each application.

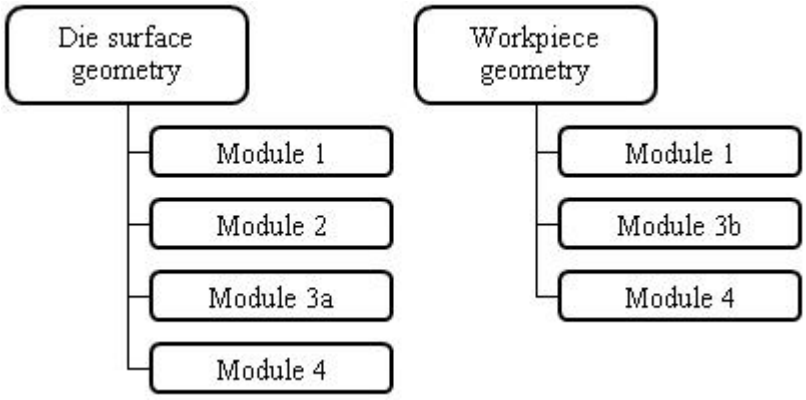


Figure 19: Generalized Application Organization

Figure 19 shows the organizational sequence of generalized algorithms used for each application developed. The following sub-sections describe the construction of each module and any pertinent algorithms. Due to partial proprietary restrictions, the modeling algorithms are not discussed in explicit detail rather key points and explanations along with visual examples of algorithm functionality are presented.

Module 1 Development

The development of Module 1 is broken into 4 parts: rootblock reverse engineering development, AS data envelope preparation, airfoil die/workpiece rootblock integration, and airfoil die/workpiece departure transition creation.

Module 1 Part 1

Part 1 imports the root attachment geometry (Section 4.1.1.1) and parametrically reverse engineers the attachment to an earlier form of developmental geometry. The reverse engineered object created around the root attachment is called a rootblock. The rootblock is shown in two different configurations in Figure 20 by the semi-transparent shell resembling a house. The rootblock encapsulates the root attachment by use of the parameterization shown in Figure 21. The rootblock represents the root geometry needed for integration with the AS file data.

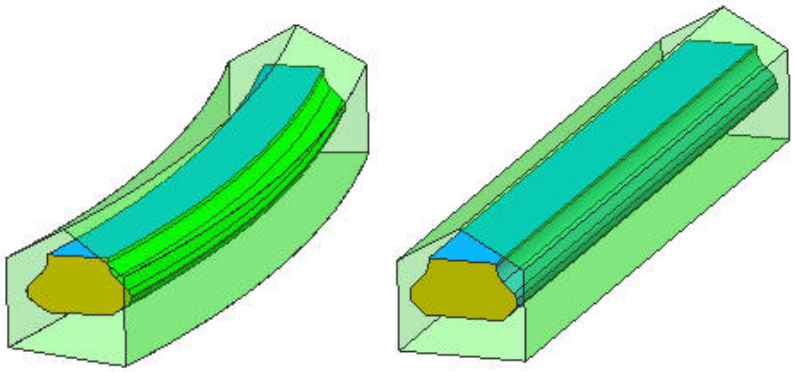


Figure 20: Rootblock (Transparent Green) for Twisted and Straight Root Attachments

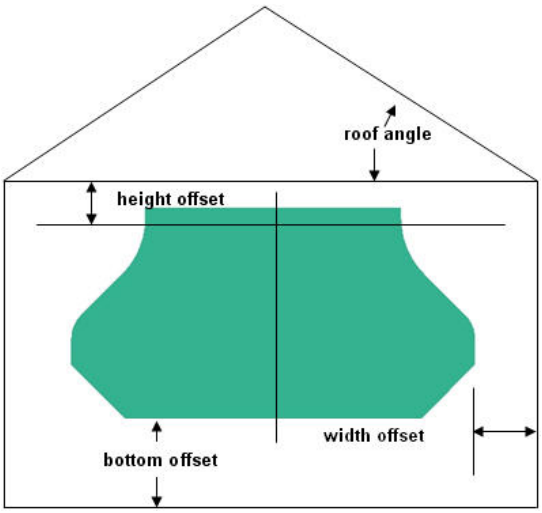


Figure 21: Rootblock Parameterization

Module 1 Part 2

Part 2 of Module 1 applies the appropriate manufacturing/machining extra amount of material (also referred to as the machining envelope) to the AS data and prepares the airfoil leading and trailing edges for proper manufacturing blade clearance. The die/blade clearance is required to ensure the forming dies provide sufficient compressive force on the workpiece to facilitate the correct cavity back pressure. This cavity back pressure ensures proper die cavity fill and forces the workpiece geometry to form to the correct die shape specifications.

It is important to note that the process being simulated is a bulk forming simulation (macro shape) not a net forming simulation (micro shape). Therefore, the simulation results will have the proper airfoil shape and curvature but will contain unfinished features such as the leading and trailing edges which are formed in a subsequent manufacturing process not covered in this thesis.

To apply the machining envelopes (additional stock material) necessary, the AS file Cartesian point data is manipulated prior to creating B-splines (cross-sections) through the AS file data points. From these curves the airfoil machining/forming surfaces can be generated.

Prior to initial cross-section curve creation the AS file data points X, Y, and Z coordinates must be adjusted to represent the pre-formed geometry needed as the simulation starting point. This is accomplished by multiple mathematical based data manipulation operations. The three main manipulation operations are listed here: uniform crush, deformation compensation, and deflection compensation. These manipulation operations have been chosen as design variables for the optimization scheme and are described in the following paragraphs.

Uniform Crush

Given that the airfoil could contain an internal scheme of hollow cavities, there is a potential during forging that the hollow cavities could collapse resulting in an unusable part. The cavity collapse would be partly due to the structural membrane covering the cavities being insufficiently thick. To compensate for a thin surface over the hollow region, a uniform extra thickness is added to the geometry to strengthen the complex part. To compensate for this issue in this research, the region is defined as a "hollow window" as shown in Figure 14 described in Section 4.1.1.1. To generate the offset, every data point that falls inside of the hollow window region has a normalized outward surface vector calculated. The normalized outward surface vector is calculated using cross-product algorithm that uses two vectors as inputs. The vectors used in the algorithm are determined from adjacent data points to the base point the surface vector is to be created as shown generally in Figure 22.

Using a projection algorithm the X, Y, and Z point coordinates are offset along the outward normalized surface vector a specified uniform distance.

Figure 23 through Figure 26shows the uniform distance offset performed on the AS file data in the hollow window region.

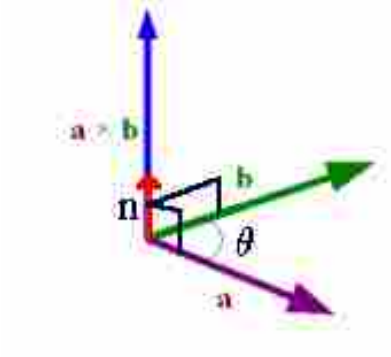


Figure 22: Normalized Surface Vector (n) from Cross-product of (a) and (b)

This operation of applying the cavity protection of the specified region is referred to as Uniform Crush. It is applied to both the airfoil workpiece and applied in reverse to the die geometry.

Deformation Compensation

The second operation is an offset applied only to the airfoil workpiece. The offset adds material to the airfoil to compensate for the plastic deformation over the hollow cavity zone of the airfoil. The algorithm calculates an offset based off of the part and die tolerances, an allowable material deformation percentage, and an estimated amount of deformation. The algorithm adjusts the data points of the workpiece where the deformation would occur along the outward surface normal vector as was describe for Uniform Crush. Figure 27 through Figure 30 shows an exaggerated application of this deformation compensation.

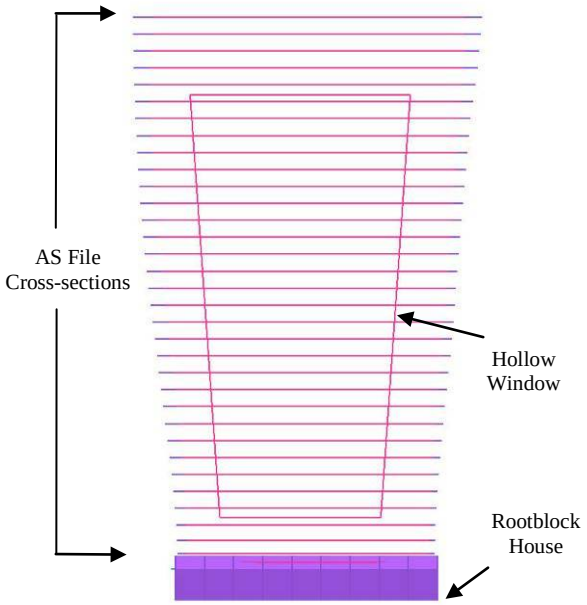


Figure 23: AS File Data Shown with Hollow Window Region (No Offset)

Figure 24: AS File Data Side Profile (0.0” Offset)

Figure 25: AS File Data Side Profile (1.0" Offset)

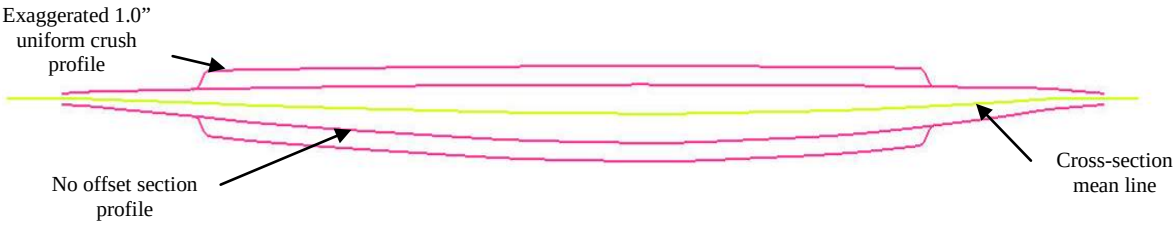


Figure 26: Uniform Crush 0.0" and 1.0" Cross-section Overlap

The estimate of deformation is another one of the design variables used in the optimization routine.

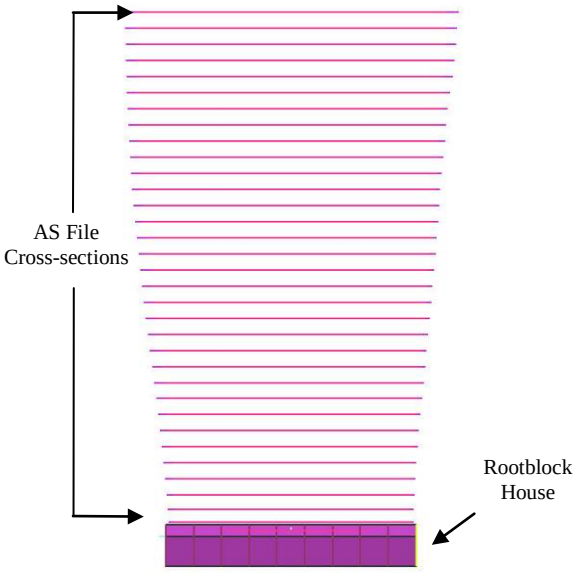


Figure 27: AS File Data Shown as Cross-sections with Rootblock

Figure 28: AS File Data Side Profile (0.0” Offset)

Figure 29: AS File Data Side Profile (0.5" Offset)

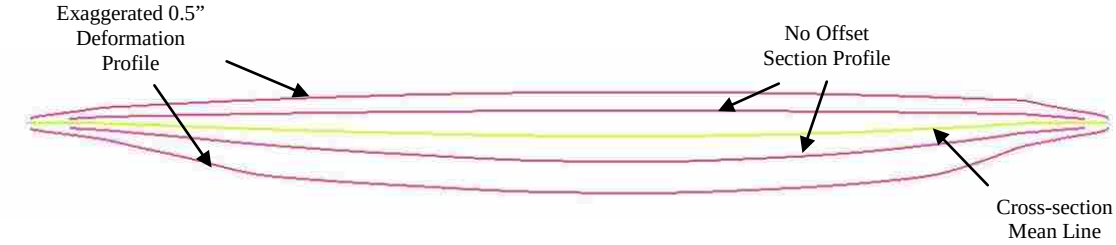


Figure 30: Deformation 0.0" and 0.5" Cross-section Overlap

Deflection Compensation

The third operation used to describe the location of the data points is a compensation for deflection to the die geometry. The die geometry is subject to large loads which lead to deflection (compression) of the die material albeit hardened tool steel. The workpiece geometry is represented by high strength titanium and therefore will slightly alter the forming shape of the dies due to the competing material properties. To compensate for the die deflection, an

algorithm has been created that offsets the die surface data points in a specified region to form what looks like a tent on the die surfaces if exaggerated (See Figure 31 to Figure 33).

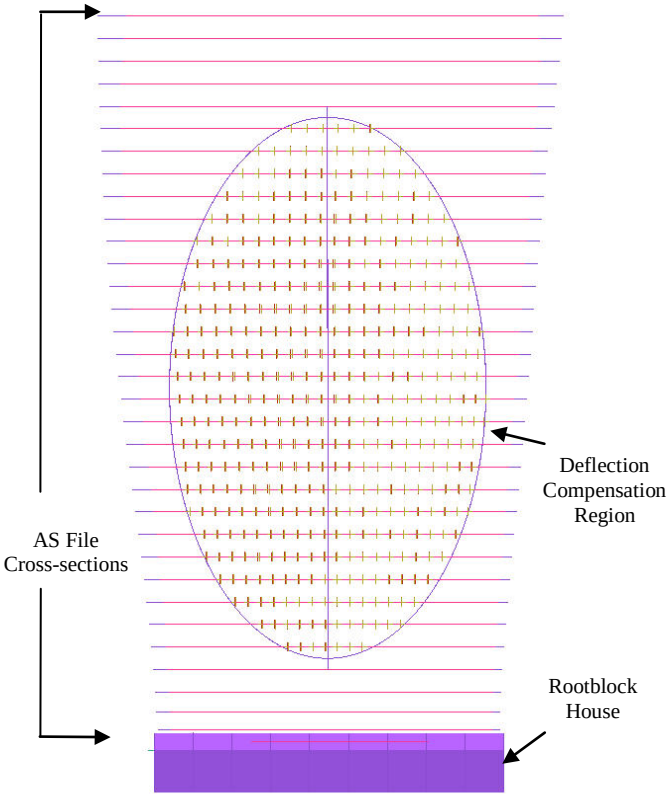


Figure 31: AS File with Oval Deflection Region and Manipulated Interior Points

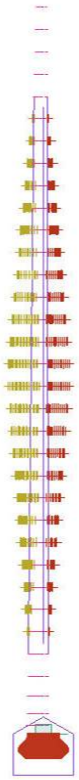


Figure 32: AS File Data Side Profile with Deflection Compensation Applied

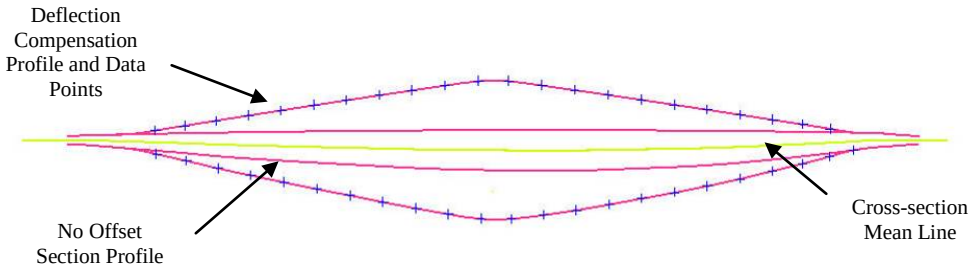


Figure 33: Deflection Compensation Cross-section Overlap

The tent shape is a simplified representation of more complicated region definitions that would be used in an industry application. The intent is to add extra die thickness to the areas where the most die deflection will be observed. We are assuming that the most deflection will occur in the middle of the airfoil geometry by using the tent like zone. By specifying the upper and lower bounds in the spanwise direction (vertical direction), the data points can be assigned an offset value which is determined by position relative to the upper and lower bounds and the apex of the compensation region. In practice the compensation is not visible to the eye and is a minor compensation ensuring the correct blade geometry can be created. Deflection compensation is the third design variable used in this thesis methodology.

Before Module 1 Part 2 is complete and Part 3 of Module 1 begins, the AS Cartesian point data, once manipulated, is used to create the initial cross-section definitions of the airfoil/die surface geometry. Defining a finished airfoil for the process being simulated, the AS file includes leading and trailing edge definitions (See Figure 34 and Figure 35).

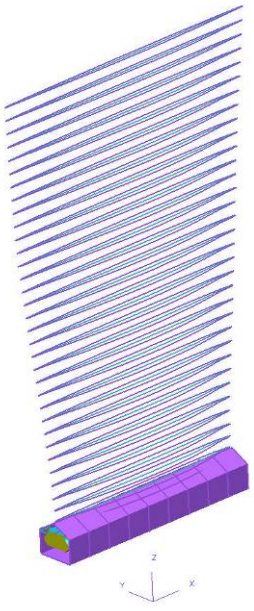


Figure 34: AS File Full Airfoil Definition

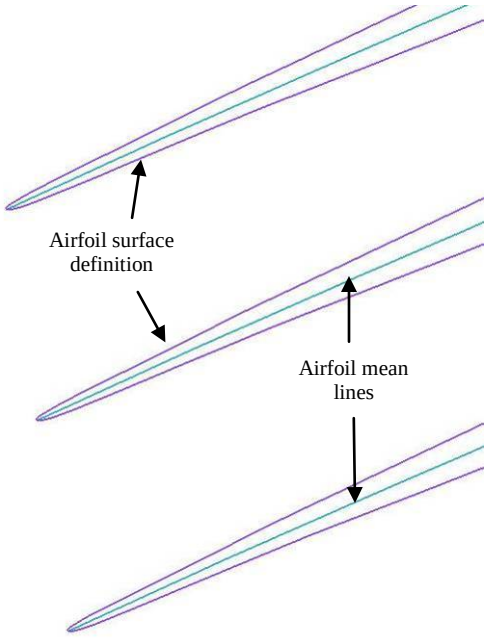


Figure 35: AS File Full Cross-section Definition (Leading Edges Shown Only)

To obtain the in-process airfoil/die geometry where the leading and trailing edges are not necessary, points of departure from the AS file are determined which allow for the proper construction of the die surfaces and in-process airfoil geometry (See Figure 36:).

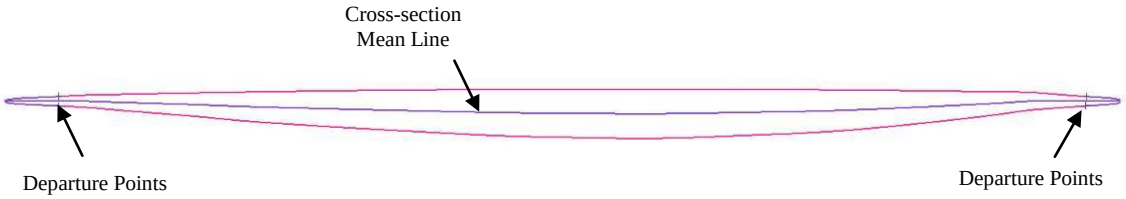


Figure 36: Airfoil Cross-section with In-process Geometry (Pink) and Departure Points

This is done by determining where to depart from the airfoil curvature at the leading and trailing edges for every cross-section following specific rules. Examples of the rules for departure could be a specified distance back along the cross-sectional cord from the leading and trailing edges or when the airfoil begins to experience a predetermined amount of curvature the closer the points of departure approach the leading and trailing edges respectively. With the points of departure identified, new cross-sectional curves are created through the modified AS data from leading edge departure point to trailing edge departure point for both sides of the workpiece and die surfaces.

Module 1 Part 3

Development of Module 1 Part 3 uses the manipulated AS data and horizontal B-splines created through the data for the desired surface profile. Part 3 integrates the AS data and root attachment into a single entity. This is accomplished by creating vertical B-spine curves through the data of the AS file and then locating the intersection point of the curves with the roof of the root block house (See Figure 37). Figure 38 shows the vertical stringers trimmed to the surface

of the rootblock roof. The portion of the vertical stringers extending from the intersection points to the tip of the airfoil definition is retained.

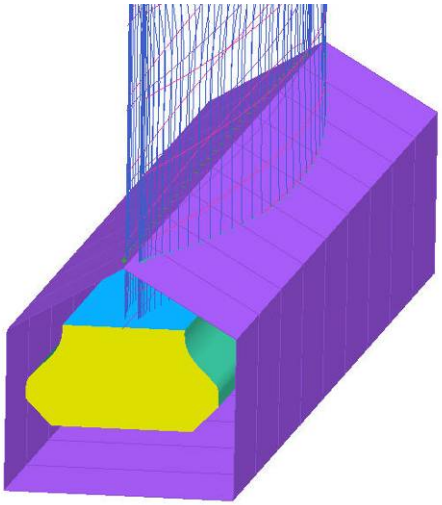


Figure 37: Vertical Stringer Intersection with Rootblock Roof

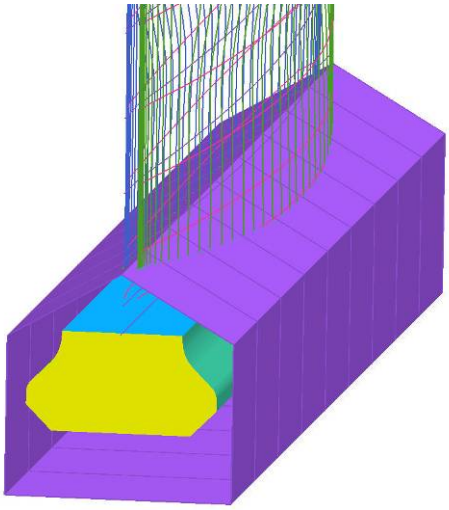


Figure 38: Trimmed Vertical Stringers to Rootblock Roof

These intersections are then used as the starting point for creating the remainder of the vertical stringers wrapped around the rootblock, with hard intersections blended with fillets of specified radius. Figure 39 shows the completed final vertical curves successfully integrating the AS data and rootblock attachment together as one in the vertical direction only.

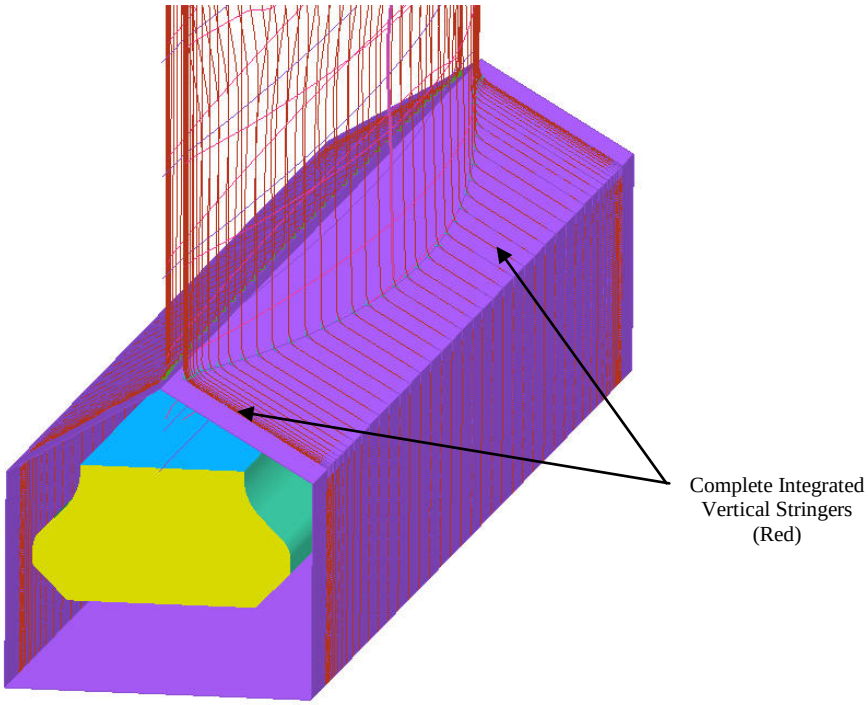


Figure 39: Complete Vertical Airfoil Stringers Integrated Over Rootblock

Module 1 Part 4

The final construction stage of Module 1, Part 4, involves preparing the horizontal cross-section curves for their final configuration. To prepare the horizontal cross-sections, the angle at which the die surfaces must depart away from the airfoil profile is defined along the leading and trailing edges. This departure angle is shown in Figure 41 with linear extension curves moving out and away from the previously determined points of departure (See Figure 40) on both the leading and trailing edges.

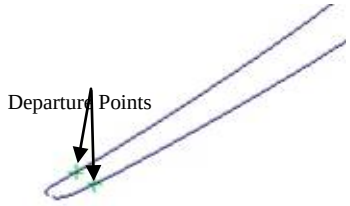


Figure 40: Cross-section with Departure Points Shown

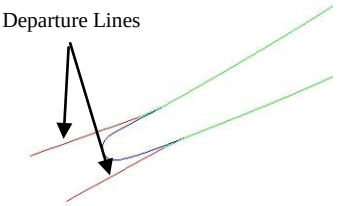


Figure 41: Cross-section Overlaid with Departure Lines (Red)

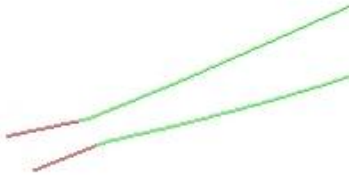


Figure 42: Blended Cross-section (Green) with Departure Lines (Magenta)

When the departure lines have been created with the proper angles and lengths, the departure lines have a blend radius created effectively joining the cross-sections and departure lines together as one (See Figure 42). The joined cross-sections are given a unique identifier and are then ready, along with the vertical stringers, for the Module 2 modeling operations. Part 4 of Module 1 is applied to both the airfoil workpiece and die surface geometry constructs. Table 2 shows the number of files and the lines of code that are required to run Module 1.

Table 2: Required Number of Files and Lines of Code for Module 1

	Number of (.hpp) Files	Number of (.cpp) Files	Lines of (.hpp) Code	Lines of (.cpp) Code
Bond Dies	2	24	805	18,676
Prebond Blade	2	14	805	17,338

Module 2 Development

Module 2 handles the transition of the die geometry from workpiece to the die plane only applied to the die surfaces portion of the automated modeling. Proprietary algorithms were developed that would allow for any cross-section of the die geometry to follow specific rules of

transition such that features as draft angle, transition radius, and die gap are constantly maintained regardless of the cross-section orientation.

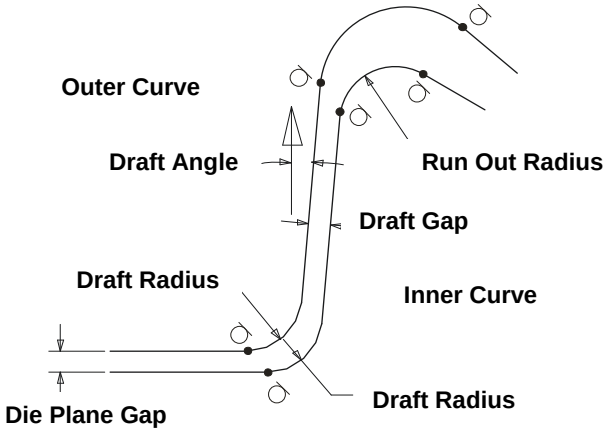


Figure 43: Transition Governing Relationships

Figure 43 shows the governing relationships for the transition**.** Figure 44 shows how the various oriented cross-sections can successfully be transitioned to the die plane thus continuously completing the cross-sections curves for both halves of the die geometry.

Figure 45 details the model as created through Module 2 for an instance of complex free-form die surfaces. Table 3 shows the number of files and also the number of lines of code required to run Module 2 for the airfoil to die plane curve transitions for any curve orientation.

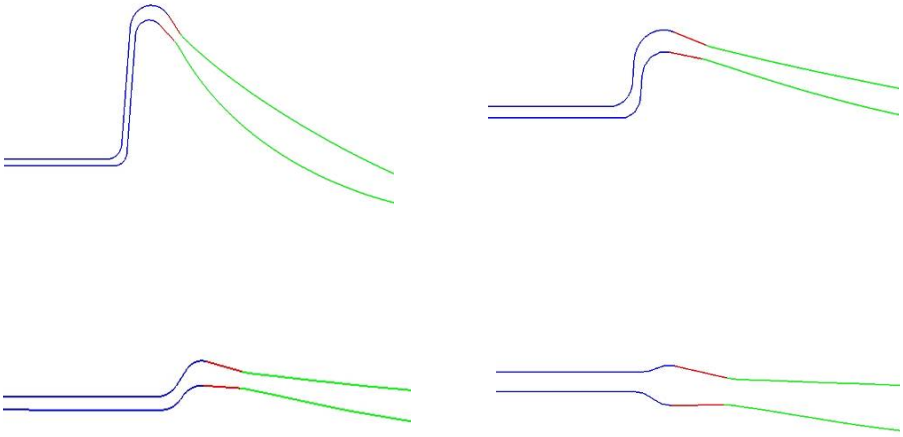


Figure 44: Module 2 Airfoil to Transition Curve Capability

Figure 45: Complete Exaggerated Module 2 Curve Transition Example #1

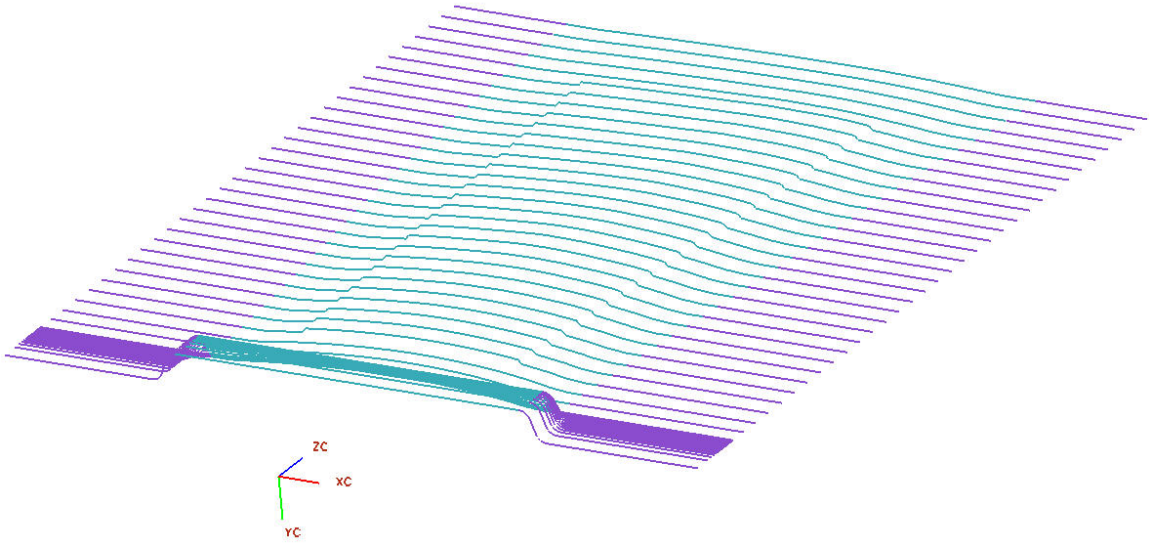


Figure 46: Completed Module 2 Example #2 (1/2 Die Shown for Clarity)

Table 3: Required Number of Files and Lines of Code for Module 2

	Number of (.hpp) Files	Number of (.cpp) Files	Lines of (.hpp) Code	Lines of (.cpp) Code
Bond Dies	2	7	805	15980

Module 3 Development

Module 3 is the lynch pin of the generative modeling operations. This development module generates the net of vertical and horizontal curves for both the work piece and the extended die geometry (airfoil to die plane section curves) in preparation for surface creation. The resulting surfaces are then joined/sewn together to form solid entities which would represent the general state of the tooling/workpiece for the forming operations.

Using a predetermined parameterization (user specified) scheme for multiple regions on the cross-section curves, vertical curves running through the parameterized locations on the cross-section curves can be run from root to the tip of the airfoil geometry. Figure 47 shows seven parameterized regions for the die geometry on the cross-sections.

The vertical and horizontal curves are given identifiers such that a native NX curve mesh surface creation algorithm can be executed. Figure 48 shows the surfaces generated to this point in Module 3 for both the die surfaces and workpiece geometry. The surfaces are now ready to be converted into solid geometry in Module 4.

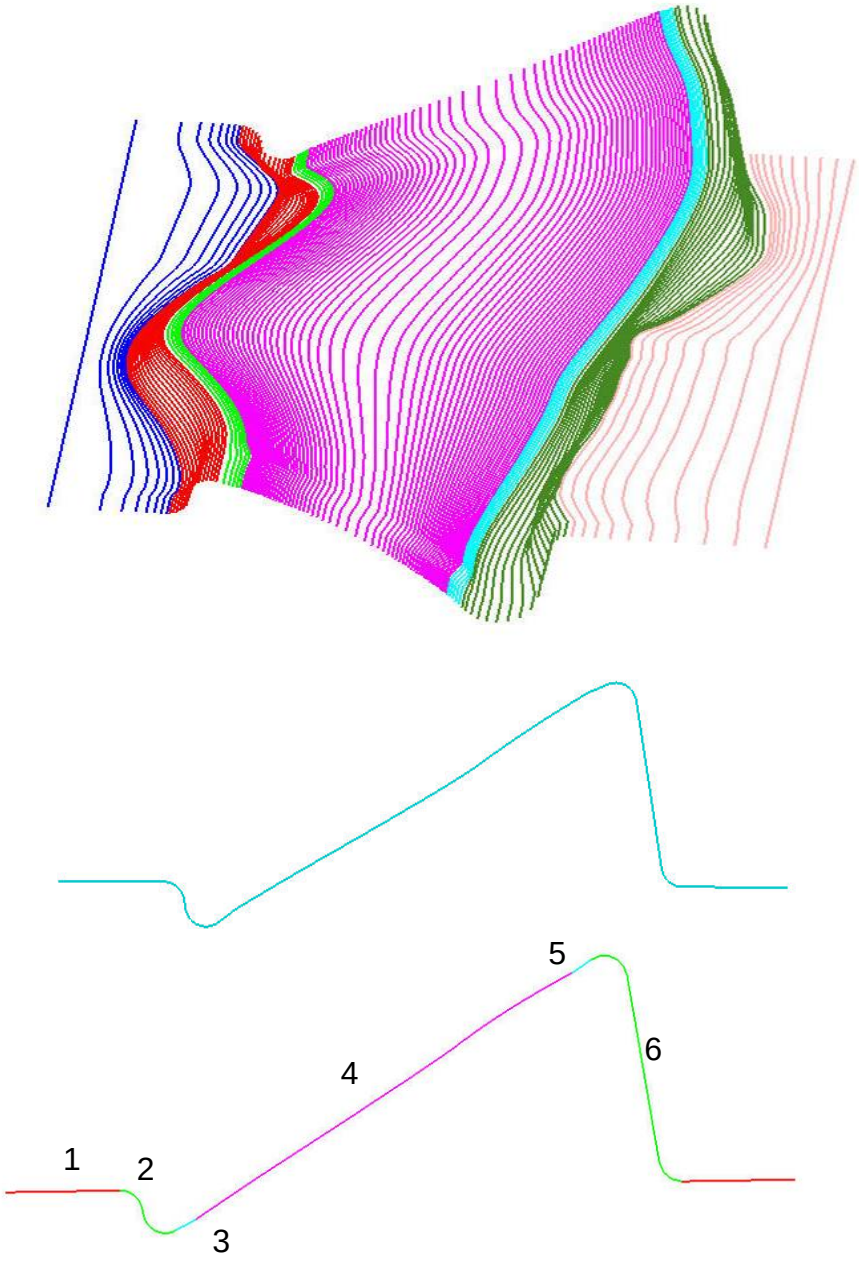


Figure 47: Module 3 Vertical Stringer Parameterized Regions and with Region Biasing

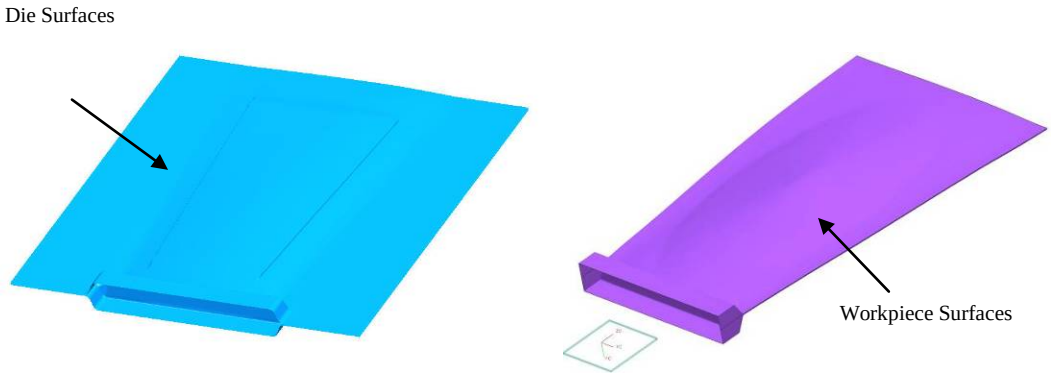


Figure 48: Module 3 Die and Workpiece Surface Examples

Table 4: Required Number of Files and Lines of Code for Module 3

	Number of (.hpp) Files	Number of (.cpp) Files	Lines of (.hpp) Code	Lines of (.cpp) Code
Bond Dies	2	5	805	15360

Module 4 Development

With the die halves and workpiece surfaces from Module 3 generated, their respective solids can be created. The workpiece solid is most easily created. The leading and trailing edge surfaces that close off the gap between the workpiece upper and lower surfaces are created by extracting the edge end points closest to the root and tip on the leading or trailing edges and creating a line object that bridges the leading and trailing edges gap at the root and tip. These two new curves are shown in Figure 49 on the leading edge of the sample airfoil.

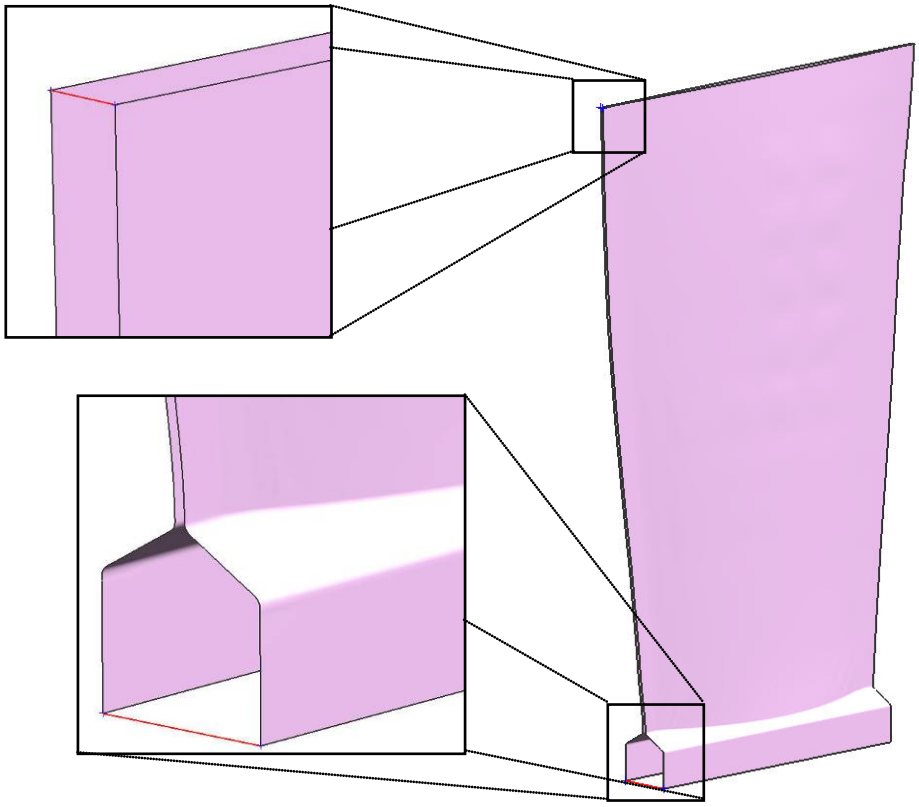


Figure 49: Workpiece Leading Edge Root and Tip Bridging Lines

Using the leading edge curves of the upper and lower workpiece surfaces, a surface sweep function is executed using the newly generated curves as guides for the boundary of the leading edge surface. The same procedure is repeated for the trailing edge. With the four surfaces (workpiece/die interfacing surfaces, leading and trailing edge surfaces) now created, the root and tip cap surfaces (See Figure 50) are created using the same method but by autonomously locating the proper edge curves, using the lines created previously. Figure 50 shows the die interfacing, leading/trailing edge, and root/cap surfaces used to create the workpiece solid in an exploded view.

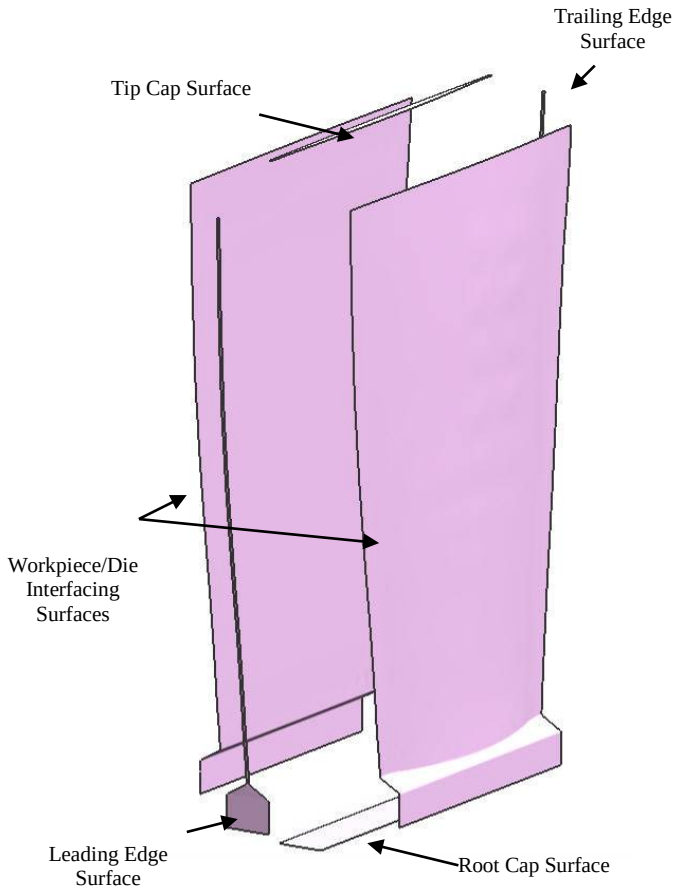


Figure 50: Workpiece Surfaces Exploded View

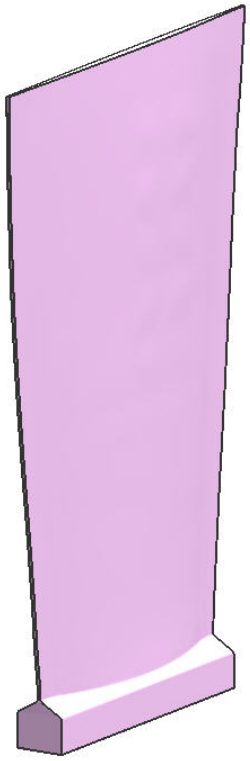


Figure 51: Workpiece as a Solid Body

The surfaces representing the blade are then sewn together using a native sew function in the modeling software to create an airtight entity that produces the workpiece solid. Figure 51 shows the solid workpiece completed and ready for subsequent modeling operations for analysis preparation.

To create the die solids, a minimum die thickness must be observed and therefore the point of maximum height on the die surfaces is located. With the max height of the die surface located an imaginary line is extended in the direction of extrusion equal to the amount of proper die thickness. The four corners of the die surface have lines extended in the same direction that terminate at the height determined by the apex of the imaginary line representing minimum die

thickness. The curves are then connected such that a wireframe of the die solid is generated. Figure 52 shows the wireframe structure developed prior to creating the enclosed volume for the die solids.

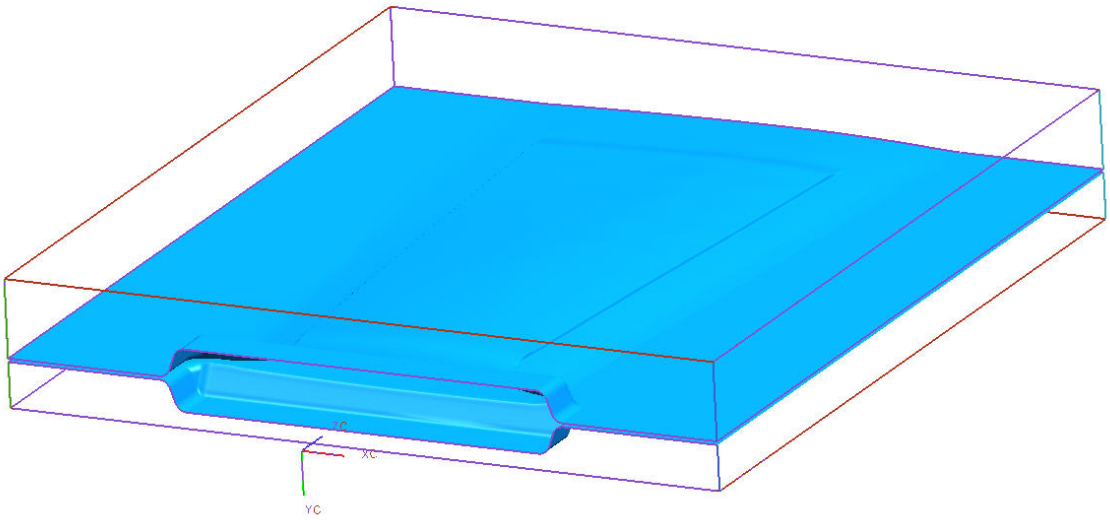


Figure 52: Wireframe of Die Solid with Die Surfaces (Blue)

With the wireframe guide curves in place, surfaces are created in a similar manner as the workpiece surfaces to form an enclosed volume which is sewn together generating the individual die solids. The same procedure is executed for both the dies except in opposite directions. Figure 53 shows the solid bodies generated from the sewn surfaces. The die and workpiece solids are now prepared to be sectioned in preparation for model simulation.

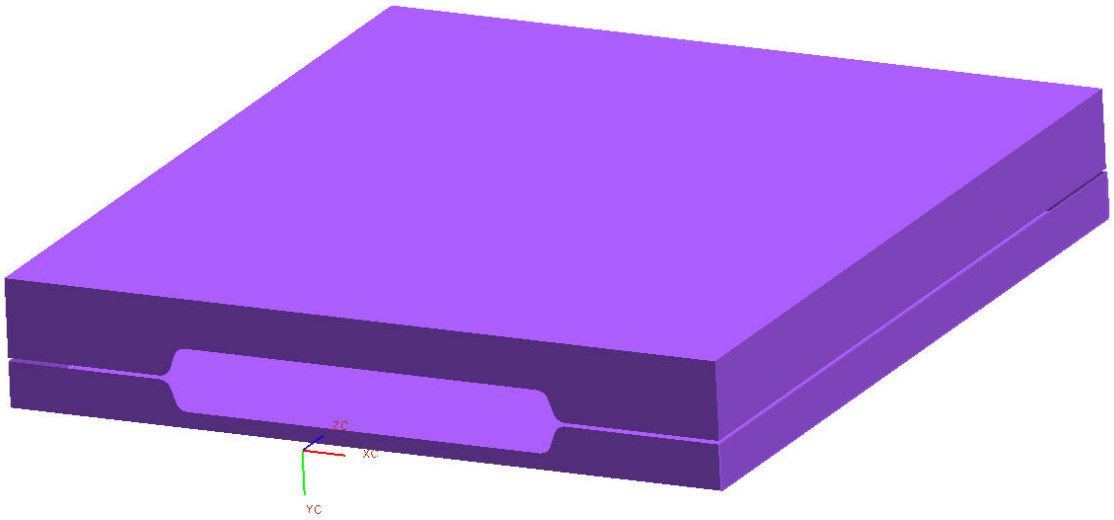


Figure 53: Die Half Solids

With 3D solid die and workpiece geometry now available, they must be sectioned into 2D planar sections as specified by the thesis methodology. The Z height locations of the section cuts in the XY plane are specified in the input file for model construction. Figure 54 and Figure 55 show the dies and airfoil workpiece sectioned.

Creating 2D cross-sections is valid for the methodology developed in this thesis due to the plane strain assumption since the sections of the dies/workpiece taken are long and thin and do not experience large deformation.

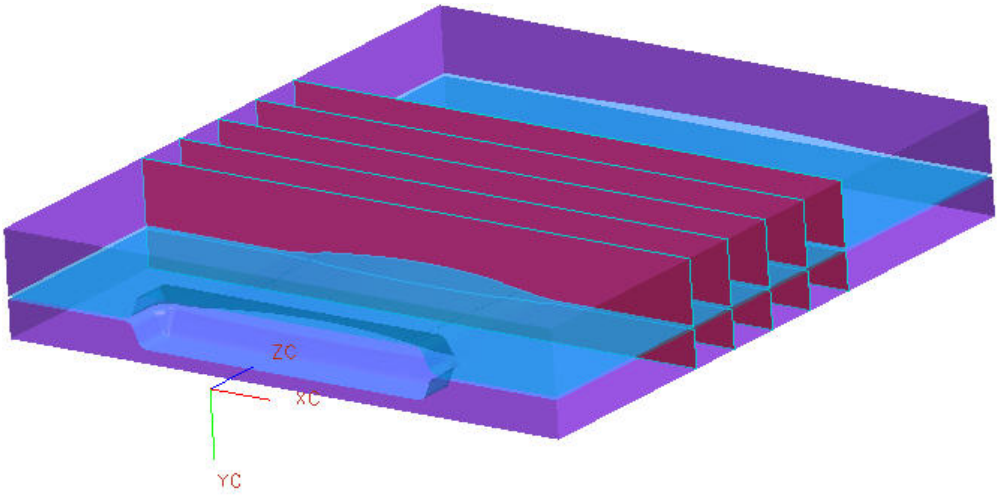


Figure 54: Module 4 Die Cross-section Results

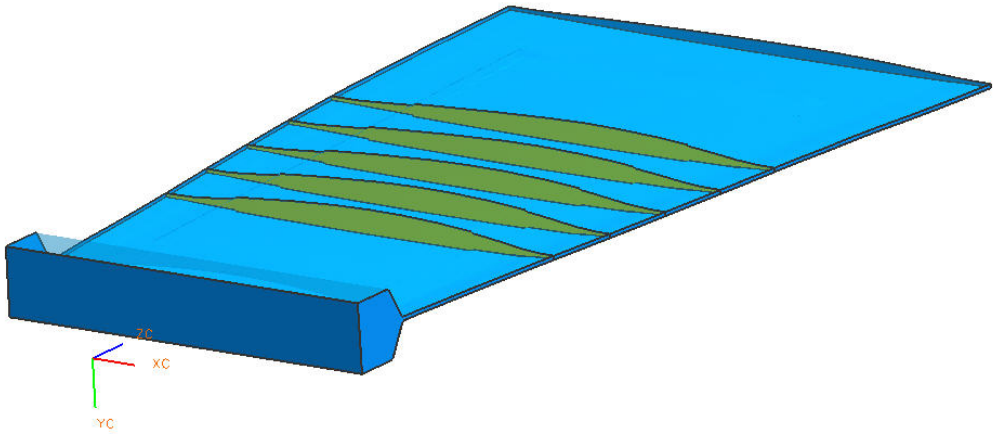


Figure 55: Module 4 Workpiece Cross-section Results

The sections are chosen based off of strategic datum plane placement (equally spaced through span of airfoil body) and then exporting the resulting 2D upper and lower die and

workpiece geometries to the appropriate part files which are saved for subsequent use in the optimization of the forging analyses (See Figure 56).

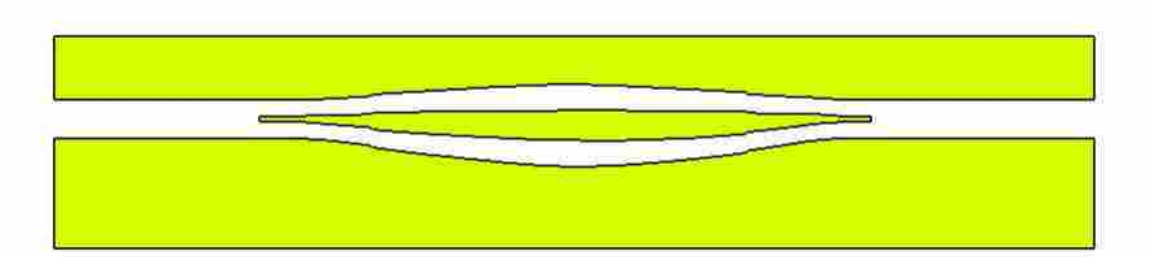


Figure 56: Sample Section Cut for Analysis

Table 5: Required Number of Files and Lines of Code for Module 4

	Number of (.hpp) Files	Number of (.cpp) Files	Lines of (.hpp) Code	Lines of (.cpp) Code
Bond Dies	2	6	805	15454
Prebond Blade	2	4	805	15271

4.1.3 Evaluation

To evaluate the parametric modeling scheme developed in this research not only do the deviations of the simulated model need to be minimized (see Section 3.3) but also some quality criterion need to be addressed. This section discusses the validation criterion for the test studies undertaken for the HFB models and expands on the required capabilities and qualities of the resulting product models. The following sections include the evaluation procedures for the necessary questions asked prior to permitting the models to be used in the optimization routine developed in this research. The questions are namely, does the modeling procedure:

- allow for a large data input?
- create reusable surface models?

- generate high-fidelity geometry?

The above questions are addressed in the sub-sections below.

4.1.3.1 Input Data

The input data for the automated parameterization tool must be capable of building HFB surface models for a variety of AS files and root body part files. The AS files used in this research represent a twisted airfoil placed flat on a level surface (no twist applied). This type of AS file is called an Unwrap AS file and is used primarily as the starting point for the workpiece and die surfaces demonstrated in this research. The bulk data point files all contain at least 5000 ordered input data points that must be read as inputs to the program. This is considered to be a large amount of data input in industry.

To further demonstrate the power of programmatic parametric modeling using CAD API's, examples are provided in Section 5.2 of the modeling parameterization scheme applied to two (2) different HFB airfoils. This shows the power and flexibility of generative CAD parametrics by way of only submitting different input AS files. Section 5.2 contains the results from the test cases performed to test the input capabilities of the HFB APT application. These tests are performed prior to allowing the models to be used with optimization.

4.1.3.2 Reusable Models

The reusable model paradigm allows CAD master models to be manipulated in order to create a new design or instantiation, rather than creating them out of nothing. In the programmatic approach, to reuse a model is to execute a user function or an executable that automatically creates a new design with new model design variables.

The generative modeling scheme automatically assigns parameters, constraints, and relationships to newly created surface geometry. A single input parameter file was used that contains all of the design variables that control the complete construction of the surfaces. This same file contains the optimization design parameters that are varied to locate the optimal parameter settings.

To determine whether reusable models can be generated from the methodology, successful generation of all the required surfaces is necessary for the design parameters investigated.

4.1.3.3 High-fidelity Geometry

An automated parameterization tool for PRSMs must generate geometry that is always consistent. An evaluation algorithm, also used for check surface deviations, has been developed to check surface geometry for proper alignment to the desired nominal geometry. A routine has been implemented which calculates the deviations between all surfaces at strategically chosen locations on the die/workpiece surfaces. See Appendix A for the complete surface deviation algorithm.

4.2 Analysis

The analyses employed for this research were developed to operate in a batch mode environment. They were created to run from a command prompt or from a batch script file. The inputs necessary for the analysis modules are read from text files that store the input information in a separate location thus allowing all phases of the model/analysis development to have access to system parameters. Using this approach simplified the process of integrating all the different

functionalities (modeling, discretizing the domains, and analyzing) into a single seamless program.

The analysis modules that were created to check the validity of the free-form surface models for validity are listed below:

- Mesh Creation
    - Structured Mesh Generation
- Forming Simulation
    - Simulation Preprocessing
    - Solve Forming Simulation
    - Simulation Postprocessing

The engineering software chosen for the system is shown in Table 1. The following sections describe how the analysis modules were developed. The sections also discuss the steps and process for creating each module.

4.2.1 Mesh Creation

For the purposes of this research, the FE mesh generated from the model geometry is created in the ANSYS environment. An ANSYS specific APDL macro is automatically executed in batch which develops custom structured mesh for each NX cross-section part file. The structured mesh file is then formatted and handed off to the analysis software using ANSYS system calls that generate a properly formatted analysis input keyword file and saved into a specific file location.

ANSYS has been chosen to generate the initial mesh for two reasons. The first reason is due to its ability to intelligently generate custom mesh and analysis scenarios. The second

reason is to show that a mesh can be generated in one program, successfully formatted and exported to another program for use. The second reason demonstrates the ability to integrate software to form custom programs across multiple disciplines.

4.2.1.1 Structured Mesh Generation

ANSYS Parametric Design Language (APDL) was used to create the 2D mesh. An APDL macro file (.mac extension), was created and used as the macro script run for every partfile requiring a mesh be constructed (three separate macro files). Following the general rule of having a minimum of at least three elements across the narrowest portion of the objects (Balling 2006), a mesh size was chosen, and the objects were meshed using a mapped meshing routine in ANSYS. Figure 57 below shows an example of the meshed objects used in the forming simulation. It is also important to note that four node quadrilateral elements were used because DEFORM 2D can only use 1st order (4 node) quadrilateral elements.

Once the mesh had been generated, a unique and properly formatted input file for DEFORM was created since a direct connection between ANSYS and DEFORM 2D currently does not exist. The generated files contain, object names, number of nodes, all Cartesian node coordinates, number of elements, and all element connectivity's. This file is then saved with a unique name.

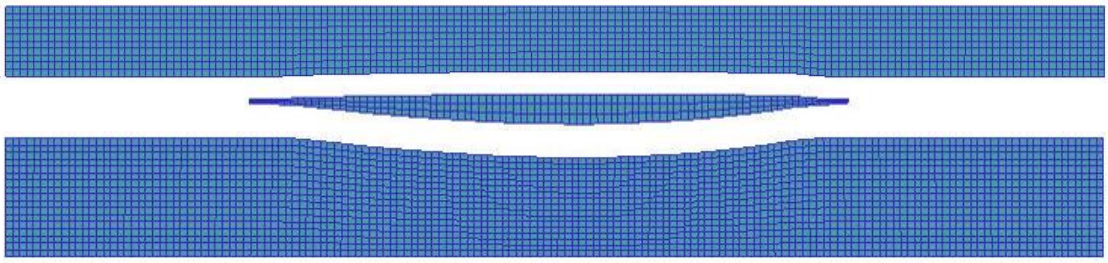


Figure 57: ANSYS Meshed Objects Example

Table 6 outlines the macro file generated for the meshing operation. The complete macro file is located in Appendix B.

Table 6: Mesh Generation Outline

~UGIN	Import NX model directly
ET	Select an element type
AESIZE	Specifies the element size to be meshed
LESIZE	Select the number of elements per edge
AMESH	Meshes the cross-section (area)
*CFOPEN	Opens a file for writing
NODE	Locates node closest to specified coordinate
VWRITE	Writes formatted data to file

4.2.2 Forming Simulation

The analysis/model simulation software of choice for this research was DEFORM 2D (developed by SFTC). DEFORM 2D is a Finite Element Method (FEM) based process simulation system designed to analyze two dimensional (2D) flow of various metal forming processes.

Due to the size and fidelity of the models, a full three dimensional simulation of the forming process considered would require enormous amounts of time. The airfoils under consideration are describing an intermediate manufacturing process before any twist or bending has been applied to the airfoil. Due to the nature of the design process being simulated and the wide die cross-section in comparison to the thickness of the die, a plane-strain assumption could be made to simplify the simulation. This assumption states that relative to the width of the dies, the thickness is small thus allowing the assumption of constant cross-section to be used. This way the DEFORM 2D solver could be used to analyze cross-sections from the blade-die situation.

The forming analysis is to calculate the stress on the dies and workpiece as well as determine the final state of the die and workpiece surfaces during an isothermal forming process. The dies for this research are made out of high quality chromoly tool steel, namely, AISI-H-13 that has been quenched and tempered at 1000 C. The effective die stress must not exceed the material yield strength of 250 ksi. As a perceived standard for the turbine engine industry, the airfoil's material composition is titanium (Ti-6Al-4V). Table 7 shows key material characteristics for the die and workpiece that are pertinent to this research. The workpiece is allowed to plastically deform to achieve the desired form whereas the dies are not.

Table 7: Die/Workpiece Material Properties

Chromoly Tool Steel (AISI-H-13)	**Units (ksi)**
Yield Stress	8000
Titanium (Ti-6 Al-4V)	
Yield Stress	250

4.2.2.1 Simulation Preprocessing

Although a portion of the preprocessing occurs inside of ANSYS (mesh generation), all the system defaults, loadings, and constraints are applied by way of DEFORM macros and the DEFORM 2D text based preprocessor. Once the macro commands have been determined, using the interactive DEFORM text-based preprocessor, the preprocessor can be called from the command line with the macro file used as the command line input. This enables the interactive preprocessor to be bypassed and now run in batch from the command line. The changes required for each simulation need to be written to the macro file before execution to ensure proper

simulation database generation. The updating of the macro file is accomplished using iSIGHT's data exchange program.

To execute the preprocessor program, simulation default boundary conditions, meshed die/workpiece objects, loading conditions, object positioning data, friction data, specified die stop reference objects, and all contact information are imported into the program and then submitted for database creation. Figure 58 shows a preprocessed model with all objects labeled and object contact points shown. The complex free-form part geometries, nodal locations, and element connectivity's are the only portions of the analysis that undergo design iterations for every simulation.

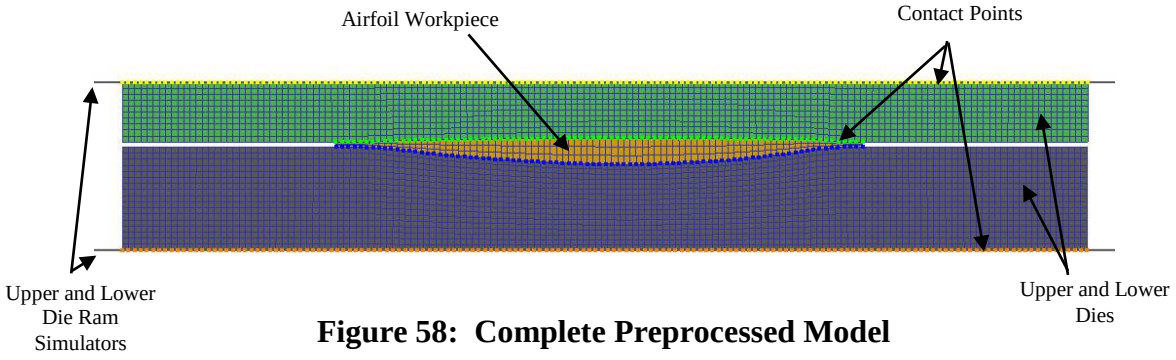


Figure 58: Complete Preprocessed Model

To obtain the stress and displacement on a cross-section the steps in Table 8 must be followed.

Table 8: Simulation Preprocessing

KFREAD	Reads a DEFORM specific keyword file
OBJPOS	Positions objects in contact with each other
CNTACT	Determines the nodes in contact
FRCFAC	Determines the type of friction
GENCTC	Generates contact elements between objects
REFPOS	Specifies a reference object
KFWRITE	Writes a keyword file based on current status
GENDB	Generates simulation database

Appendix C shows the complete macro file along with the fields of the macro that are updated with each optimization run (highlighted in blue).

4.2.2.2 Solve Forming Simulation

Once preprocessing has been completed and the DEFORM database file generated, the simulation is run directly from the command line. This is accomplished by executing DEFORM's simulation control script from the command line with the database file as the input. Since the simulations need to be run sequentially due to the single DEFORM license available for this research, the simulations are run in a sequential batch mode. Once a solution has been obtained, the solution database file containing the results are saved for further interrogation and data extraction.

4.2.2.3 Simulation Postprocessing

Postprocessing takes place in two (2) parts. Part 1 takes place in two steps and Part 2 a single step. DEFORM is primarily an interactive program requiring large amounts of user interaction. Since the program needed to run in a batch mode, all of the interactive actions for postprocessing were automated. Step 1 of Part 1 requires the database to be queried and to extract the final time step of the solution simulation (final position of the forming operation). This is accomplished with the system keywords shown in Table 9.

Table 9: Simulation Postprocessing Step 1

DBREAD	Reads a specified time step from solution
KFWRITE	Writes a keyword file based on current status

Step 1 allows for the extraction of the final time step where it is saved off as the results file (.key extension) for the particular simulation (see Figure 59). Complete results of the optimization are presented in Chapter 5.

Figure 59: Postprocessed Final Simulation Time Step (Effective Stress Plot)

Step 2 of Part 1 is more involved. Due to the fact that the geometry was meshed in ANSYS and imported into DEFORM, the object boundaries are unknown to DEFORM. Using the text-based preprocessor and a sequence of static commands written in a text file, the preprocessor utilizes the interactive text based preprocessor to extract the boundary edges (nodal coordinates) and write them to the output file previously created in Step 1 of Part 1. Appendix D shows the inputs to the text based preprocessor used to extract the peripheral geometry used for evaluation and deviation calculation.

Part 2 of postprocessing is executed after all simulations have been completed and all result files have been saved for the design iteration. Part 2 consists of cycling through the result files and locating the maximum effective stress (Von Mises Stress) and maximum principle stress (X and Y directions only) for each object in the simulation. The objective of Part 2 is to identify any simulations that violated any of the material max yield stress conditions. The dies are not to yield but the airfoil blade must move plastically but not yield. Yielding of the airfoil would reduce the blade loading capability if exceeded. The maximum effective stress (Von Mises Stress) is calculated as shown in Equation 10 below.

$$MaximumEffectiveStress = \sqrt{\sigma_x^2 + \sigma_y^2 - \sigma_x\sigma_y + 3\tau_{xy}^2} \quad (10)$$

The extraction and calculation of the max stress conditions for the dies and workpiece are calculated in the following code fragment.

```
    .
    .
    .

34  count=0;
35  FILE* outFile;
36  outFile = fopen("stressOutput.txt","w");
37  for(int i=0;i<1;i++)      //0 for PS and 1 for SS
38  {
39      if(i==0){strncpy(side,"cordwise",9);}
40      else{strncpy(side,"spanwise",9);}
41      printf("%s\n",side);
42
43      //numCordSecs because of the
44      //numCordSecs section cuts
45      for(int j=0;j<numCordSecs;j++)
46      {
47            sprintf(filename,
48                  "..\\Simulation\\I-O\\
49                   OUTPUT_%s_%d.KEY",side,j);
50            printf("%s\n",filename);
51
52            FILE *outputFile = fopen(filename,"r");
53            if(outputFile==NULL)
54            {
55                  printf("Output file
56                          '%s' to be read is not
57                         correct\n",filename);
58                  return 5;
59            }
60            //Iterate 3 times since 3 objects with stresses
61            fprintf(outFile,"%s_%d\n",side,j);
62            fprintf(outFile,
63                    "Obj#\tNode#\tMaxEffStress\
64                       tPrDir\tMaxPrStress\n");
65            for(int k=1;k<4;k++)
66            {
67                  fgets(val,100,outputFile);
68                  while(strncmp(val,"STRESS",6)!=0 && count<1000000)
69                        {
70                              fgets(val,100,outputFile);
71                              count++;
72                        }
73                        printf("\n%s",val);
73
74                  sscanf(val,"STRESS %s %s",temp,temp1);
75                  printf("%s %s\n",temp,temp1);
76                  ObjNum = atoi(temp);NumDataPnts = atoi(temp1);
```

```
77                  for(int m=1;m<NumDataPnts+1;m++)
78                  {
79                        fscanf(outputFile,"%s %s %s %s %s\n",
80                              effStressNodeNum,sigX,sigY,sigZ,tauXY);
81
82                        stressEff = sqrt(pow((atof(sigX)),2)+
83                              pow((atof(sigY)),2)-
84                              (atof(sigX))*(atof(sigY))+
85                              3*pow((atof(tauXY)),2));
86
87                        if(stressEff>maxEffStress)
88                        {
89                              //Extracting the max stress
90                              maxEffStress = stressEff;
91                              maxEffStressNodeNum =
92                                  atoi(effStressNodeNum);
93
94                              if(k==2 && stressEff>objTwoMax)
95                              {
96                                    objTwoMax = stressEff;
97                              }
98                              if(k==3 && stressEff>objThreeMax)
99                              {
100                                   objThreeMax = stressEff;
101                             }
102                       }
103
104                       //maxPrincipleDir 1=X, 2=Y, 3=Z
105                       if(abs(atof(sigX))>maxPrincipleStress)
106                       {
107                             maxPrincipleStress = atof(sigX);
108                             maxPrincipleDir = 1;
109                       }
110                       if(abs(atof(sigY))>maxPrincipleStress)
111                       {
112                             maxPrincipleStress = abs(atof(sigY));
113                             maxPrincipleDir = 2;
114                             if(k==1 && abs(atof(sigY))>objOneMax)
115                             {
116                                   objOneMax = abs(atof(sigY));
117                             }
118                       }
119                       if(abs(atof(sigZ))>maxPrincipleStress)
120                       {
121                             maxPrincipleStress = abs(atof(sigZ));
122                             maxPrincipleDir = 3;
123                       }
124                 }
125                 fprintf(outFile,
126                       "%d\t%d\t%lf\t%d\t%lf\n",
127                       k,maxEffStressNodeNum,maxEffStress,
128                       maxPrincipleDir,maxPrincipleStress);
129                 maxEffStress = 0.0;
130                 maxEffStressNodeNum = 0;
131                 maxPrincipleStress = 0.0;
132                 maxPrincipleDir = 0;
133                 count=0;
```

```
134           }
135           fclose(outputFile);
137           fprintf(outFile,"\n");
138      }
139   }
140   fprintf(outFile,"Obj#1MaxPrincipleStressYDir:\t%lf\n",objOneMax);
141   fprintf(outFile,"Obj#2MaxEffectiveStress:\t%lf\n",objTwoMax);
142   fprintf(outFile,"Obj#3MaxEffectiveStress:\t%lf\n",objThreeMax);
144   fclose(outFile);

.
.
.
```

Shown above is the calculation of the effective stress (line 82 to 85) for the two dimensional results from DEFORM 2D. Once the maximum stress conditions have been located, they are written to a separate file (line 140 to 144) to later be tested against the constraints of the optimization.

4.3 Numerical/Geometric Surface Interpolation

The overall objective of this research is to determine the remaining deviations after simulation on the complex free-form surfaces representing the dies and workpiece. With the simulations complete and the data postprocessed, the results must be compared against the nominal design intent for accuracy. The calculations were done inside of a NX custom executable.

The program for determining the deviations takes place in three (3) steps. Step 1 imports the nominal AS file into NX and constructs an interpolated geometric representation of the die and blade surfaces. The nominal surface is split into two separate surfaces representing the top/bottom die and the left/right side of the workpiece. Importing the text file containing the intersection locations (same input file used for model generation), the intersect locations are projected onto the surface half closest to the data point in the direction of die movement. The

points now on the nominal surface are reference points used to calculate surface deviations with the simulated surface results.

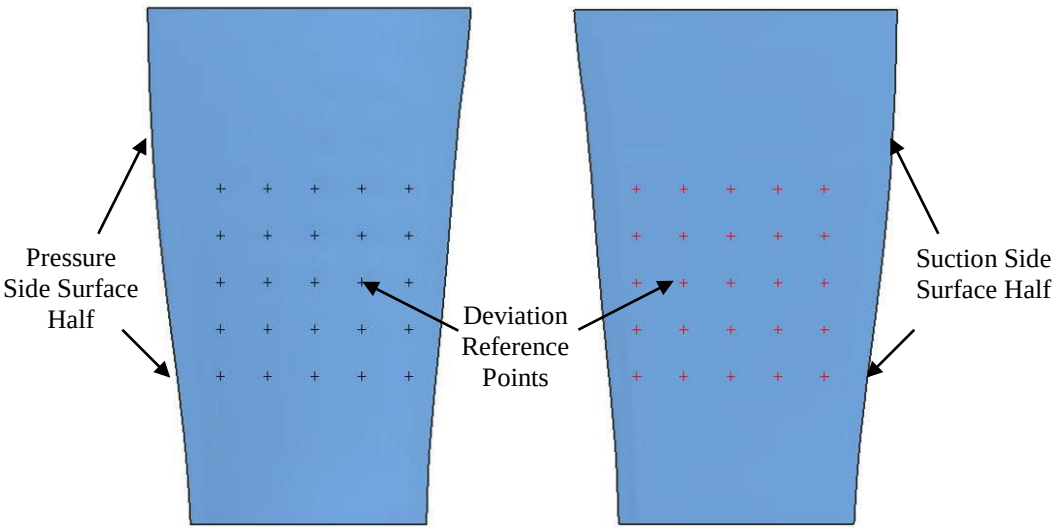


Figure 60: Example of Nominal Surface Halves with Deviation Reference Points

Step 2 consists of reading in the peripheral geometry of the die and workpiece geometry. The border of the geometry is stored in the output file from the DEFORM simulation using the proper notation. To simplify the deviation calculations, a single B-spline is created through the data points of each object. The curves created for each object are then used in conjunction with a direction vector in the direction of the die movement and the reference points of the nominal surface to perform intersection routines. The intersection algorithm determines the distance the simulated surface (in the direction of die movement) is from the nominal surface. These distances are then saved in the database for use in Step 3, calculating the deviations.

The final step, Step 3, of the numerical/geometric surface interpolation is to determine the global fitness value for the deviations. The global fitness value is a final representation of all the deviations found for the design iteration. The calculated value is used as the objective value

for the optimization iteration. The equation below shows the analytical representation for determining the fitness value.

$$F = \left(P_i - T_i\right)^m \tag{11}$$

F Iteration deviation fitness value,
P_i Deviant data point of simulated surface,
T_i Data point used as reference on nominal surface,
m Deviation weighting factor.

The C/C++ function below implements Equation 11 above in a programmatic form. It is important to note that $\left(P_i - T_i\right)$ will already have been calculated and inputted as the vector of deviations. The deviation weighting factor used for the optimization is m = 2.

```
double calc_global_deviation(std::vector<double> &deviations)
{
	int i=0;
	int size=(int)deviations.size();
	double sum=0.0;
	for(i=0;i<size;i++)
	{
		sum = sum + (deviations[i]*deviations[i]);
	}
	return sum;
}
```

Once the global deviation fitness function has been calculated for the design iteration, the deviations and the fitness values are written to a text file for use by the optimization routine.

4.4 Optimization

After the all the modeling and analysis modules have been created, the optimization of the entire system is ready to be constructed. The process of creating the optimization loop involves linking the modeling, simulation, and evaluation modules together. In conjunction with

the modules, design variables need to be selected, optimization algorithm and associated settings determined, design variable limits prescribed, determine the appropriate objectives, and identify the constraints. The objective of the design scenario implied for this research is to minimize the deviations from the simulated surfaces in comparison to the design intent (nominal surfaces). In order to reduce the computation time to perform the optimization, a plane-strain assumption was made which suggests that using long and thin cross-sections of the overall 3D model will produce allowable results at a much more efficient pace. This assumption allowed the use of 2D analysis to increase the efficiency of the optimization loop. The following sections discuss the steps of the optimization process.

4.4.1 Master Program

The optimization of the HFB airfoil and die surfaces required that the modeling, analysis, and evaluation modules be run in sequence. These modules were integrated into the master program. The master program of the design optimization scheme links the modeling, meshing, and analysis together so that they can be executed independently from the optimization if desired. A generalized schematic of the master program showing the integration of the modules is found in Figure 61.

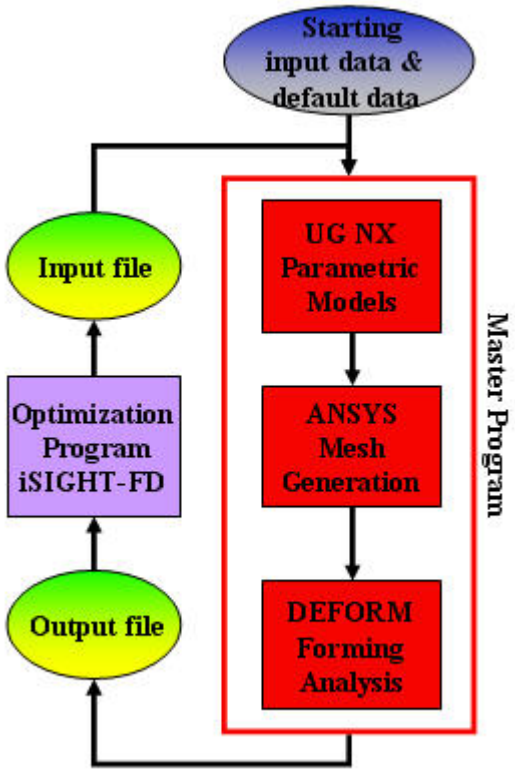


Figure 61: Generalized Optimization Program

4.4.2 iSIGHT-FD Optimization Environment

After all the modules have been created and set up to prepare their output to represent the proper input to the downstream program the optimization master program needs to be constructed within the iSIGHT-FD environment. This was done using iSIGHT-FD's graphical user interface. The first step in setting up the optimization was to establish a task plan. The task plan is the loop that the optimization will perform to evaluate a design and then return the results. The task plan can consist of as many operations that are needed to handle your iteration scenario. The optimization loop used for the forming analysis is shown in Appendix E.

The next step is to establish limits for the design variables, add constraints, and specify an objective. The last step is to specify which type of algorithm to use. A generalized reduced

gradient (GRG) algorithm was chosen due to its ability to enforce feasibility of the topology due to the imposed constraints and follow a “path of steepest descent/ascent” to the design optimum.

The optimization was run with three design variables, three constraints of yield stress conditions (one per object in simulation) and an objective function to minimize the deviation fitness value (see Section 4.3).

4.5 Response Surface Analysis Methodology

It became apparent during initial simulation trial runs that a complex design space could possibly exist. An experimental Design of Experiments (DOE) was necessary to properly explore and characterize the design space. A Response Surface would be used to show the estimated response over the variable ranges. To develop a surface profile to predict the model response a quadratic equation would need to be constructed. Factorial designs commonly only allow for estimation of all main effects and interactions. In order to get a quadratic equation (the squared terms for each factor X_i) an estimation is made using a set or axial points (star points) and center points. The axial points and the center points essentially are a set of one-at-a-time experiments with three levels of each of the independent variables.

An Inscribed Central Composite RSM design was chosen due to its ability to produce a response surface using fewer experiments than a full factorial design with sufficient fidelity. Another reason an “inscribed” central composite design was chosen is that the design variables have defined upper and lower limits that could not be altered. The inscribed method allows axial points to be at the limits for the design variables but not exceed them. Figure 62 is a visual of the Inscribed Central Composite design.

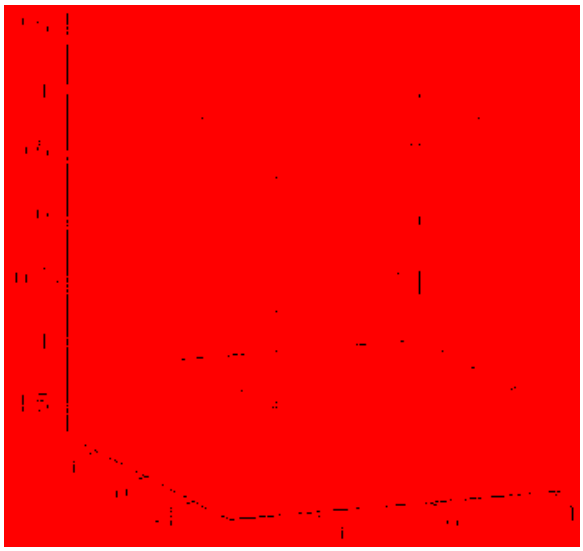


Figure 62: Inscribed Central Composite Design

Central Composite designs contain an imbedded factorial (fractional factorial) with center points and a group of “star points” that allow for estimation of curvature. This requires having 5 levels for each design variable.

The limits for the three design variables (Uniform Crush, Deflection Compensation, and Deformation Compensation) are true limits and exploring outside of the maximum and minimum values would be impossible due to model design constraints. Figure 63 shows the design variable ranges and mapped 5 levels for each variable.

With the variables mapped to the five factor settings the Inscribed CC design could be created for both test cases (see Figure 64).

Legend	X1 Deflection Compensation	X2 Deformation Compensation	X3 Uniform Crush
-α	0	0	0
-1	0.001	0.001	0.001
0	0.002	0.002	0.002
1	0.003	0.003	0.003
α	0.004	0.004	0.004
	0.005	0.005	0.005
	0.006	0.006	0.006
	0.007	0.007	0.007
	0.008	0.008	0.008
	0.009	0.009	0.009
	0.01	0.01	0.01
	0.011	0.011	
	0.012	0.012	
	0.013	0.013	
	0.014	0.014	
	0.015	0.015	
	0.016	0.016	
	0.017	0.017	
	0.018	0.018	
	0.019	0.019	
	0.02	0.02	
	0.021	0.021	
	0.022	0.022	
	0.023	0.023	
	0.024	0.024	
	0.025	0.025	
	0.026	0.026	
	0.027	0.027	
	0.028	0.028	
	0.029	0.029	
	0.03	0.03	

Figure 63: Design Variable Ranges and Inscribed Central Composite Mapping

JMP statistical analysis software was used to perform the linear regression analysis to determine the best fitting quadratic equation for the data. Once the quadratic equation was obtained through regression the equation was plotted over the design variable ranges to give a graphical representation of the design space. This allowed the optimum design variables values to be obtained and verified with the optimization routine.

	Run #	X1 Defl Comp	X2 Defor Comp	X3 Unl
Factorial	1	0.007	0.007	0.002
	2	0.023	0.007	0.002
	3	0.007	0.023	0.002
	4	0.023	0.023	0.002
	5	0.007	0.007	0.008
	6	0.023	0.007	0.008
	7	0.007	0.023	0.008
	8	0.023	0.023	0.008
Star Points	9	0	0.015	0.005
	10	0.03	0.015	0.005
	11	0.015	0	0.005
	12	0.015	0.03	0.005
	13	0.015	0.015	0
	14	0.015	0.015	0.01
Center Points	15	0.015	0.015	0.005
	16	0.015	0.015	0.005
	17	0.015	0.015	0.005
	18	0.015	0.015	0.005
	19	0.015	0.015	0.005
	20	0.015	0.015	0.005

Figure 64: Inscribed Central Composite Design (Test Cases 1 and 2)

CHAPTER 5: DISCUSSION OF RESULTS

The objective of this thesis was to develop a methodology that applies an automated tooling design scheme that integrates commercial CAD/CAM/CAE technologies. The intent is to obtain tooling and pre-formed work piece geometries optimized to produce near-to-design intent products. This was accomplished by developing an automated generative parametric modeling scheme and integrating it with analysis and optimization capabilities.

This chapter presents the results from the die/workpiece optimization implementation as laid out in Chapter 4 as well as the results of the DOE and subsequent response surface.

5.1 Test Cases/Concept Generation

5.1.1 Test Case 1: Double Sided Machined Airfoil

Test case 1 is based on an AS file that contains the nominal (post forming) definition of an airfoil that is designed to receive machining operations on both sides of the airfoil. It is an airfoil that is symmetric about the modeling 'X' axis. The reasoning for machining both sides is to obtain the desired surface thickness over the hollow cavities of the blade. This airfoil has 34 cross-sections with 112 data points defined per cross-section. Figure 65 and Figure 66 show the distinct cross-sectional shape and form the airfoil of interest possesses.

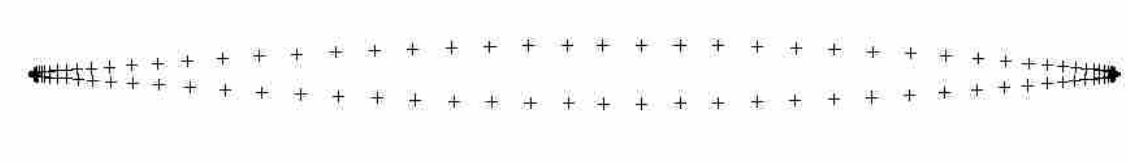


Figure 65: Double Sided Cross-section Top View

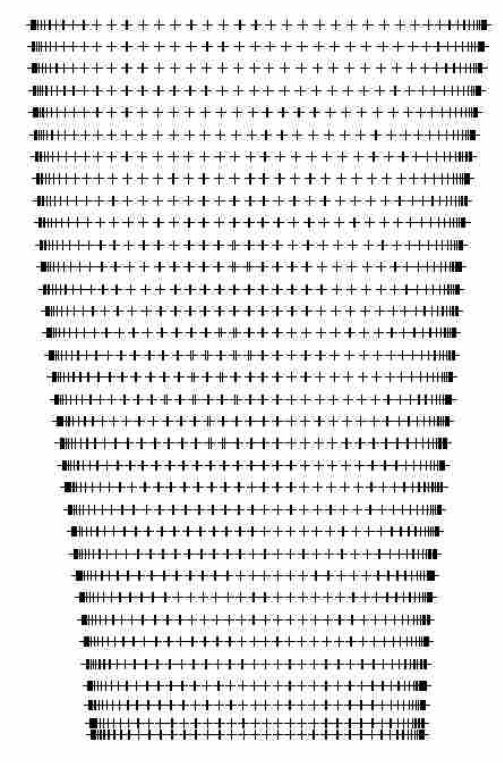


Figure 66: Double Sided All Cross-sections Front View

5.1.2 Test Case 2: Single Sided Machined Airfoil

Test case 2 is represented the same as Test case 1. The profile is slightly altered to represents a conceptual method to manufacture an airfoil, namely machining a single sided of the airfoil. In other words one side of the airfoil is formed to near net shape while the opposite receives machining operations to obtain proper airfoil surface to hollow cavity thickness. This

airfoil has 35 cross-sections with 112 data points defined per cross-section. Figure 67 and Figure 68 show the distinct cross-sectional shape and form the single sided machined airfoil possesses.

Figure 67: Single Sided Single Cross-section Top View

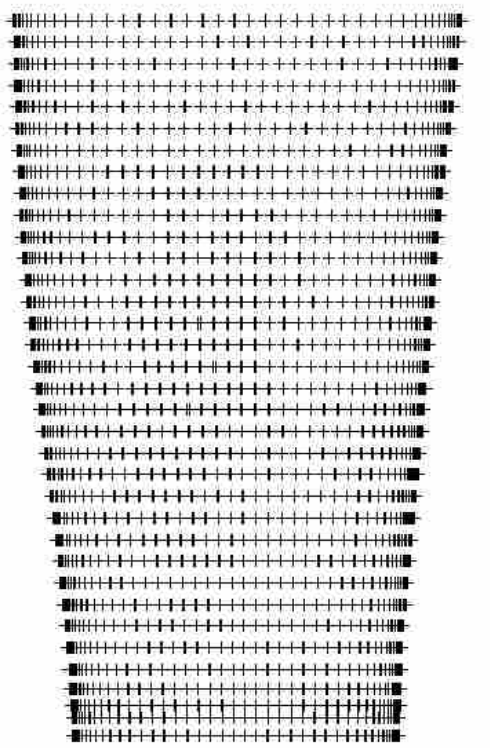


Figure 68: Single Sided All Cross-sections Front View

5.2 Results: Parametric Modeling

The modeling of the die and workpiece geometry followed Section 4.1. The die and workpiece CAD geometry was successfully created using the generative parametric approach. Approximately 100,000 lines of code (see Table 10) were used to develop the generative

parametric modeling scheme. When writing the code to handle the modeling of both the airfoil and the forging dies it was found that approximately 70% of the code was common. The remaining 30% of the code was specific to either construction of the airfoil or the forging dies. Executing the model creation via a "behind the scenes" program proved to be a fairly simple task since the code was tested in a piecewise fashion in the GUI prior packaging for test case use. Running a visual execution within the NX environment was an effective method for debugging the code prior to packaging for standalone execution.

Figure 69 shows an instance of the solid geometry created for each test case.

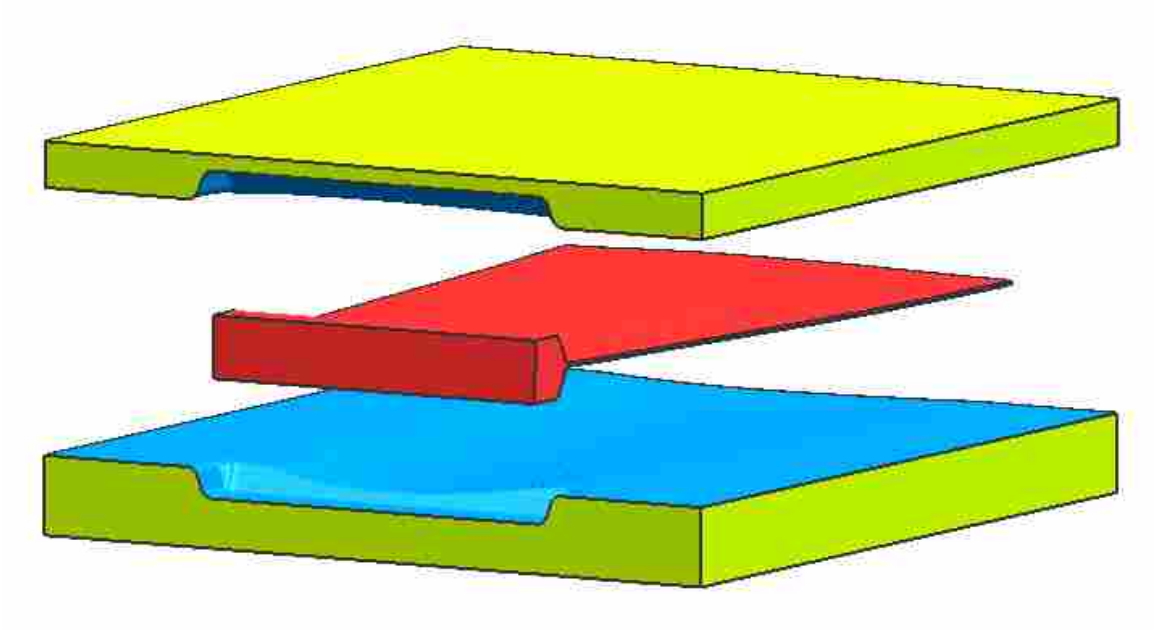


Figure 69: Single and Double Sided Solid Die and Workpiece Geometry

A minor challenge was identified during the sectioning of the solid geometry which was to identify a sectioning method that would successfully replicate the sectioning of the model for every desired cross-sectional cut. There is a native sectioning tool in NX that is frequently employed for sectioning activities. It was found that although a solid model was created that the

section tolerances were excessive and a closed curve defining the 2D perimeter of a cross-sectional piece was not always obtainable (would require joining operations to create a continuous curve). An alternative method using datum planes to create intersections with the solids proved to be successful since a projection of the solid models intersection with the datum would always produce a continuous closed curve. The closed curve was extracted and a bounding plane surface was (required for area meshing operations) created that would be exported and uniquely identified for downstream analysis (See Figure 70).

Figure 70: Cross-sectional Airfoil and Die Geometry Used for Analysis

It was estimated by the Hollow Fan Blade Design Engineering group at Pratt & Whitney (East Hartford, CT) that manual modeling, employing as much parameterization as was possible for the complexity of the design intent, would take approximately 2 man months (~320 man hours to develop) and would be plagued with update errors. Executing the generative parametric scheme (on average) required 12 minutes to model the 3D geometry and save off the cross-

sections as individual files for downstream operations. This was characterized by Pratt & Whitney as an extreme success and modeling times could be decreased when run on faster computer processors with more RAM than that which was used for this research.

To develop the automated program for model development, code development was extensive. Table 10 has some simple statistics such as number of files and number of lines of code that were required for generating the airfoil and die geometry. It is also estimated that an individual with extensive programming and CAD API background could reduce the file/line count although lines of code would still remain on the order of thousands. Given the amount of coding required, serious consideration must be taken prior to proceeding down this path although not every application will be as complicated as the one presented in this research.

Table 10: Modeling Workspace Statistics

Airfoil	*.cpp	*.h/*.hpp	Total
Files	25	2	27
Lines of Code	32,609[1]	805[2]	33,414
Dies	*.cpp	*.h/*.hpp	Total
Files	14	2	16
Lines of Code	65,470[1]	805[2]	66,275

[1] Same *.h/*.hpp files used for Airfoil and Dies
[2] X # files are common to both the Airfoil and Dies

A few simple questions should be asked while deciding if generative parametrics is viable for the modeling application under consideration:

1. Will the model be used repeatedly?
2. Is the parameterization scheme necessary to build the model overly complicated and prone to errors?
3. Is manual model development extremely time intensive (more than 80+ man hours)?

4. Are modeling operations executed in series therefore preventing parametric modeling updates?

Answering these questions will help the designer decide if a traditional modeling approach can be taken or a generative parametric scheme would be appropriate.

The development of in process airfoil workpieces and the associated forging dies meet the above criteria. Some key results are outlined below:

1. Die/blade geometry created easily and efficiently with no modeling errors
2. Design variables easily read and adjusted via text based input file
3. Hundred of unique parametric design variables built into the code to control unique features of the models autonomously

One of the objective questions (See Section 1.1) of this thesis was whether or not the implementation of the methodology can produce a significant reduction in design/engineering tool development lead time. This can be answered in the affirmative due to the reduction in model development time not considering the initial development costs of customizing the application. Knowing that the model will need to be created for many different design variable combinations allowed for integrating this approach for the application. Moving from ~4-8 weeks per design iteration to ~ 12 minutes per iteration is considered a significant time savings of the manufacturing lead times. That represents an approximate ~99.94% decrease in modeling times. This alone classifies this research as a tremendous success if it were the only success that was had. In order to develop such a modeling scheme the developer must have a complete understanding of the CAD API environment and software. A medium level programming proficiency is also required since you must utilize the API and develop User Defined Functions (UDFs). Table 11 summarizes the modeling results.

Table 11: Modeling Methods/Time Summary

Modeling Method	Airfoil	Dies
Manual Parametric Modeling	~ 3 weeks	~ 5 weeks
Generative Parametric Modeling	~ 4 minutes	~ 8 minutes

5.3 Results: Analysis

Analysis results are presented in two separate sections since mesh development was executed outside of the DEFORM simulation environment in ANSYS.

5.3.1 ANSYS Results

ANSYS was used to develop the structured 2D meshed sections. A macro was developed using native API functions to automatically import NX geometry, identify section edges using spatial positioning, and create a structured quadrilateral mesh with sufficient element numbers (minimum of 3 elements, Balling 2006) through the thinnest web thickness of the cross-sections by automatically assigned node density functions. Figure 71, Figure 72, and Figure 73 show finished examples of each of the die/blade geometries.

Figure 71: ANSYS Meshed Upper Half of Forming Die

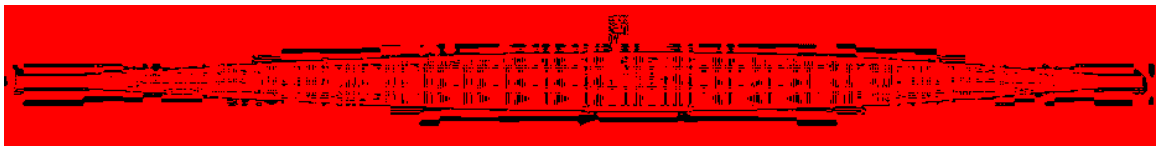


Figure 72: ANSYS Meshed Airfoil Cross-section

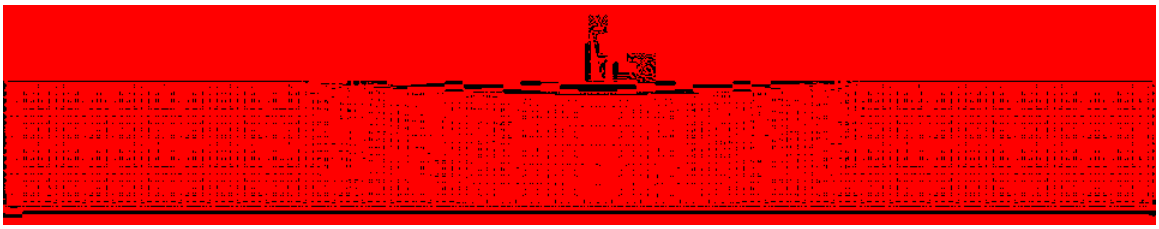


Figure 73: ANSYS Meshed Lower Half of Forming Die

The mesh data for each cross-sectional piece was exported in the required DEFORM input format as a text file. The data exported contained Cartesian node coordinates, number of elements, and all element connectivity's which properly define the geometry in DEFORM. The files, since created from the ANSYS mesh macro, were always repeatable. The ANSYS macro used is located in Appendix B.

Manual mesh development times for a cross-sectional set of data as well as the required automated mesh development times are presented in Table 12.

Table 12: Mesh Methods/Time Summary

Mesh Generation Method	Airfoil	Dies
Manual Cross-section Meshing	~ 4 minutes	~ 4 minutes/die half
Automated Cross-section Meshing	~ 30 seconds	~ 30 seconds/die half

5.3.2 DEFORM Results

The forging simulations used the exact same input deck (preprocessing, contact definitions, material properties, loading conditions, etc) for every simulation with only the meshed geometry input files (Cartesian nodal coordinates and element connectivity's for the

airfoil and die halves) being updated for each simulation run. This allowed the forging simulations to be executed repeatedly with the only meshed geometry changing each run.

Figure 74: Analyzed Model Stations with Datum Planes

The simulation results for the 2D sections (See Figure 74) per iteration were extracted at the last time step in the simulation before minimum squeeze (final minimum distance between the dies) was met. The final time step was extracted with the data of each piece of analyzed geometry (element nodal locations, stresses, and strains) and saved for the evaluation step in the optimization scheme (see Section 5.4). Figure 75 shows a cross-section visual simulation final time step.

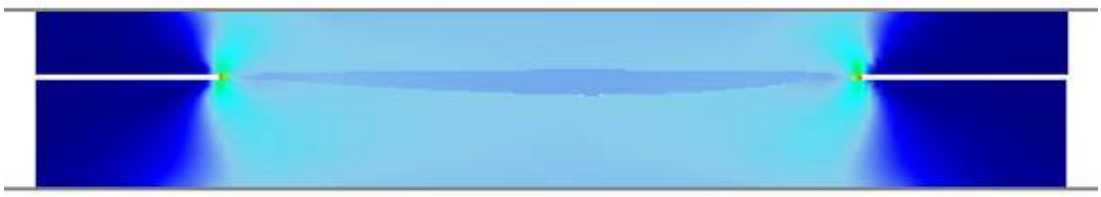


Figure 75: DEFORM Postprocessed Final Simulation Time Step (Effective Stress Plot)

Execution times using DEFORM 2D are presented in Table 13.

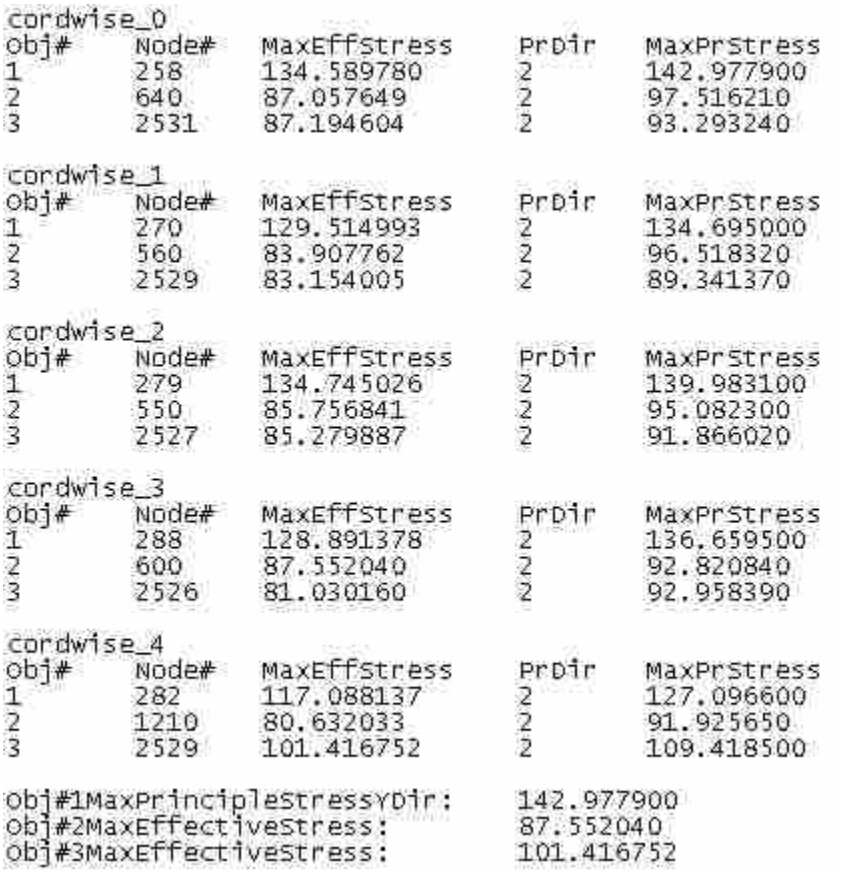
Table 13: 2D Simulation Methods/Time Summary

Forging Simulation Method	Simulation Execution Times
Manual 2D Execution Time	~15 minutes per cross-sectional simulation
Automated 2D Execution time	~5 minutes per cross-sectional simulation

Utilizing the plane-strain assumption and analyzing faster 2D cross-sectional models a 97.5% reduction (~17 hrs vs. ~25 min/5 sections) in simulation execution times was expected and easily demonstrated prior to integration into the methodology. The significant time savings with minimal fidelity being sacrificed was the basis for selecting 2D analysis that proved to allow far more design iterations in less time than a single 3D analysis.

5.4 Results: Evaluation

Evaluation of the simulation results followed the methods described in Section 4.3. Two important data extractions were made. The final resting place (nodal positions) and max principle stress in the X and Y directions for each simulated piece of the cross-sectional cuts were extracted for evaluation (see Figure 76). To interpret Figure 76 a few definitions are needed as shown in Table 14.



```
cordwise_0
Obj#    Node#   MaxEffStress    PrDir   MaxPrStress
1       258     134.589780      2       142.977900
2       640     87.057649       2       97.516210
3       2531    87.194604       2       93.293240

cordwise_1
Obj#    Node#   MaxEffStress    PrDir   MaxPrStress
1       270     129.514993      2       134.695000
2       560     83.907762       2       96.518320
3       2529    83.154005       2       89.341370

cordwise_2
Obj#    Node#   MaxEffStress    PrDir   MaxPrStress
1       279     134.745026      2       139.983100
2       550     85.756841       2       95.082300
3       2527    85.279887       2       91.866020

cordwise_3
Obj#    Node#   MaxEffStress    PrDir   MaxPrStress
1       288     128.891378      2       136.659500
2       600     87.552040       2       92.820840
3       2526    81.030160       2       92.958390

cordwise_4
Obj#    Node#   MaxEffStress    PrDir   MaxPrStress
1       282     117.088137      2       127.096600
2       1210    80.632033       2       91.925650
3       2529    101.416752      2       109.418500

Obj#1MaxPrincipleStressYDir:    142.977900
Obj#2MaxEffectiveStress:        87.552040
Obj#3MaxEffectiveStress:        101.416752
```

Figure 76: Stress Output Example

Table 14: Stress Output Key

cordwise_#	Cross-section number; 0 being closest to root and 4 being closest to tip
obj#	Workpiece or die half identifier; 1 – workpiece, 2 – upper die, 3 – lower die
Node#	The node belonging to the stress location
MaxEffStress	The maximum effective stress (Von Mises)
MaxPrStress	Maximum principle stress in the respective objects
PrDir	Direction of maximum principle stress; 1 – X direction, 2 – Y direction

The workpiece surface nodal locations were used to determine the distance between the nominal surface profile and the simulated. As described in Section 4.3 the deviations were calculated using the resultant blade surface profile at each cross-section and a global fitness value for the five cross-sections surface deviations calculated. This value became the objective function value for the design iteration. Figure 77 shows a portion of a text based output file that contains the results for the deviations. There are two types of deviations that were calculated and used for comparison and validation that the deviations were in the "same ballpark" with each other. All simulations produced deviations from both calculation methods that were consistently representative of the data. The object intersection ("objInterDevObjective") deviation was the most accurate indicator of the deviations (Modeled after movement of die in a linear fashion) and was sent to iSIGHT as the objective value.

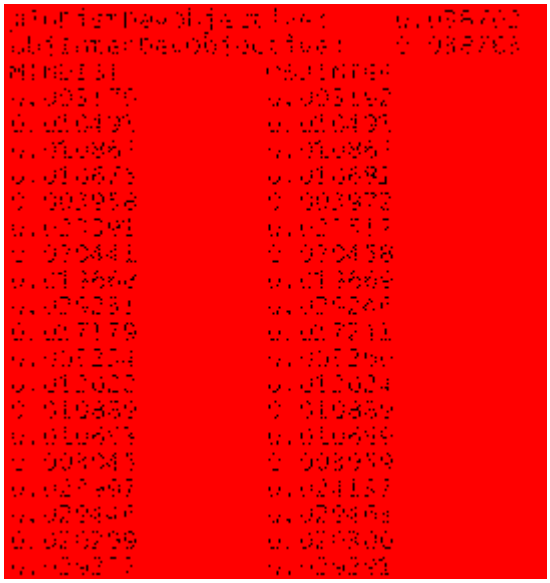


Figure 77: Deviations Output Example

With the principle stresses (σ_x, σ_y) and shear stress in the XY plane (τ_{xy}), the max effective stress σ_{max}, otherwise known as the von mises stress, was calculated for both sides of

the dies and the workpiece. This information was saved and compared against the materials respective yield stress conditions (See Table 7) to ensure the die and workpiece material had not failed during forging simulation. This data was correlated to the optimization design constraints and a condition of success or failure flagged for the design iteration. During all the simulations the yield stress conditions were never violated for the workpiece and dies.

5.5 Results: Optimization

Optimization was executed as described in Section 4.4 using iSIGHT-FD as the integration software. All modules (modeling, meshing, simulation, data extraction, and deviations analysis) along with all the data exchanges we created successfully. Appendix E shows the complete optimization architecture employed. The material property constraints were integrated and maintained throughout all iterations. The objective function was easily imported into the iSIGHT optimization engine as described in Section 5.4. Figure 78 shows an example of the history plots for the design variables. They show how convergence to a solution was achieved for a specific starting point for the 3 design variables.

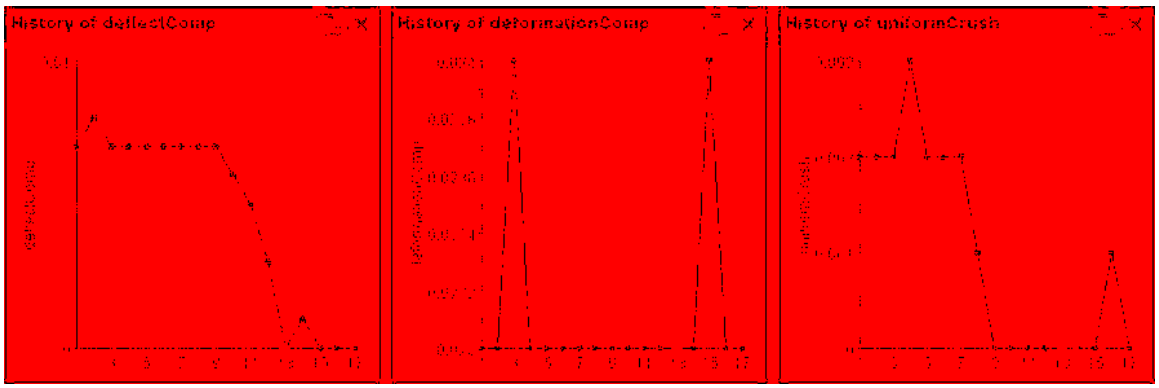
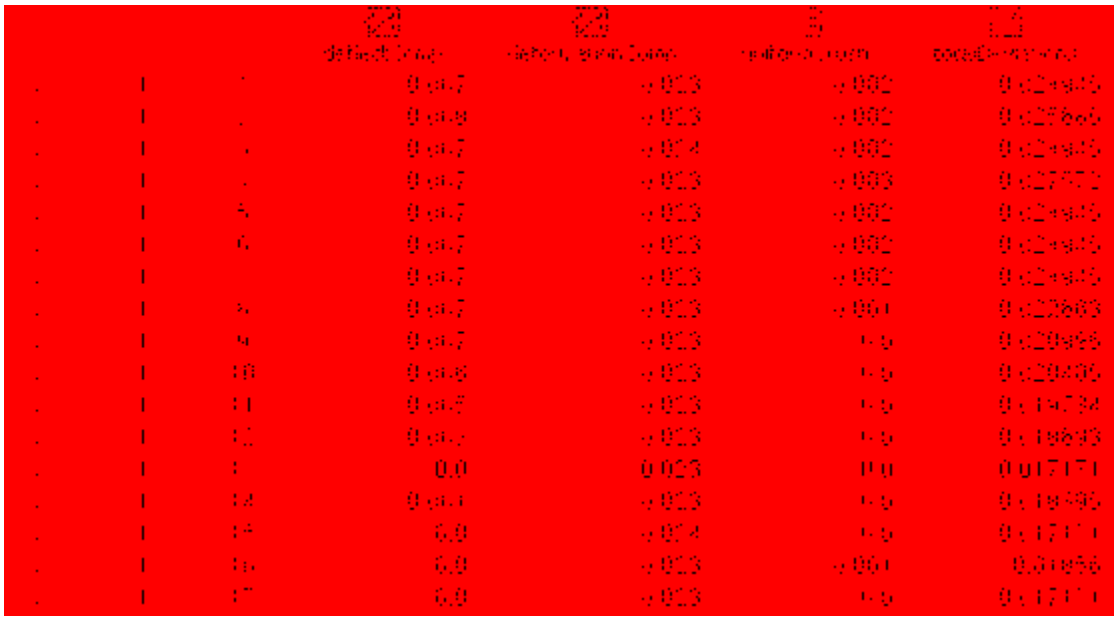


Figure 78: Optimization History Plot Example

Table 15: Optimization Data History Example



Initial trials showed convergence to a solution occurred routinely although the solution varied depending on the initial design variable inputs. A trial and error method was employed to obtain a general feel for the complexity of the design space and to determine approximate run times for converging to a solution. It became apparent that a more structured approach to determining the optimal design variable values for minimum surface deviations was needed. That observation led to the use of a Design of Experiment (DOE) that facilitated effective characterization of the model response using Response Surface Modeling (RSM) methods for predicting best possible variable combinations.

5.6 Results: Design of Experiments

Using the experimental design described in Section 4.5, the inputs were fed into the optimization routine and the minimum objective function results obtained once convergence occurred. Table 16 shows the results obtained.

Table 16: Test Case 1 and 2 Optimization Results

			CCI		
	X1	X2	X3	Y_{opt1}	Y_{opt2}
Run	Defl Comp	Defor Comp	Uni	Test Case 1	Test Case 2
1	0.007	0.007	0.002	0.017171	0.016189
2	0.023	0.007	0.002	0.02283	0.02727
3	0.007	0.023	0.002	0.017171	0.016189
4	0.023	0.023	0.002	0.02283	0.022434
5	0.007	0.007	0.008	0.037224	0.045019
6	0.023	0.007	0.008	0.023058	0.035634
7	0.007	0.023	0.008	0.037244	0.045019
8	0.023	0.023	0.008	0.023058	0.016189
9	0	0.015	0.005	0.030452	0.016189
10	0.03	0.015	0.005	0.024827	0.035964
11	0.015	0	0.005	0.02283	0.025287
12	0.015	0.03	0.005	0.023172	0.025287
13	0.015	0.015	0	0.024288	0.025287
14	0.015	0.015	0.01	0.042926	0.016189
15	0.015	0.015	0.005	0.023172	0.025287
16	0.015	0.015	0.005	0.023172	0.025287
17	0.015	0.015	0.005	0.023172	0.025287
18	0.015	0.015	0.005	0.023172	0.025287
19	0.015	0.015	0.005	0.023172	0.025287
20	0.015	0.015	0.005	0.023172	0.025287

Prior to any statistical analysis being done with the experimental data it was noted that the center point results for each test case produced no variation (curvature) (See Table 16). That result can be attributed to the repeatability of computer simulations producing the same result which further verified model stability.

Liner regression analysis was used to obtain a predictive quadratic equation for the models (Test cases 1 and 2). JMP statistical analysis software was used for the analysis. An attribute variable was assigned to each test case which allowed for both data sets to be analyzed simultaneously and viewed on-demand in the regression results. The JMP input deck used to obtain the linear regression analysis results is found in Appendix F.

The regression results show that a single model could represent both test cases. The attribute variable used as the switch between both test cases proved to be insignificant which demonstrated that a single model could represent both test cases. Figure 79 shows the regression results (significant factors and factor combinations).

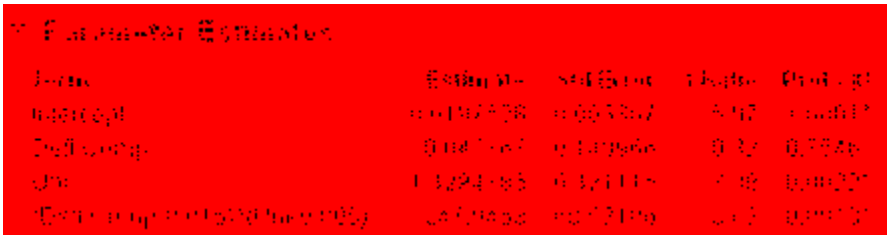


Figure 79: Regression Results

Deflection Compensation and Uniform Crush along with their interaction proved to be the significant factors and effects of the analysis. The resultant quadratic equation from the analysis is shown below in Equation 12.

$$Y_{opt} = 0.0197 - 0.0476*(DeflComp) + 1.4294*(Uni) - 247.9453*(DeflComp*Uni) \quad (12)$$

From Equation 12, a 3D contour plot of the design space was generated (See Figure 80). From these graphs it is clear that a saddle situation exists inside of the design space. The minimum amount of surface deviations (Yopt) occurs when deflection composition and uniform crush are both at their minimum values of zero. The predicted value, when both significant variables are at their minimum, should fall in the range of 0.01157+/-0.012562 inches.

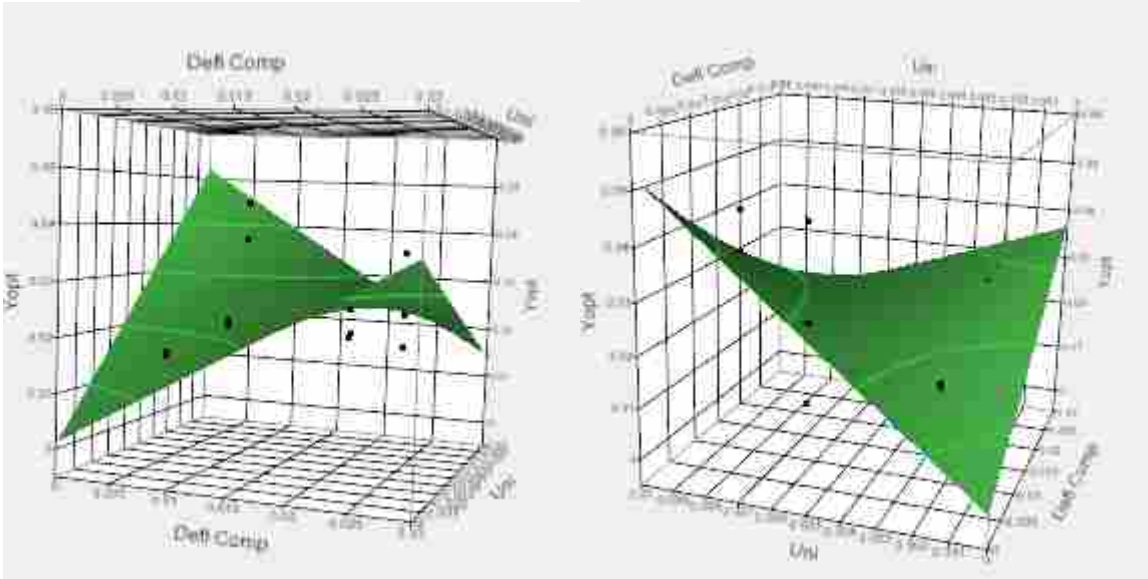


Figure 80: Response Surface Model

To further show the identified optimum a Prediction Profiler was used in JMP to visualize the predicted output given various significant variable inputs. Figure 81 to Figure 83 show different predictive outputs for Deflection Comp and Uniform Crush settings. The vertical red dotted line represents the variable values and the dotted blue lines represent the predicted variability.

It is important to note that the variability band narrows around the deflection comp and uniform crush values of 0.015 and 0.005 respectively. This is attributed to the experimental central composite design chosen (See Figure 62). Increasing the number of experimental data points obtained inside of the inscribed sphere of the design space (while maintaining the properties of orthogonality, rotatability and uniform precision) it is believed that a smaller error on the predicted optimum would have been obtained. This is in part due to an increased number of experimental points inside of the inscribed sphere of the design space effectively improving the error bounds attained for the predictions.

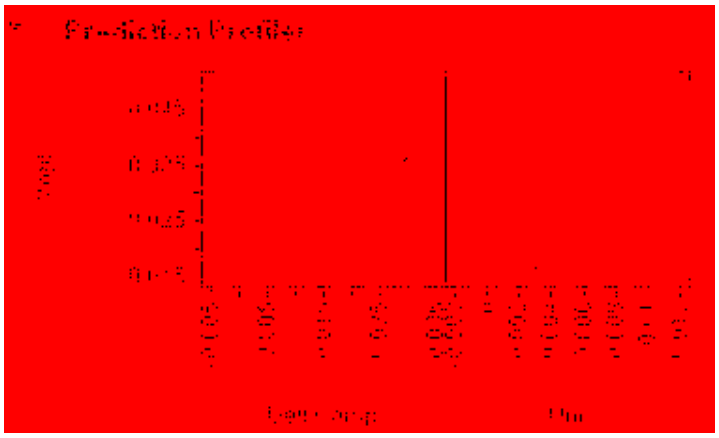


Figure 81: Prediction Profiler Lower Range

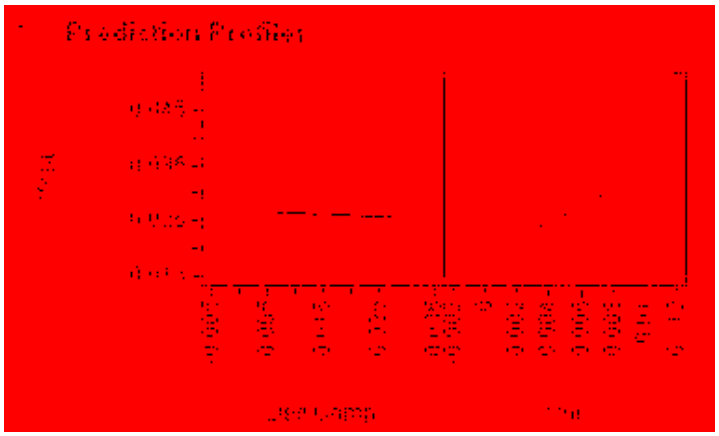


Figure 82: Prediction Profiler Mid Range

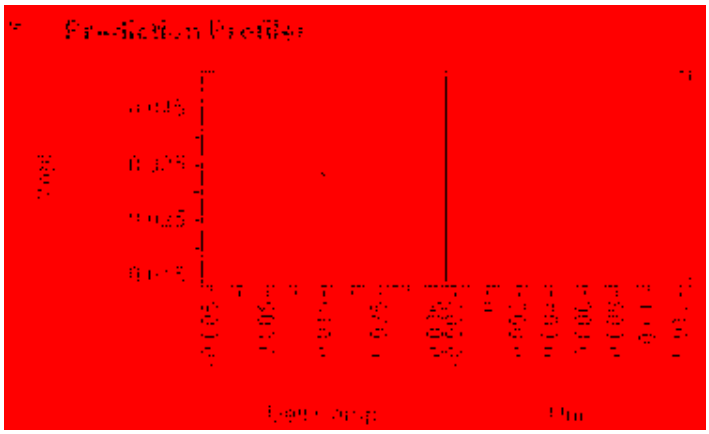


Figure 83: Prediction Profiler Upper Range

Since deformation compensation was considered insignificant it was irrelevant as to what its starting value should be. Using the mid range value for deformation compensation and the predicted settings for uniform crush and deflection compensation the predicted optimum for test case 1 and 2 were executed (see Table 17).

Table 17: Predicted Optimal Design Variable Settings

X1	X2	X3
Deflection Compensation	Deformation Compensation	Uniform Crush
0	0.015	0

5.6.1 Results: Case Study 1- High Fidelity Model (Double Sided Machined Blade)

Using the optimization inputs from Table 17, the double sided blade input produced a predicted minimum deviation of 0.017171 inches at the design variable settings shown in Table 17. Figure 84 and Table 18 shows the optimization history plots and history data for the optimum.

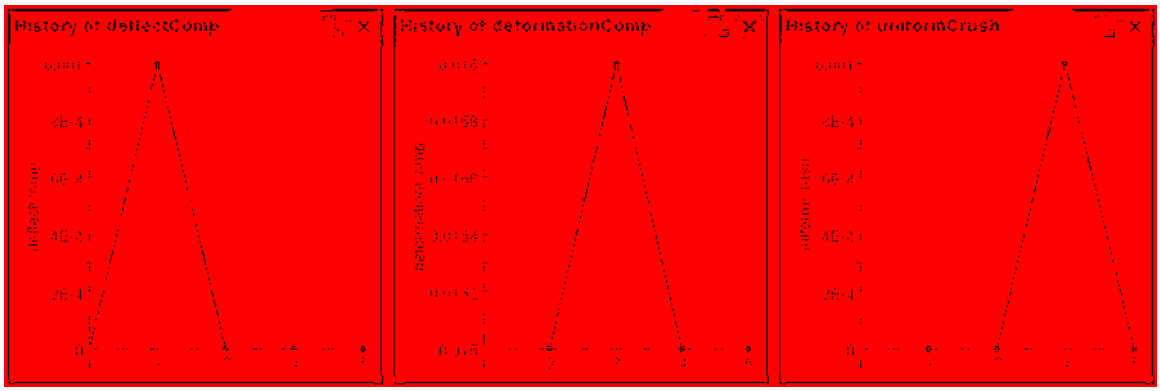
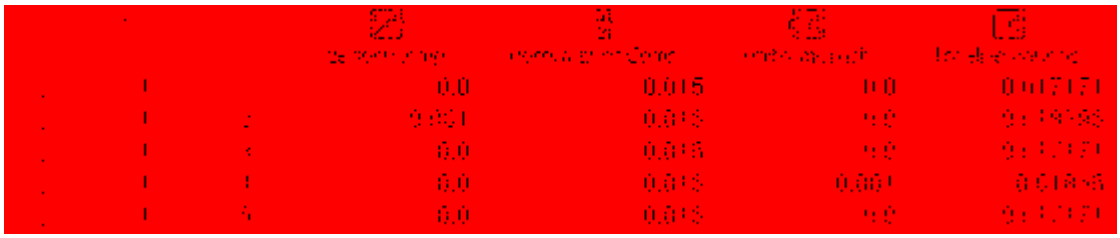


Figure 84: Test Case 1 Optimized History Plots

Table 18: Test Case 1 Optimum Data History



This value falls inside of the RSM predicted minimum deviation of 0.01157 +/– 0.012562 (the range is 0.024132 to -0.000992 or otherwise '0'). Given the variable (factor) ranges, the results for test case 1 can be considered accurate and only improved by using more design points for the DOE/RSM activity.

5.6.2 Results: Case Study 2- High Fidelity Model (Single Sided Machined Blade)

Using the same optimization inputs from Table 17 as was used in the previous section for the double sided airfoil, the single sided blade input produced a predicted minimum deviation of 0.016189 inches at the design variable settings shown in Table 17. Figure 85 and Table 19 shows the optimization history plots and history data for the optimum.

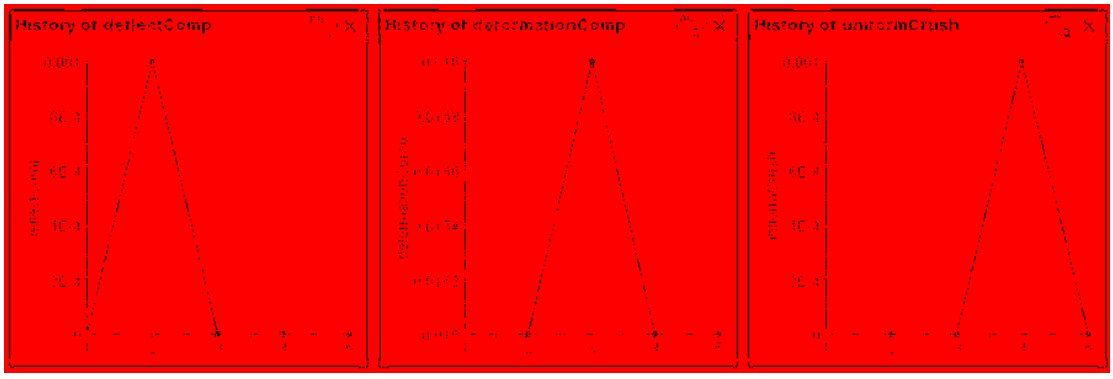
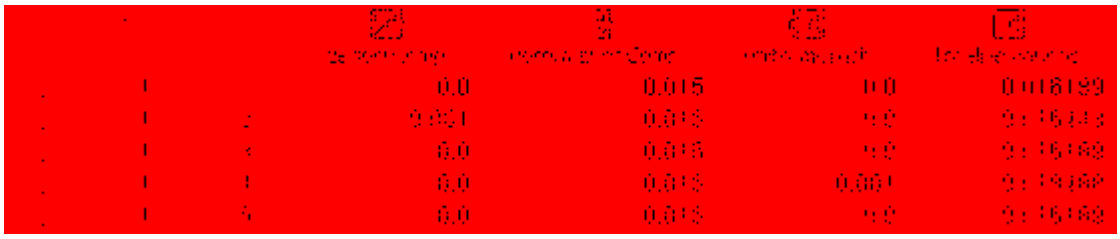


Figure 85: Test Case 2 Optimized History Plots

Table 19: Test Case 2 Optimum Data History



This value falls inside of the RSM predicted minimum deviation of 0.01157 +/– 0.012562 (the range is 0.024132 to -0.000992 or otherwise ‘0’). Given the variable (factor) ranges, the results for test case 2 can be considered accurate and only improved by using more design points for the DOE/RSM activity.

CHAPTER 6: CONCLUSIONS

6.1 Conclusions

The objective of this research was to develop an automated tooling design scheme that integrates commercial CAD/CAM/CAE technologies. In order to demonstrate this objective, a methodology was developed that automated creation of complex free form surface models, component meshing, simulation, and surface evaluation all coupled with an optimization engine. In Chapter 4, the application for complex free-form surfaces was developed for a jet engine shroudless hollow fan blade. The implementation of a generative parametric CAD paradigm applied to complex surface models integrated with the various CAx tools accomplished the objectives described in Section 1.1. The objectives are repeated here for reference.

1. Can the integration of parametric CAD, meshing capabilities, bulk forming simulation, and optimization accomplish a realistic part/die models prediction when the parts being formed are defined using complex free-form surfaces?

   The technologies mentioned in objective 1 were successfully integrated into a master program that was capable of driving to an optimal solution for the test cases presented. The generative parametric models were created to the desired fidelity. The objective function of the optimization showed that the total surface deviation calculations could be reduced by identifying the correct design variable combinations.

Using RSM, experiments were run that allowed for the creation of a response surface and identification of the optimal design variable settings to achieve the minimal amount of surface deviations.

2. Can numerical surface construction and interpolation techniques be used to accurately allow for an objective rating on the accuracy of predictive results compared to the original design intent?

   Yes, surface creation through the design intent allowed for an interpolative method of deviation calculation to be determined. It worked quite well using both surface normal deviations off of specified points on the nominal surface and also a minimum distance technique from the nominal surface location and the derived surface. Although both methods were calculated the deviations used for this research were the distance from the nominal surface in the direction of die movement. The surface to surface deviations were captured in the objective function and easily fed into the optimization engine for iteration to iteration evaluation.

3. Can the implementation of this methodology (the integration and automation of associative tooling surfaces automatically derived from the original part geometry) produce a significant reduction in design/engineering tool development lead time?

   It was demonstrated by engineers at Pratt & Whitney in charge of developing workpiece and die models for the complex free form die surfaces that approximately 2 man months (~320 hrs) were needed to model the free-form geometry (non-parametric). This research demonstrates that full 3D models of both the airfoil workpiece and the forming die geometry could be modeled and sectioned in ~10 to 12 minutes. The generative parametric modeling was easily repeatable for varying

design inputs. Using the optimization scheme and the need to build new models for each new design variable input there was a result of ~600x speed up in modeling capability.

4. Can this methodology be used as a blue print for any automotive or aerospace tooling industry to eliminate significant time and costs from the manufacture/design of complex free-form components?

   Yes. It was demonstrated that for complex 1st stage compressor blades for a jet engine's shroudless hollow fan blade that significant time and cost savings for high fidelity models can be achieved. This methodology could be easily adapted for other complex free form surface applications such as complex automotive body panels, ship hulls, consumer goods, machinery, bio-mechanical systems, etc. It is important to note that development of generative parametric models of the magnitude demonstrated integrated with simulation/analysis would only be justified when the desired model would be needed for more than a single design scenario. Performing a cost verse benefit analysis to determine if the methodology would be an overall cost savings would be required.

The generative parametric modeling scheme coupled with the automated analysis for optimization were evaluated to show the effectiveness of the design tool in overcoming the obstacles commonly found when analyzing such large models. Chapter 5 revealed results of the automated optimization scheme and test cases conducted as well as discussed the advantages/time savings of using a generative parametric design tool. The methodology

allowed identification of the design parameter needed to obtain tooling and pre-formed work piece geometries optimized to produce near-to-design intent products.

The results from the test cases (Test Case 1: 0.017171 inches and Test Case 2: 0.016189 inches) would be acceptable given that they were within 20 mils of the nominal surface for both test cases and are representative of in process forging geometry. It is interesting to note that the experimental machining method (single sided machining – Test Case 2) for the airfoils investigated would produce smaller deviation errors per the simulation. This discovery will be passed on to Pratt & Whitney for investigation.

From the results and accomplishments of this conceptual methodology, automation of design and integration of the various CAx applications was successful at identifying the best possible design variable combinations within the given modeling constraints. The results show successful implementation on complex free-form surfaces that were previously unable to be developed in any efficient parametric way.

It is recommended by the author that tool design groups spending too much time with design iterations should invest in robust state-of-the-art design tools that automate time intensive processes and eliminate problematic characteristics of such complex models into simply executed programs. Manual (interactive) die and workpiece construction along with trial and error methods to determine the proper surface contours is extremely cumbersome. The application of generative programmatic parametrics on complex free form surfaces yields a high rate of return and delivers, in the end, a faster tool development cycle time

6.2 Future Work

Further research and a logical next step in the development of this methodology would be to implement it on true production level airfoil definitions and test if simulation output produces similar results.

The models proved out in this thesis were not capable of being physically verified. The cost to develop a die capable of forming such complex surfaces costs on the order of $1,000,000 with the workpiece cost at $50,000. If a 3D forming simulation tool had been available during the development and testing of this research, further verification could have been achieved by comparing the 2D vs. 3D results to have better estimated the modeling/simulation error.

In depth DOE's could be used to determine the appropriate design variable ranges. The variable ranges were estimated based on actual experience from the chief design engineer for the Hollow Fan Blade group at Pratt & Whitney. Performing statistical screening studies would have allowed the most pertinent and feasible variable and associated values to be identified (improved model fidelity).

Another recommendation that could be key furthering implementation of this research would be to integrate the DOE into the iSIGHT workflow. Connecting the two major activities can be easily accomplished and would have eliminated the need to manually load optimization starting points into the system. It is also safe to point out that results would be automatically cataloged. One major advantage to integrating optimization and DOE would be to eliminate the need to manually set up the DOE in JMP. A seamless autonomous optimization could offer even further time savings in determining the optimal design conditions.

Development of the iSIGHT workflow required a high level of user interaction/preparation. Had a scripting language (such as PERL) been used to execute the various modules a more modular optimization system could have been created. iSIGHT with integration of a script based programming language would have required less user interaction and it is strongly recommended that iSIGHT only be used to execute standalone programs that encompass all the required steps for optimization execution.

Although the research was classified as a success, higher fidelity results could have been achieved by slightly modifying the chosen central composite design. This would have been accomplished by using an extended central composite design for computer simulation experiments. That entails analyzing more experimental points inside of the feasible design space. Result there from would reveal a higher fidelity RSM that would provide a more detailed estimation of the error for the model prediction.

REFERENCES

Allanda, V., Anand, S. (1995), "Feature-based modeling approaches for integrated manufacturing: state-of-the-art survey and future research directions," *International Journal of Computer Integrated Manufacturing*, Vol. 8, Issue 4, pp. 411-440

Anderl, R. Mendgen R., (1995), "Parametric Design and its Impact on Solid Modeling Applications," *Proceedings of the 3rd Symposium on Solid Modeling Applications*, Salt Lake City, Utah December 1995, pp.1-12

Ardalan, S. (2000) "DrawCraft: A Spacecraft Design Tool for Integrated Concurrent Engineering," *Aerospace Conference Proceedings*, IEEE, Vol. 11, pp. 501-510

Bates, P. D., Stewart, M. D., Desitter, A., Anderson, M. G., Renaud, J. P., and Smith, J. A. (2000). "Numerical simulation of floodplain hydrology." *Water Resources Research*, 36(9), 2517-2529.

Bedient, P. B. and Huber, W. C. (1988). *Hydrology and Floodplain Analysis*, Addison-Wesley.

Balling, R.J., (2006), *Continuum Mechanics and Finite Element Analysis*, Brigham Young University Academic Publishing, Provo, UT, ISBN# 0-7003-8553-3.

Biles, W. E., 1984, "Design of Simulation Experiments," *1984 Winter Simulation Conference (WSC) (Sheppard, S., Pooch, U.,et al., eds.)*, Dallas, TX, IEEE, pp. 99-104.

Box, G. E. P. and Draper, N. R., 1987, *Empirical Model-Building and Response Surfaces*, John Wiley & Sons, New York.

Choi, B.K., (1991), *Surface Modeling for CAD-Cam*, Elsevier Science Inc., New York, NY

Delap, D. C. (2003), CAD-*Based Creation and Optimization of a Gas Turbine Flowpath Module with Multiple Parameterizations*, M.S. Thesis, Brigham Young University

Dieter, G.E., (2000), *Engineering Design: A Materials and Processing Approach*, McGraw Hill, 3rd Ed.

Elliott, J. H. (2004), *An Automated Approach to Feature-Based Design for Reusable Parameter-Rich Surface Models*, M.S. Thesis, Brigham Young University

Farin, G., (1988), *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, San Diego, CA.

Haimes, B., (2003), "Unified Geometry Access for Analysis and Design", *Proceedings of the 12th International Meshing Roundtable*, Sandia National Laboratories, pp.21-31

Hardee, E., Chang, K., Tu, J., Choi, K., Grindeanu, I., and Yu, X., (1999), "A CAD-based Design Parameterization for Shape Optimization of Elastic Solids," *Advances in Engineering Software*, Vol. 30, pp. 185-199.

Giunta, Anthony, A., (1997), *Aircraft Multidisciplinary Design Optimization Using Design of Experiments Theory and Response Surface Modeling Methods*, Doctorate of Philosophy in Aerospace Engineering, Virginia Polytechnic Institute

Beichang, He, Roehl, Peter J., Irani, Rohinton K., (1998), "CAD and CAE integration with application to the forging shape optimization of turbine disks", AIAA Paper 98-2032, *7th AIAA/USAF/NASA/ISSMO Symposium on Multi-disciplinary Analysis and Optimization*, St. Louis, MO, September 1998.

Hoffman, C., Kim, K., (2001), "Towards Valid Parametric CAD Models," *Computer-Aided Design*, Vol. 33, 2001, pp. 81-90

Hoffman, C., (2005), "Constraint-Based Computer-Aided Design," Journal of Computing and Information Science in Engineering, Vol. 5, Issue 3, pp.182-187

Hogge, D., (2002), *Integrating Commercial CAx Software to Perform Multidisciplinary Design Optimization*, M.S. Thesis, Brigham Young University.

Hosaka, M., (1992), Modeling of Curves and Surfaces in CAD/CAM, Springer-Verlag New York Inc., New York, New York

Hsu, T., Sinha, D. (1992), *Computer-Aided Design: An Integrated Approach*, West Publishing, St. Paul, MN.

King, M. (2004), *A CAD-Centric Approach to CFD Analysis with Discrete Features*, M.S. Thesis, Brigham Young University

Lawson, J., Erjavec, J., (2001), *Modern Statistic for Engineering and Quality Improvement*, Duxbury, Pacific Grove, CA.

Lee, K., (1999), *Principles of CAD/CAM/CAE Systems*, Addison Wesley, Reading, MA.

Liker, J.K., Fleischer, M., Arnsdorf, D., (1992), *Fulfilling the Promise of CAD*, Sloan Management Review, pp.74-86

Magalhaes, W. (2005), *Parametric Void Insertion for CAD-Centric Topological Optimization*, M.S. Thesis, Brigham Young University

Mäntylä, M., (1988), *An Introduction to Solid Modeling*, Computer Science Press, Rockville, Maryland

Mortenson, M.E.,(1985), *Geometric modeling*, John Wiley & Sons, Inc. New York, NY

Ou, H., Armstrong, C.G., (2002), " Die shape compensation in hot forging of titanium aerofoil sections," *Journal of Materials Processing Technology*, Vol. 125-126, February 2002, pp. 347-352.

Ou, H., Armstrong, C.G., Price, M.A., (2003), Die shape optimization in forging of aerofoil sections," *Journal of Materials Processing Technology*, Vol. 132, pp. 21-27.

Parkinson, A., (2006), *Optimization-Based Design*, Brigham Young University Academic Publishing, Provo, UT., ISBN# 0-7003-7848-0

Piegel, L., Tiller, W. (1997), *The NURBS Book*, Springer Verlag, Berlin-Heidelberg-New York, ISBN# 3-540-61545-8

Ramaswamy, R., Ulrich, K., Kishi, N., Tomikashi, M. (1993), "Solving Parametric Design Problems Requiring Configuration Choices," *ASME Journal of Mechanical Design*, Vol. 115, pp. 20-28.

Raphael, B., Smith, I.F.C., (2003), *Fundamentals of Computer Aided Engineering*, John Wiley, ISBN# 978-0-471-48715-9

Requicha, A. G. (1980), "Representation for Rigid Solids: Theory, Methods and Systems," *ACM Computing Surveys*, Vol. 12 No. 4, pp. 427-464.

Requicha, A.A.G., Voelcker, H.B.,(1982) "Solid Modeling: A Historical Summary and Contemporary Assessment,"*IEEE Computer Graphics and Applications*, Vol. 2, No. 2, March 1982, pp. 9-24.

Rohm, T., Tucker, S. S., Jones, C. L., Jensen, C. G. (2000), "Parametric Engineering Design Tools and Applications," *ASME Design Automation Conference*, Baltimore, Maryland, September 10-13, 2000.

Rohm III, T. (2001), *Graphical Creation of CAD Parametric Application Programs*, M.S. Thesis, Brigham Young University.

Rossignac, J.R., Turner, J. (1991), Proc. Symp. Solid Modeling Foundations and CAD/CAM Applications, ACM Press, New York

Samareh, J. A., "Survey of Shape Parameterization Techniques for High-Fidelity Multidisciplinary Shape Optimization," *AIAA Journal*, Vol 39, No. 5, 2001, pp.877-884

Sederberg, T., (2007), *Computer-Aided Geometric Design*, http://cagd.cs.byu.edu/~557/text/

Shah, J.J., Mantyla, M, (1995), Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications, Wiley-Interscience, ISBN# 978-0-471-00214-7

Su, B., Liu, D., (1989), Computational geometry: curve and surface modeling, Academic Press Professional, Inc., San Diego, California,

Simpson, T., Korte, J., Mauery, T., Mistree, F., (1998), "Comparison of Response Surface and Kriging Models for Multidisciplinary Design Optimization," AIAA-98-4755

Tang, P.S., Chang, K.H., (2001), "Integration of Topology and Shape Optimization for Design of Structural Components," Structural and Multidisciplinary Optimization, Vol. 22 No. 1, pp.65-82

Vanderplaats, G., (1999), "Structural design optimization status and direction," *Journal of Aircraft*, Vol. 36, No. 1, January - February 1999, pp. 11-19.

Waterbury, S.C., (1999), "STEP for Multi-Disciplinary Model Management: 'Intelligent PDM'," *ESA/NASA Video/Teleconference STEP for Aerospace applications*, July 16, 1999

Wilson, M., (2004), *Integration of Rapid Prototyping Preprocessing Operations with a Commercial CAD System*, M.S. Thesis, Brigham Young University

Wu, Wei-Tsu, Tang, JuiPeng, (2004), "Manufacturing Process and Process Modeling," Scientific Forming Technologies Corporation (DEFORM), SFTC Paper # 394

Zeid, Ibrahim, (2005), Mastering CAD/CAM, McGraw-Hill, NY, NY

APPENDIX A. SURFACE DEVIATION ALGORITHM

```
/*==========================================================================
=========================
Program           : SurfaceDeviations.cpp
Project           : Thesis Project
Purpose     : Calculates the deviations between simulated data and design
intent

Revision History  June 1, 2007                  Started
                        KGF
==========================================================================
=========================*/
#include "master_include.hpp"
#include "kevin_master_include.hpp"
#include "objectData.hpp"

int SurfaceDeviations(char* path)
{
      tag_t ps_surf = cycle_by_name("ps_surface");unblank(ps_surf);
      tag_t ss_surf = cycle_by_name("ss_surface");unblank(ss_surf);

      char val[256],val1[256];
      int numCordSecs=0,numSpanSecs=0;
      sprintf(val,"num_cordwise_sections");get_input_data1(path,val,numCordSe
cs);
      sprintf(val,"num_spanwise_sections");get_input_data1(path,val,numSpanSe
cs);
      char* filename = "C:\\Research\\PartFiles\\intersectLocationsPS.txt";
      char* filename1 = "C:\\Research\\PartFiles\\intersectLocationsSS.txt";

      FILE* psPntsFile = fopen(filename,"r");
      FILE* ssPntsFile = fopen(filename1,"r");
      int i=0,j=0,k=0;
      //int numPnts = numCordSecs*numSpanSecs;
      double projectDir[3] = {0,1,0};
      double*** ps_pnts = new double**[numSpanSecs];
      double*** ss_pnts = new double**[numSpanSecs];
      char junk[256],junk1[256],junk2[256],junk3[256];

      for(i=0;i<numSpanSecs;i++)
      {
            ps_pnts[i] = new double*[numCordSecs];
            ss_pnts[i] = new double*[numCordSecs];
            for(j=0;j<numCordSecs;j++)
            {
```

```
                  ps_pnts[i][j] = new double[3];ss_pnts[i][j] = new
double[3];
                  fgets(val,100,psPntsFile);fgets(val1,100,ssPntsFile);
                  //gets the intersection points of the cord and span
sections cuts from model
                  sscanf(val,"%s    %s
      %s\n",junk,junk1,junk2);ps_pnts[i][j][0] = atof(junk);ps_pnts[i][j][1]
= atof(junk1);ps_pnts[i][j][2] = atof(junk2);
      //create_dumb_pnt2(2,ps_pnts[i]);
                  sscanf(val1,"%s   %s
      %s\n",junk,junk1,junk2);ss_pnts[i][j][0] = atof(junk);ss_pnts[i][j][1]
= atof(junk1);ss_pnts[i][j][2] = atof(junk2);
      //create_dumb_pnt2(3,ss_pnts[i]);


      pnt_obj_project(ps_pnts[i][j],ps_surf,projectDir,100,ps_pnts[i][j]);
      create_dumb_pnt2(186,ps_pnts[i][j]);

      pnt_obj_project(ss_pnts[i][j],ss_surf,projectDir,100,ss_pnts[i][j]);
      create_dumb_pnt2(216,ss_pnts[i][j]);
            }
      }
      fclose(psPntsFile);
      fclose(ssPntsFile);

      //bring in simulated die and workpiece geometry
      double** pntData;
      char side[256],prefix[256],outputFilename[256];
      int count=0,ObjNum=0,NumDataPnts=0,NodeNum=0;
      double cut_value=0.0;
      double XCoord=0.0,YCoord=0.0,ZCoord=0.0;
      double origin[3] = {0,0,0};
      double vec_dir[3] = {0,-1,0};

      vector<ObjectData*> ObjectVector;
      ObjectData* tempObj;

      //0,1 because the cuts were in the cordwise and spanwise directions
      //for(i=0;i<2;i++)            //0 for PS slices and 1 for SS slices
      for(i=0;i<1;i++)        //For use when only using the cordwise cut
direction for plain strain assumption
      {
            if(i==0){strncpy(side,"cordwise",9);strncpy(prefix,"cord",5);}
            else{strncpy(side,"spanwise",9);strncpy(prefix,"span",5);}

            for(int j=0;j<numCordSecs;j++)      //5 becasue of the 5 section
cuts(#'s 0-4) in each direction
            //for(j=0;j<1;j++)
            {

      sprintf(val,"%s_section_%d",prefix,j);get_input_data2(path,val,cut_valu
e);               //getting the value ('z if cord' and y if span') for where
cuts happened

                  //sprintf(outputFilename,"..\\Simulation\\I-
O\\OUTPUT_%s_%d.KEY",side,j);//printf("%s\n",filename);           //Used with
.exe
```

```
                  sprintf(outputFilename,"C:\\Research\\Simulation\\I-
O\\OUTPUT_%s_%d.KEY",side,j);                  //Used with .dll

                  FILE *outputFile = fopen(outputFilename,"r");
                  if(outputFile==NULL)
                  {
                        printf("Output file '%s' to be read is not
correct\n",filename);
                        return 5;
                  }

                  //Iterate 3 times since 3 objects with peripheral geometry
                  for(int k=1;k<4;k++)
                  {
                        fgets(val,100,outputFile);
                        while(strncmp(val,"DIEGEO",6)!=0 && count<1000000)
                        {
                              fgets(val,100,outputFile);count++;
                        }

                        sscanf(val,"DIEGEO %s %s %s",junk,junk1,junk2);
      printf("%s %s %s\n",junk,junk1,junk2);
                        ObjNum = atoi(junk);NumDataPnts = atoi(junk2);
      printf("ObjectNum: %d\tNumDataPnts: %d\n",ObjNum,NumDataPnts);

                        //system("PAUSE");

                        pntData = new double*[NumDataPnts];
                        for(int m=0;m<NumDataPnts;m++)
                        {
                              pntData[m] = new double[3];
                              fscanf(outputFile,"%s %s %s
%s\n",junk3,junk,junk1,junk2);//printf("%d  %1.4f  %1.4f
%1.4f\n",atoi(junk3),atof(junk),atof(junk1),atof(junk2));
                              pntData[m][0] = atof(junk);pntData[m][1] =
atof(junk1);
                              if(i==0)
                              {
                                    pntData[m][2] = cut_value;    //shift the
z coord back to original height
                              }
                              else
                              {
                                    pntData[m][2] = atof(junk2);
                              }

                              //rotate the points for spanwise sections back
into position
                              // Only used when evaluating both directions.
Not used when only using cordwise cuts
                              if(i==1)
                              {
                                    tag_t point =
create_dumb_pnt(pntData[m]);
                                    rotate_object1(point,origin,vec_dir,-90);
      //Rotate datapnt objects back into YZ plane from XY by -90 deg about y
axis
```

```
                                    ask_point_data(point,pntData[m]);
                                    pntData[m][0] = cut_value;del(point);
                              }
                        }
                        //tag_t testCurve =
create_dumb_bcurve2(NumDataPnts,3,1,pntData);//unblank(testCurve);

                        //Load information into the class for the vector of
class objects
                        if(i==0)
                        {
                              //Cordwise Cuts go here
                              tempObj = new ObjectData();
                              tempObj->cordOrSpan = i;
            //CordwiseCut or Spanwise Cut direction
                              tempObj->cutNum = j;
            //Cut number in the cord or spanwise direction
                              tempObj->objectNum = k;
            //Associated deform object ID
                              tempObj->numPnts = NumDataPnts;
            //Number of points associated with the deform geometry
                              tempObj->GetPoints(pntData,NumDataPnts);
      //Actual points of the die/workpiece geometry
                              ObjectVector.push_back(tempObj);
                        }
                        else
                        {
                              //Spanwise Cuts go here
                              tempObj = new ObjectData();
                              tempObj->cordOrSpan = i;
                              tempObj->cutNum = j;
                              tempObj->objectNum = k;
                              tempObj->numPnts = NumDataPnts;
                              tempObj->GetPoints(pntData,NumDataPnts);
                              ObjectVector.push_back(tempObj);
                        }
                        count=0;
                  }
                  fclose(outputFile);
            }
      }
      printf("Vector Size: %d\n",(int)ObjectVector.size());

      double y_dir[3] = {0,1,0},tempPnt[3];
      double approxDist = .005;
      double dev= 0.0,dev1 = 0.0;
      int counter = 0;
    std::vector<double> CordDeviation,CordDeviation1;   //Corddev is for the
mindist; CordDev1 is the objprojection dev's
      std::vector<double> SpanDeviation,SpanDeviation1;
      std::vector<double> Deviations,Deviations1;

      //Calculate deviations     # dev's = 2*numPnts  or
(int)ObjectVector.size()
      for(i=0;i<(int)ObjectVector.size();i++)
      {
```

```
            tag_t curve = create_dumb_bcurve2(ObjectVector[i]-
>numPnts,3,1,ObjectVector[i]->dataPnts);
            set_name2(curve, "curve",i);
            pr("\n");
            if(ObjectVector[i]->cordOrSpan==0)
            {
                  //Cordwise since deviation checks
                  j=ObjectVector[i]->cutNum;
                  printf("Cordwise_%d\n",j);

                  //Workpiece checking (both PS and SS sides of workpiece)
                  if(ObjectVector[i]->objectNum==1)
                  {
                        printf("Object1\n");
                        //PS
                        for(k=0;k<numSpanSecs;k++)
                        {
                              dev =
get_min_dist2(ps_pnts[j][k],curve,tempPnt);create_dumb_pnt2(100,tempPnt);

      while(pnt_obj_project(ps_pnts[j][k],curve,y_dir,approxDist,tempPnt)!=0)
                              {
                                    approxDist = approxDist+.005;

                                    counter++;   //same as  count = count+1;
                                    if(counter>100)
                                    {
                                          //Jump out of the while loop
                                          break;
                                    }
                              }

                              if(counter>100)
                              {
                                    dev1 = dev;  // used when pnt_obj_project
fails to find intersection.... forces min dist deviation to continue
                              }
                              else
                              {
                                    dev1 =
diff(ps_pnts[j][k],tempPnt);approxDist=0.005;
                              }

      printf("Obj1_psDev: %lf\t%lf\n",dev,dev1);

                              //Deposit information into vector

      Deviations.push_back(dev);Deviations1.push_back(dev1);      //These are
for both sides
                              counter = 0;//have to reset count
                        }
                        //SS
                        for(k=0;k<numSpanSecs;k++)
                        {

                              dev =
get_min_dist2(ss_pnts[j][k],curve,tempPnt);create_dumb_pnt2(100,tempPnt);
```

```
      while(pnt_obj_project(ss_pnts[j][k],curve,y_dir,approxDist,tempPnt)!=0)
                              {
                                    approxDist = approxDist+.005;
                                    counter++;   //same as  count = count+1;
                                    if(counter>100)
                                    {
                                          //Jump out of the while loop
                                          break;
                                    }
                              }

                              if(counter>100)
                              {
                                    dev1 = dev;  // used when pnt_obj_project
fails to find intersection.... forces min dist deviation to continue
                              }
                              else
                              {
                                    dev1 =
diff(ss_pnts[j][k],tempPnt);approxDist=0.005;
                              }

                              //dev1 =
diff(ss_pnts[j][k],tempPnt);approxDist=0.005;

                              printf("Obj1_ssDev: %lf\t%lf\n",dev,dev1);

                              //Deposit information into vector

      Deviations.push_back(dev);Deviations1.push_back(dev1);
      //These are for both sides
                              counter = 0;
                        }
                  }
                  else if(ObjectVector[i]->objectNum==2)
                  {
                        printf("Object2\n");
                        //PS  Die side of things
                        for(k=0;k<numSpanSecs;k++)
                        {
                              dev =
get_min_dist2(ps_pnts[j][k],curve,tempPnt);create_dumb_pnt2(100,tempPnt);

      while(pnt_obj_project(ps_pnts[j][k],curve,y_dir,approxDist,tempPnt)!=0)
                              {approxDist = approxDist+.005;}
                              dev1 =
diff(ps_pnts[j][k],tempPnt);approxDist=0.005;

                              printf("Obj2_psDev: %lf\t%lf\n",dev,dev1);

                              //Deposit information into vector

      Deviations.push_back(dev);Deviations1.push_back(dev1);
      //These are for both sides
                        }
                  }
```

```
                  else
                  {
                        printf("Object3\n");
                        //SS Die side of things
                        for(k=0;k<numSpanSecs;k++)
                        {
                              dev =
get_min_dist2(ss_pnts[j][k],curve,tempPnt);create_dumb_pnt2(100,tempPnt);

      while(pnt_obj_project(ss_pnts[j][k],curve,y_dir,approxDist,tempPnt)!=0)
                              {approxDist = approxDist+.005;}
                              dev1 =
diff(ss_pnts[j][k],tempPnt);approxDist=0.005;

                              printf("Obj3_psDev: %lf\t%lf\n",dev,dev1);

                              //Deposit information into vector

      Deviations.push_back(dev);Deviations1.push_back(dev1);
      //These are for both sides
                        }
                  }
            }
      }
//return 0;
      double devs = calc_global_deviation(Deviations);
      double devs1 = calc_global_deviation(Deviations1);


      //For the deviations for both sides (all PS and SS)
      //char* devFileCord = "Deviations.txt";
      char* devFileCord = "C:\\Research\\Evaluation\\Deviations.txt";

      FILE* devsOutCord = fopen(devFileCord,"w");
      fprintf(devsOutCord,"minDistDevObjective:\t%lf\n",devs);
      fprintf(devsOutCord,"objInterDevObjective:\t%lf\n",devs1);

      fprintf(devsOutCord,"MINDIST\t\tOBJINTER\n");
      for(i=0;i<(int)Deviations.size();i++)
      {
            fprintf(devsOutCord,"%lf\t%lf\n",Deviations[i],Deviations1[i]);
      }

      fclose(devsOutCord);

      //clean up
      for (i=0;i<numCordSecs;i++)
      {
            for(j=0;j<numSpanSecs;j++)
            {
                  delete [] ps_pnts[i][j];
                  delete [] ss_pnts[i][j];
            }
      }
      delete [] ps_pnts;
      delete [] ss_pnts;
```

```
ObjectVector.~vector();
CordDeviation.~vector();CordDeviation1.~vector();
SpanDeviation.~vector();SpanDeviation1.~vector();
//system("pause");
return 0;}
```

APPENDIX B. MESH GENERATION MODULE

```
!================================================================================
!       Program         mesh.mac
        !Project        Thesis optimization
!       Purpose         To autonomously generate the meshed objects (in ANSYS)for
!                        simulation in DEFORM2D and export a .KEY inputfile for part
!
!       Revision History        March 9, 2007   Started                 KGF
!                               June 26, 2007   Revised for Cord only   KGF
!================================================================================

! Load IGES file
FINISH
/CLEAR,NOSTART

!Sets the filepath name
pathName='C:\Research\'
partsPath='%pathName%PartFiles\'

!Changes the working directory
/cwd, '%pathName%\Meshing\'

num=2

!Read in the file name and imports the UG geometry
*DIM, name2use, STRING,1
*SREAD,name2use ,part,txt
string = name2use(1)

!Used for checking what object number should be assigned
*SREAD, name2use,part,txt,,2
checkString = name2use(1)


! Work around for getting the parameter name for the part file working
*cfopen,'%pathName%Meshing\commandFile',txt      !open file
*CFWRITE,~UGIN,string,prt,partsPath,SURFACES,,0

!*CFWRITE,checkString
*CFCLOS

! Calls the UGIN command
/INPUT,commandFile,txt,,,0

! Plots the areas of model
/NOPR
```

```
/GO
APLOT

! Displays a message of partfile name
!*MSG,ui,itrai
!ps_cordwise_0

! Go into the preprocessor
/PREP7

! Define element types
ET,1,PLANE42,0


!Mesh the parts properly
!the parameter NUMDIV is for the number of divisions to match for
!lines such that mapped meshing can happen
NUMDIV=0
*if,'%checkString%',EQ,'ps',then
	!Mesh the area
	AESIZE,ALL,.2
	*GET,MINAREA,AREA,0,NUM,MIN
	MSHKEY,1
	AMESH,MINAREA
*elseif,'%checkString%',EQ,'ss'
	!Mesh the area
	AESIZE,ALL,.2
	*GET,MINAREA,AREA,0,NUM,MIN
	MSHKEY,1
	AMESH,MINAREA
*else
	!For the prebond option
	*GET,MAXIMUM,LINE,0,COUNT
	*DO,Incr,1,MAXIMUM,1
		*GET,LENGTH,LINE,Incr,LENG
		*IF,LENGTH,LT,1,THEN
			LESIZE,Incr,0.05
		*ELSEIF,LENGTH,GT,1,AND,LENGTH,LT,4
			LESIZE,Incr,0.40
		*ELSE
			LESIZE,Incr,0.2
		*ENDIF
	*ENDDO
	*GET,MINAREA,AREA,0,NUM,MIN
	MSHKEY,2
	AMESH,MINAREA
*endif


!Reorient view
/VIEW,1,,,1
/ANG,1
/REP,FAST
EPLOT
```

```
! For trying to create JPEG of mesh
!/SHOW,JPEG
!/RGB,INDEX,100,100,100,0
!/RGB,INDEX,0,0,0,15
!/DEV,PSFN,NINC
!/gfile,600
!PLNSOL,S,EQV
!/SHOW,CLOSE
!/IMAGE,save,'%path_name%current','jpg'
!/cle
!*ENDDO
!/EXIT

! Get the number of nodes
nsel,all,node
*get,mxnd_,node,,num,max

! Get the number of elements
esel,all,elem
*get,mxelm_,elem,,num,max

! Open new keyword file for simulation input
*cfopen,'%pathName%KeyWordFiles\%string%',KEY

! Check to see what type of object number should be used
! NODENUM is used to locate the proper nodes for the simulation die stop criteria
*if,'%checkString%',EQ,'ps',then
        object=2
        NODENUM=NODE(-10000,-10000,0)
        *vwrite,NODENUM
NODENUM     %I
*elseif,'%checkString%',EQ,'ss'
        object=3
        NODENUM=NODE(-10000,10000,0)
        *vwrite,NODENUM
NODENUM     %I
*else
        !For the prebond option
        object=1
*endif

! Write the output file containing all nodal locations and elem connectivity
*vwrite,object
OBJNAM      %I
*vwrite,string
%C
*vwrite,object
OBJTYP      %I      4       0
*vwrite,object,mxnd_
RZ  %I %I

*DO,i,1,mxnd_,1,
  *get,x_loc,node,i,loc,x
  *get,y_loc,node,i,loc,y
  *vwrite,i,x_loc,y_loc
```

```
   %I %8.4F %8.4F
*ENDDO


!*if,'object',eq,'3',then
!*vwrite,object,mxelm_
!ELMCON   %I %I
!*DO,j,1,mxelm_,1,
!  *get,one_node,elem,j,node,1
!  *get,two_node,elem,j,node,2
!  *get,three_node,elem,j,node,3
!  *get,four_node,elem,j,node,4
!  *vwrite,j,one_node,two_node,three_node,four_node
!  %I %I %I %I %I
!*ENDDO
!*else
*vwrite,object,mxelm_
ELMCON   %I %I
*DO,j,1,mxelm_,1,
  *get,one_node,elem,j,node,1
  *get,two_node,elem,j,node,2
  *get,three_node,elem,j,node,3
  *get,four_node,elem,j,node,4
  *vwrite,j,one_node,two_node,three_node,four_node
  %I %I %I %I %I
*ENDDO
!*endif

! Close the newly created inputfile
*cfclose

Fini
```

APPENDIX C. SIMULATION PREPROCESSING

```
KFREAD
C:\Research\Simulation\SimulationDefaults.KEY
KFREAD
C:\Research\KeywordFiles\prebond_cordwise_0.KEY
KFREAD
C:\Research\KeywordFiles\ps_cordwise_0.KEY
KFREAD
C:\Research\KeywordFiles\ss_cordwise_0.KEY
KFREAD
C:\Research\Simulation\TEMPFIX.KEY

OBJPOS 2 2 0.0001 1 0 -1
OBJPOS 3 2 0.0001 1 0 1
OBJPOS 4 2 0.0001 2 0 -1
OBJPOS 5 2 0.0001 3 0 1

CNTACT 1 2 1
FRCFAC 1 2 1 0 0.3
CNTACT 1 3 1
FRCFAC 1 3 1 0 0.3
CNTACT 2 4 1
FRCFAC 2 4 1 0 1.0
CNTACT 3 5 1
FRCFAC 3 5 1 0 1.0

GENCTC 0.01

REFPOS 2 2 1
REFPOS 3 2 1
MDSOBJ 2 3 2 0.15

KFWRITE
C:\Research\Simulation\I-O\INPUT_cordwise_4.KEY
GENDB 2
C:\Research\Simulation\Simulation.DB
```

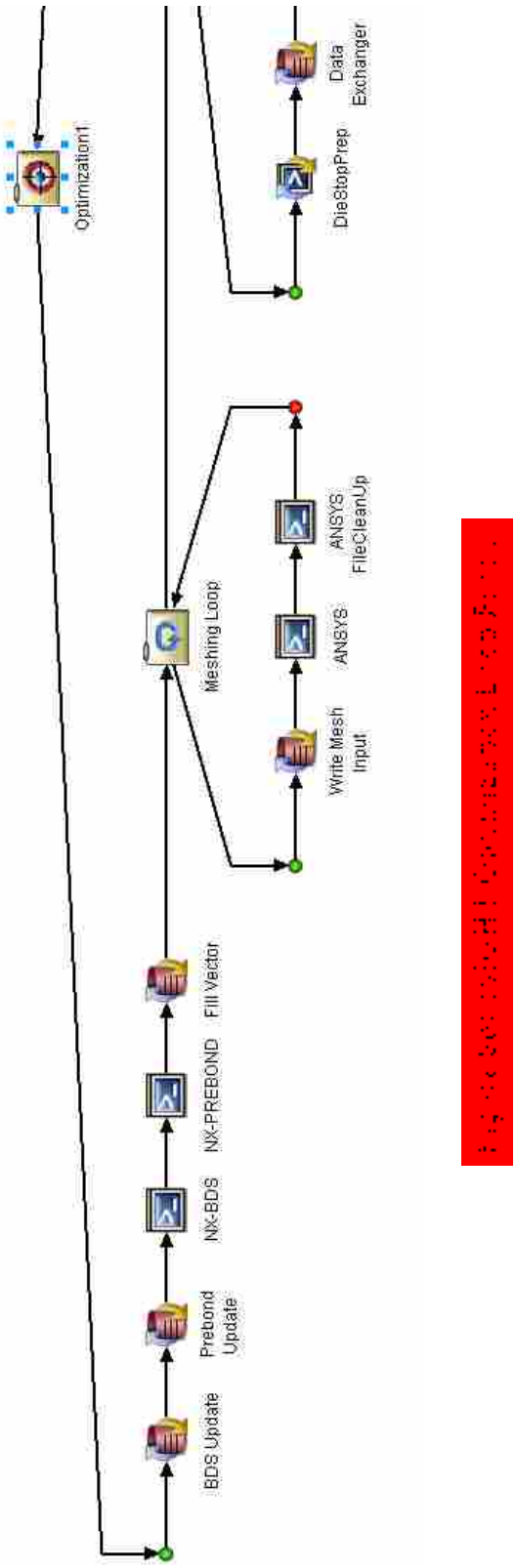
APPENDIX D. SIMULATION POSTPROCESSING

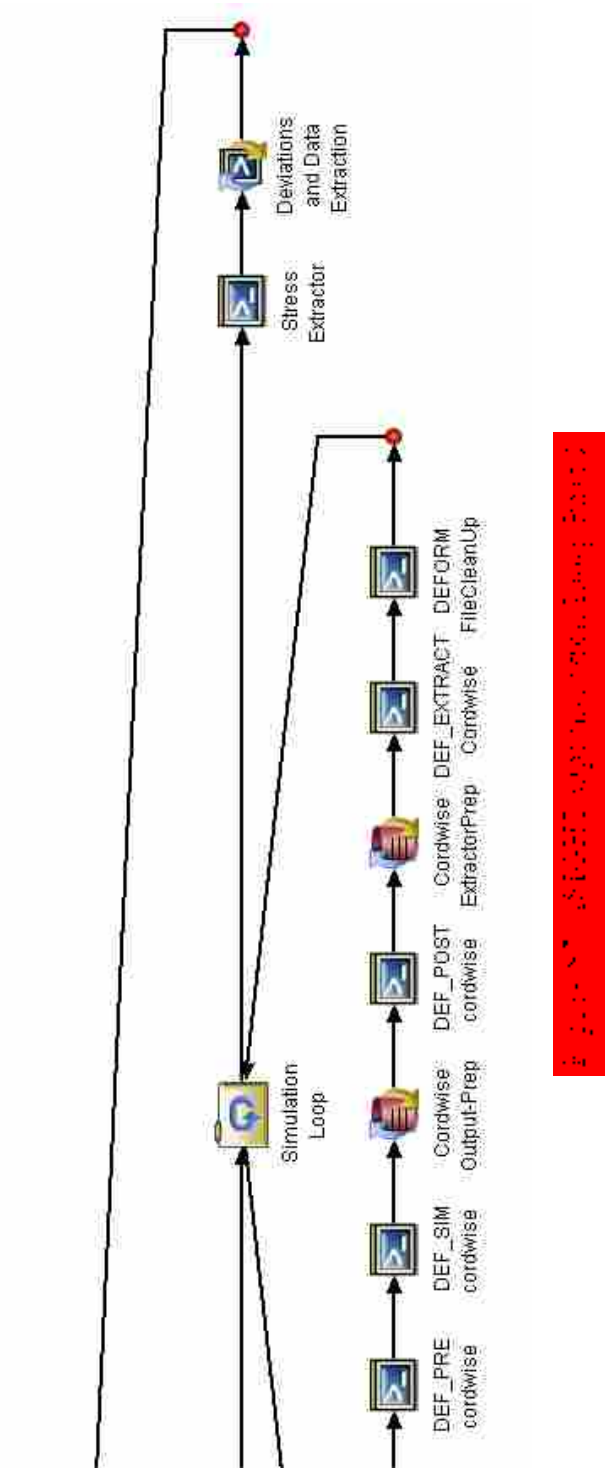
```
2
1
C:\Research\Simulation\I-O\OUTPUT_cordwise_0.KEY

E
5
1
1
5
3
E
1
2
5
3
E
1
3
5
3
E
E
8
C:\Research\Simulation\I-O\OUTPUT_cordwise_0.KEY
Y

E
Y
```

APPENDIX E. ISIGHT OPTIMIZATION LOOP


Optimization1
Data Exchanger
DieStopPrep
ANSYS FileCleanUp
ANSYS
Write Mesh Input
Meshing Loop
Fill Vector
NX-PREBOND
NX-BDS
Prebond Update
BDS Update


Deviations and Data Extraction
Stress Extractor
Simulation Loop
DEF_PRE cordwise
DEF_SIM cordwise
Cordwise Output-Prep
DEF_PDST cordwise
Cordwise ExtractorPrep
DEF_EXTRACT Cordwise
DEFORM FileCleanUp

APPENDIX F. JMP INPUT DECK

[illegible]

Trial No.	Indiv Gang	Group Gang	Max	Case	Indiv Case	Group Case	Max Case	Indiv Group	Indiv Max	Group Max	Indiv Group Case	Indiv Max Case	Group Max Case	Index
1	0	0.017	0.007	0	0	0	0	0	0	0.000675	0	0	0	0
2	0	0.015	0.035	1	0	0.015	0.035			0.000525			0.000525	
3	0.007	0.007	0.002	0	0	0	0	0.000049	0.000014	0.000014	0	0	0	0.000049
4	0.007	0.007	0.002	1	0.007	0.007	0.002	0.000049	0.000014	0.000014	0.000049	0.000014	0.000014	0.000049
5	0.007	0.007	0.038	0	0			0.000049	0.000266	0.000266				0.000049
6	0.007	0.007	0.008	1	0.007	0.007	0.008	0.000049	0.000056	0.000056	0.000049	0.000056	0.000056	0.000049
7	0.007	0.021	0.002	0	0	0	0	0.000147	0.000014	0.000042	0	0	0	0.000049
8	0.007	0.023	0.002	1	0.007	0.023	0.002	0.000161	0.000014	0.000046	0.000161	0.000014	0.000046	0.000049
9	0.007	0.021	0.008	0	0	0	0	0.000147	0.000056	0.000168	0	0	0	0.000049
10	0.007	0.023	0.038	1	0.007	0.023	0.038	0.000161	0.000266	0.000874	0.000161	0.000266	0.000874	0.000049
11	0.015	0	0.005	0	0	0	0	0	0.000075	0	0	0	0	0.000225
12	0.015	0	0.005	1	0.015	0	0.005	0	0.000075	0	0	0.000075	0	0.000225
13	0.015	0.015	0	0	0			0.000225						0.000225
14	0.015	0.015	0	1	0.015	0.015	0	0.000225	0	0	0.000225	0	0	0.000225
15	0.015	0.015	0.005	0	0	0	0	0.000225	0.000075	0.000075	0	0	0	0.000225
16	0.015	0.015	0.005	1	0.015	0.015	0.005	0.000225	0.000075	0.000075	0.000225	0.000075	0.000075	0.000225
17	0.015	0.015	0.01	0	0	0	0	0.000225	0.00015	0.00015	0	0	0	0.000225
18	0.015	0.035	0.01	1	0.015	0.035	0.01	0.000525	0.00015	0.00035	0.000525	0.00015	0.00035	0.000225
19	0.015	0.03	0.005	0	0	0	0	0.00045	0.000075	0.00015	0	0	0	0.000225
20	0.015	0.03	0.005	1	0.015	0.03	0.005	0.00045	0.000075	0.00015	0.00045	0.000075	0.00015	0.000225
21	0.023	0.007	0.002	0	0			0.000161	0.000046	0.000014				0.000529
22	0.023	0.007	0.002	1	0.023	0.007	0.002	0.000161	0.000046	0.000014	0.000161	0.000046	0.000014	0.000529
23	0.023	0.007	0.008	0	0	0	0	0.000161	0.000184	0.000056	0	0	0	0.000529
24	0.023	0.007	0.008	1	0.023	0.007	0.008	0.000161	0.000184	0.000056	0.000161	0.000184	0.000056	0.000529
25	0.023	0.023	0.002	0	0	0	0	0.000529	0.000046	0.000046	0	0	0	0.000529
26	0.023	0.023	0.035	1	0.023	0.023	0.035	0.000529	0.000848	0.000848	0.000529	0.000848	0.000848	0.000529
27	0.023	0.023	0.008	0	0	0	0	0.000529	0.000184	0.000184	0	0	0	0.000529
28	0.023	0.023	0.008	1	0.023	0.023	0.008	0.000529	0.000184	0.000184	0.000529	0.000184	0.000184	0.000529
29	0.03	0.015	0.025	0	0			0.00045	0.00075	0.000375				0.0009
30	0.03	0.015	0.035	1	0.03	0.015	0.035	0.00045	0.00105	0.000525	0.00045	0.00105	0.000525	0.0009

Table 21: JMP Input Deck Part 2