



2011-10-05

# Automated Flow Path Design Optimization Using Mesh Morphing

William Dennis Gough

*Brigham Young University - Provo*

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>



Part of the [Mechanical Engineering Commons](#)

---

## BYU ScholarsArchive Citation

Gough, William Dennis, "Automated Flow Path Design Optimization Using Mesh Morphing" (2011). *All Theses and Dissertations*. 2843.

<https://scholarsarchive.byu.edu/etd/2843>

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact [scholarsarchive@byu.edu](mailto:scholarsarchive@byu.edu), [ellen\\_amatangelo@byu.edu](mailto:ellen_amatangelo@byu.edu).

Automated Flow Path Design Optimization  
Using Mesh Morphing

William D. Gough

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  
Steven L. Gorrell  
Dale R. Tree

Department of Mechanical Engineering  
Brigham Young University  
December 2011

Copyright © 2011 William D. Gough

All Rights Reserved



## ABSTRACT

### Automated Flow Path Design Optimization Using Mesh Morphing

William D. Gough  
Department of Mechanical Engineering  
Master of Science

The use of CAD models and CFD analysis has become an essential part of fluid flow design. To reduce the time spent determining a design, optimization frameworks have been implemented to automate the process. Mesh morphing has been implemented within these frameworks to further reduce the time needed. While optimization methods have been developed to optimize a fluid flow path, the optimum design needs to be recreated in a CAD model. A method has been developed which eliminates the need to recreate the optimal results in CAD. This is accomplished by using mesh morphing, CAD and CFD together in an optimization framework. The method developed has been implemented with a significant time savings over the use of a traditional meshing optimization framework.

Keywords: William Gough, optimization, mesh morphing, intake design, design automation



## ACKNOWLEDGMENTS

I would like to express my appreciation for all those who have helped me to complete this research. I thank Dr. Jensen, for his support, guidance and unwavering belief in me to accomplish something great. I thank Dr. Gorrell and Dr. Tree for their guidance and input during this process. I am grateful to Ron Thue at Altair Engineering and the team at the Sandy, Utah office for their help and guidance with HyperWorks. I also thank Mark Landon and John Jenkins from Optimal Solutions for their guidance and help.

Finally, I thank my wife, Katie, for her constant support and encouragement and our children for their sacrifice and support.



## TABLE OF CONTENTS

<b>LIST OF TABLES .....</b>	<b>vii</b>
<b>LIST OF FIGURES .....</b>	<b>ix</b>
<b>1 Introduction.....</b>	<b>1</b>
1.1 Problem Statement .....	2
1.2 Objectives .....	3
1.3 Delimitations.....	3
<b>2 Background .....</b>	<b>5</b>
2.1 Intake Manifold Design .....	5
2.2 Parametric Computer Aided Design .....	6
2.3 Application Programming Interface .....	7
2.4 Mesh Deformation Technology .....	9
2.5 Computer Aided Design and Computer Aided Engineering Tools Integration....	15
2.5.1 Attribute Management .....	15
2.6 Optimization .....	16
2.7 Flow Optimization Methods .....	17
<b>3 Method .....</b>	<b>21</b>
3.1 Parametric Flow Path Model .....	22
3.2 Flow Path Mesh Generation.....	24
3.3 Morphing Procedure .....	24
3.4 CFD Procedure.....	26
3.5 Optimization .....	27
3.6 Summary .....	30



<b>4</b>	<b>Implementation .....</b>	<b>31</b>
4.1	Parametric Model.....	32
4.2	Mesh Generation.....	36
4.3	Mesh Morphing Process .....	39
4.4	Flow Model Analysis .....	42
4.5	Optimization Framework .....	46
4.5.1	Optimization Using Traditional Method.....	51
4.5.2	Optimization Using Morphing Method .....	51
<b>5</b>	<b>Results .....</b>	<b>53</b>
5.1	Traditional Method Loop.....	53
5.2	Morph Method Loop.....	56
5.3	Traditional and Morph Loop Comparison .....	58
<b>6</b>	<b>Conclusions.....</b>	<b>63</b>
6.1	Future Work .....	64
	<b>REFERENCES.....</b>	<b>67</b>
	<b>Appendix A. C++ NX Executable code .....</b>	<b>71</b>
A.1	Tokenizer Header File.....	71
A.2	NX Main .cpp File .....	71
	<b>Appendix B. HyperMorph TCL script for meshing.....</b>	<b>75</b>
	<b>Appendix C. HyperMorph TCL Script .....</b>	<b>79</b>
	<b>Appendix D. Fluent Journal File .....</b>	<b>83</b>
D.1	Mesh Fluent Journal.....	83
D.2	Morph Journal File.....	84

## LIST OF TABLES

Table 3-1: Flow Feature Identification .....	23
Table 3-2: Geometric Feature Identification.....	23
Table 4-1: Parameter Variables and Limits .....	34
Table 4-2: Model Flow Features.....	37
Table 4-3: Mesh Type and Size .....	39
Table 4-4: Model Geometric Features .....	41
Table 4-5: Flow Boundary Conditions .....	42
Table 4-6: Grid Independence Mesh Size and Flow Results .....	43
Table 4-7: Grid Independence Results Comparison .....	43
Table 4-8: Flow Solver Setup .....	45
Table 5-1: Computer Specs.....	53
Table 5-2: Traditional Optimization Results .....	54
Table 5-3: Morph Optimization Results .....	56

Table 5-4: Loop Comparison .....58

## LIST OF FIGURES

Figure 2-1: Smith Port Deformation Example.....	10
Figure 2-2: Morph Volume.....	12
Figure 2-3: Morphed Volume Comparison.....	13
Figure 2-4: Node Deformation Example .....	13
Figure 2-5: Domain Morph Example.....	14
Figure 3-1: Optimization Framework Overview .....	21
Figure 4-1: Optimization Framework Implementation.....	31
Figure 4-2: Design Space.....	32
Figure 4-3: Control Dimensions and Parameters.....	33
Figure 4-4: Control Dimensions End View .....	33
Figure 4-5: $Pc_1 = 28\text{mm}$ .....	35
Figure 4-6: $Pc_1 = 26\text{mm}$ .....	35
Figure 4-7: Mesh Sample.....	38

Figure 4-8: Optimization Loop Example .....	47
Figure 4-9: Optimization Settings.....	48
Figure 5-1: Traditional Velocity Contour Comparison .....	55
Figure 5-2: Traditional Static Pressure Contour Comparison.....	56
Figure 5-3: Morphing Velocity Contour Comparison .....	57
Figure 5-4: Morphing Static Pressure Comparison .....	58
Figure 5-5: Optimal Velocity Comparison .....	59
Figure 5-6: Elbow Velocity Profile.....	59
Figure 5-7: Pressure Profile Comparison.....	60

## **1 INTRODUCTION**

Automotive engine development is a critical part of vehicle development. An extremely competitive market, customer expectations and government regulations all contribute to the need to continually improve the automotive engine. Some of the critical factors of engine development are power, efficiency, durability, noise, vibration, and cost to mention a few. Because of the many different factors of development, a broad spectrum of expertise is needed in many areas and needs to come together seamlessly to be both effective and competitive. An efficient design and development process is needed to be able to accomplish the task of efficiently bringing all these disciplines together for a desirable final product. To this end, the use of computer aided design, analysis, and manufacturing tools (CAx tools) provide a means to accomplish this automation. Though these tools have vastly increased the productivity of engineering development in general, there is still great potential for the implementation of these tools in the development of the automotive engine.

With minimal user input, an ideal automotive engine development tool would be able to automatically generate the geometry, perform the analyses, and optimize an engine or sub-assembly based on the needed design requirements. The creation of this tool must effectively use CAx tools and include knowledge based engineering (KBE), CAD-centric and database-centric model and information management, efficient handling and passing of data from one process or program to another, robust parametric model creation, efficient and automated

generation of analysis models and optimization updates and many others. The design process should be fast and simple to extract the necessary information from the various analyses and optimization routines. The responsible personnel should be able to quickly select the most appropriate design based on the available information that has been provided through the design and analysis processes. While this comprehensive tool does not currently exist, there are many examples of automated portions and tasks. These elemental tools have greatly contributed to the reduction of time and costs of doing laborious engineering tasks, while contributing to better designs. A major component that enables engines to produce power is the air intake flow system. The power an engine is able to produce is directly related to the amount of air that flows through the engine. Because of the significant impact airflow design can have on performance and efficiency, a tool is being considered that can automate much of the CAx design of this process. The motivation for this research is to improve the design of intake manifolds. This will be accomplished by developing a method that will aid in the design process of manifold runners, which is the focus of this research. The method that will be developed for this research will be an important step towards the development of a complete internal combustion engine tool.

## **1.1 Problem Statement**

Intake manifold design for internal combustion engines currently takes a significant amount of time to perform. Steps include generating the initial geometry for the space and performance requirements, setting up and performing analysis for a given design, adjusting the design to meet specified requirements, and repeating the process until a satisfactory flow design is achieved. Frequently, engines are used in different vehicles where similar engine requirements need to be met in a different design space. Changes in the new design space for an engine, as well as changes in the required engine driving dynamics directly relate to the design or

redesign of the intake manifold. Going forward, design, redesign and analysis will be considered the same. A change in a manifold design is mandated whenever the intake geometry is obstructed or when the engine performance characteristics need adjustment. These necessitate frequent redesign of an intake manifold to meet the new space and/or performance requirements. To reduce the time it takes for these changes, an automated process is needed to reduce the amount of time both for the initial design, as well as for changing performance requirements and space constraints in the engine compartment.

## **1.2 Objectives**

The Objective of this research is to reduce the amount of time it takes to design and analyze an intake manifold runner by implementing design automation with CAx tools to optimize the flow path of an intake runner. Two parts of this process will receive special focus.

1. . Integrate pipe flow analysis process from CAD, to meshing/morphing utility, to CFD package, via an optimization framework.
2. Optimize the shape of an intake runner with full or partial automation.

## **1.3 Delimitations**

Ideally, a tool that would both generate the geometry and then perform an optimization on that design would be the purpose and intent of this research. This analysis would be simulated using the desired engine operating conditions with a fully operating engine simulation tool and detailed 3D analysis. While the ideal tool for this process would include the complete intake system, the development of a tool this complex is beyond the scope of this thesis. The scope will be limited to a steady state case with the intent to develop basic optimization methods for pipe flow. The methods developed may be implemented into a complete analysis tool that



would include the complete intake system in a transient 3D analysis simulation. Within this thesis, fundamentals will be developed which may be built on to create a more complete optimization tool.

## **2 BACKGROUND**

As mentioned, tools have been developed which implement parts of an automated design process for engine development. While these tools and processes demonstrate many of the capabilities possible, none have been found which completely utilize the capabilities currently available through automation, integration and optimization. In this chapter, an overview of currently available functions and basic methods will be covered. Also, different tools and methods already developed will be discussed including their advantages and disadvantages. This will lay the foundation for how these different methods may be used in unison to the greatest advantage.

### **2.1 Intake Manifold Design**

As mentioned previously, intake manifolds have a significant effect on engine performance. Restrictions in the air flow such as an air filter, throttle plate, turns, or other obstructions result in a loss of pressure from the inlet at atmospheric pressure to the pressure into the cylinder. This pressure loss has the effect of making the engine underperform because work is required to bring the air in through the obstacles of the intake system. This is sometimes referred to as pumping work. To reduce the pressure losses it is important that a flow path with the least amount of pressure drop is used. Another critical factor about the air flow through an intake manifold is that it is transient. Because the air flow into a cylinder is constantly being

stopped and started pressure waves are formed within the intake manifold. Depending on the length of the intake runners, valve timing and other factors, these pressure waves may be reflected at the correct time at a specific engine speed to achieve additional power and efficiency. This effect is referred to as a ram effect [1]. To take advantage of this effect over a broader range of engine speeds, some intake manifolds have mechanisms that effectively change the length of the intake runner so that better performance can be achieved over a broader engine operating range.

Many different factors affect intake flow dynamics. Shape and cross-sections are two of the important factors that determine a well-designed runner. For a complete discussion of engine fundamentals and flow for internal combustion engines such as Heywood [1] and Ferguson and Kirkpatrick [2].

## **2.2 Parametric Computer Aided Design**

It has been estimated that approximately 80% of a designer's time is spent redesigning for different applications [3]. This emphasizes the need to reuse a part or part assembly and model designs without manual redesign. Parametric modeling enables a model to be reused many times during a part or assembly lifecycle [4]. Parametric computer aided design (CAD) refers to the ability to update a CAD model with new dimensions or parameters without the need to manually recreate or redesign the geometry. Parametric CAD enables changes to quickly be made so design studies and optimizations can be performed for a specific application. These capabilities exist in most modern CAD packages such as Siemens NX, Dassault Systems CATIA, Autodesk Inventor and PTC Creo Elements/Pro.

What enables a model to be parametric is the ability to assign a name and dimension to a given feature on a part or assembly and associate these parameters with other features or

dimensions. When a part or model has all of the dimensions assigned and constrained correctly, one or more dimensions may be adjusted and the model will update to accommodate the changes. If the parametric model is correctly constrained then little to no work will be needed to reuse or adapt the model to a new assembly [5].

Dimensions on a part may have multiple constraints to make a model work given differing requirements. As a simple example, an annulus may be considered. If the inner radius needs to be changed and it becomes larger than the original outer radius, then the model is no longer valid, but if a constraint relative to the inner radius is placed on the outer radius, such as 1.5 times the inner radius, then as the inner radius changes, the outer radius will automatically adjust. This principle is applied to very complex models and assemblies, making it possible to eliminate the need for a complete redesign when changes need to be made to different parts of the model. It is important that each of the parameters in a model be assigned correctly when making a parametric model so that the model will not fail when a specific design change is implemented. The time it will take to make a completely parametric model is longer than a non-parametric model, but the time it will take to adapt that model in the future can be nearly completely eliminated [6].

Parametric CAD enables design time to be greatly reduced because models are able to be adapted to similar applications with minimal user input. The ability to update a model this way makes it possible to implement changes automatically via programming. The ability to program CAD packages and other design tools is an integral part to design automation.

### **2.3 Application Programming Interface**

An application programming interface (API) is a feature that enables a user access to features and functions that the software uses [7]. This enables a user the ability to add

functionality, automate repetitive tasks and program routines to be used interactively in executable format or in batch mode.

A software package may either have a common programming language that can be used or it may have its own custom programming language. For example, the Siemens NX API can be programmed in multiple languages, such as C, C++, Java and Visual Basic. Other programs may have their own specific language that is used, such as the Ansys Developer Programming Language (ADPL). Many benefits are enabled by an API. One is that modern CAD packages and other Computer Aided engineering tools (CAx) have the ability to be programmed to automate tasks that would take extensive amounts of time to complete when done manually. If a program is developed that performs a series of tasks automatically, an engineer can use that time for other development and research. There are numerous examples of different applications and customizations that have been implemented using an API to customize different applications. Using an API provides access to internal commands of a software tool so that they may be used to create a program for that tool. In NX it is possible to write a program to perform a basic geometric operation, such as make a cylinder of a specified size. An example by Hepworth [8] demonstrates the use of a custom geometry kernel for composites design to speed up the design and analysis process. In this example, the API was used to implement the kernel that replaced the native geometry kernel. Because the kernel access through the API was indirect, a different kernel was used to speed up the process. Another example of customization involves that of implementing communication tools to capture design rationale. In this example, Mix [9] shows how communication tools can be used to capture and record this information so that in the future other engineers would be able to determine the background to decisions made during the design

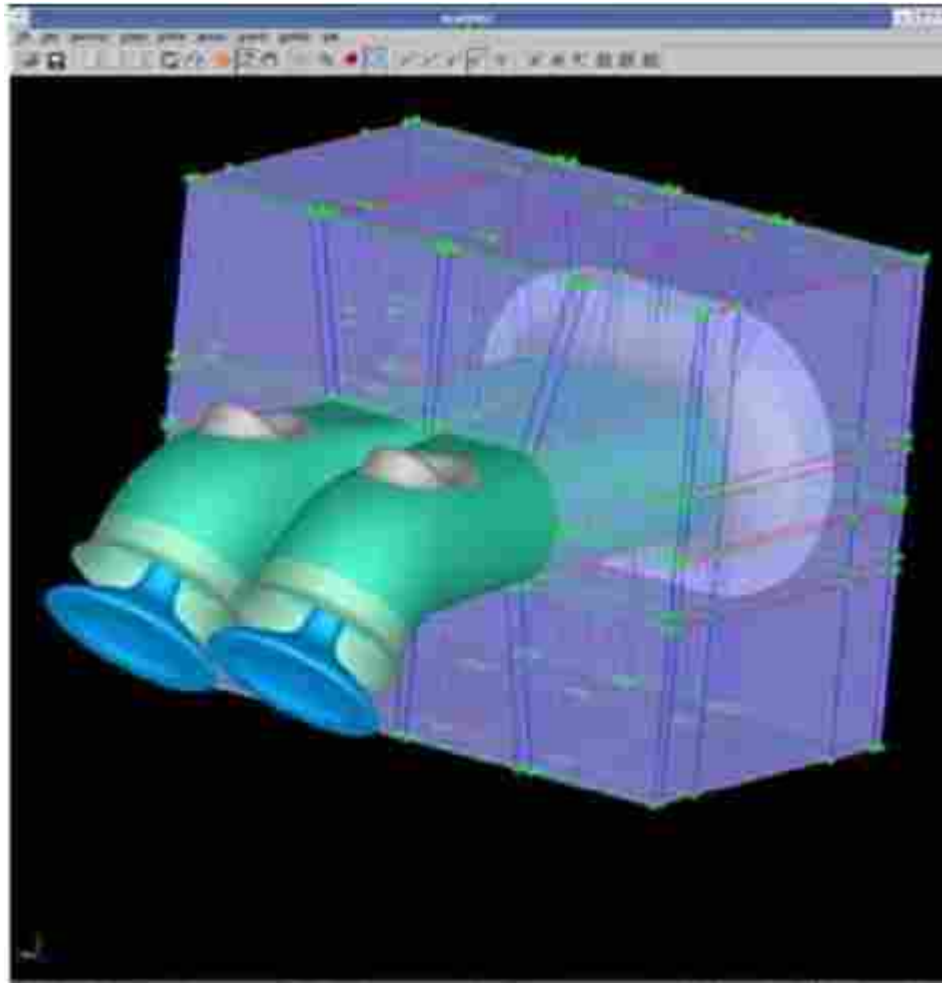
and analysis process. This method involves the implementation of Skype into NX to facilitate design.

API's can be used in multiple ways to implement needed customization for a specific design or process. As a result engineers are able to save the time and money that otherwise would have been used without the utilization of a programming interface.

## **2.4 Mesh Deformation Technology**

“Over 50% of the time spent in industry on a CFD project is devoted to the definition of the domain geometry and grid generation” [10]. While this comment is general and not focused on a specific industry, it is indicative of the amount of time that is used for grid generation for CFD projects. The model preparation for a design to be analyzed in CFD takes a significant amount of time. Mesh Generation alone can be very time consuming depending on the size and type of the model, and to mesh similar designs will take about the same amount of time. Performing this task repeatedly for an optimization is both time consuming and inefficient. This type of change hereafter will be referred to as the traditional method. To help eliminate this problem, a process to change the mesh without re-meshing the entire part has been developed which is referred to as mesh deformation. Mesh deformation technology (also referred to as mesh or grid morphing) enables the mesh of an analysis model to be updated or changed from a new or updated design in CAD without re-generating the mesh. This makes it possible to perform design studies much more rapidly [11], [12] because the process of modifying the original part, and then re-meshing is eliminated from the optimization process. Those two processes can be replaced by changing the original mesh and then reanalyzing the model [13]. An example of mesh morphing is shown in an optimization for engine intake ports by Smith [14]. In Figure 2-1 a morphing volume used to change the shape of the ports is shown. Within

this shape it was possible to change the mesh with the control volume rather than generate new geometry and a new mesh for each iteration, eliminating the time needed for remeshing.



**Figure 2-1: Smith Port Deformation Example**

Mesh deformation enables a meshed model to be changed either manually or programmatically, but there are rules that need to be followed for the mesh morphing process to be valid. In an application discussing how a mesh deforms, Gao [15] states four rules that need to be followed for the mesh deformation process. Following is a quote of those rules:

1. No grid line crossover may occur to ensure the positivity of the cell volume.
2. The grid orthogonality must be, at least nearly, preserved. It is important to ensure such orthogonality in the viscous region near the surface grid..
3. The transition of the deformation from that of the surface grid to the zero deformation in the far-field grid should be kept as smooth as possible.
4. The clustering of the near-body grid cannot be compromised as the body deforms. The grid must maintain its clustering in the viscous layer.

Rule 2 refers to structured grids, while this rule does not directly apply to unstructured grids, it can safely be assumed that good quality of the grid needs to be maintained in the viscous region. These rules ensure that when a mesh model is morphed that the quality of the mesh is not compromised, thus enabling the analysis to be performed [15]. As long as these rules are followed then the model may be morphed to adapt to the changes that are desired. However, when keeping a good quality mesh is not possible because the morphing change is too drastic or large the model may need to be partially or completely re-meshed. Morphing tools can check the quality of a mesh after it has been morphed to verify that the changes made are not beyond the limits of the morphing range for the model. Some tools can repair or correct problems which are encountered after a morph has been done. Morphing limits are dependent on the model, the magnitude of the changes made, and the type of morphing procedure used.

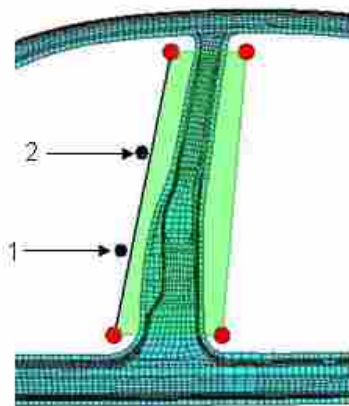
Mesh morphing can also have a significant advantage in CFD analysis with design changes. Often the solver can start from the previous solution instead of resolving the entire model because the same mesh is used. This may result in a significant reduction in the amount of time required for subsequent CFD models to be solved because they start with the previous solution [11].



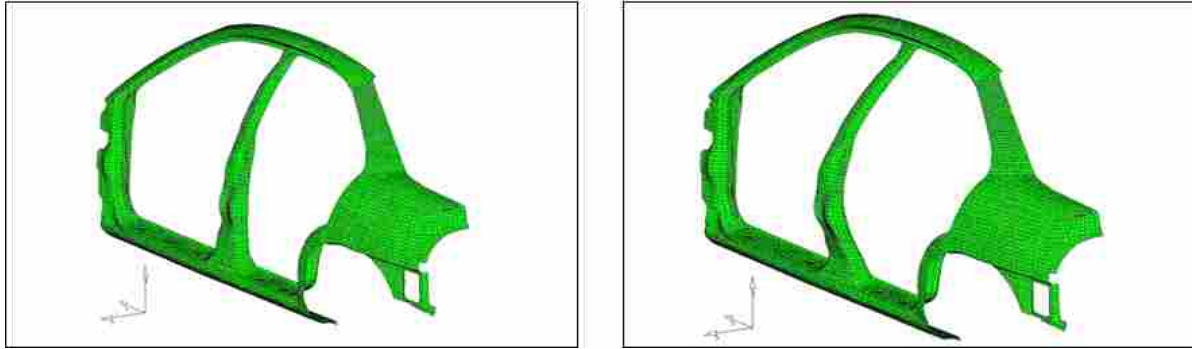
There are different methods for morphing a mesh within morphing programs. While the names used may differ between the different morphing software packages, the principles are similar. As outlined in Altair HyperWorks Documentation [16] these methods include:

- Volume Deformation
- Direct Node Deformation
- Domains and Handles Deformation

Volume deformation involves creating a geometric volume around the mesh that defines how the mesh will move within this volume. This method deforms the mesh based on how the control volume is changed. This means the mesh is deformed based on the algorithm controlling the deformation inside of the volume. An example of this can be seen from a HyperMorph tutorial example on morph volumes HM-3550 [16]. Figure 2-2, from the HyperWorks Tutorial, shows the volume around the portion of the mesh that will be changed. In Figure 2-3, from the HyperWorks Tutorial, the change is shown on the right with the B-pillar morphed outward, compared to the original B-pillar in the standard location on the left.

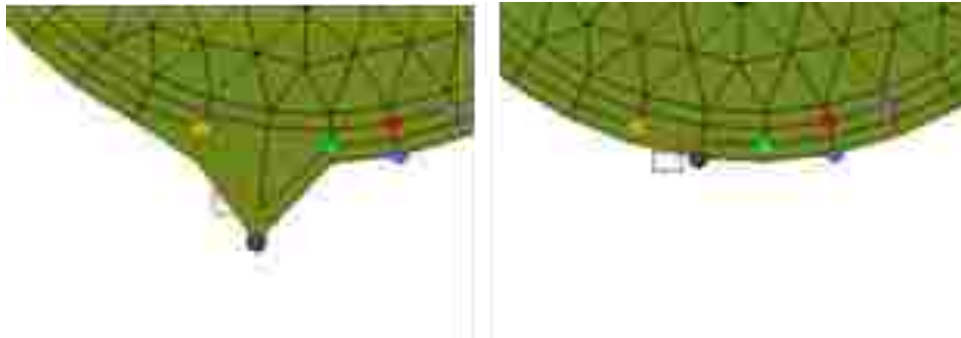


**Figure 2-2: Morph Volume**



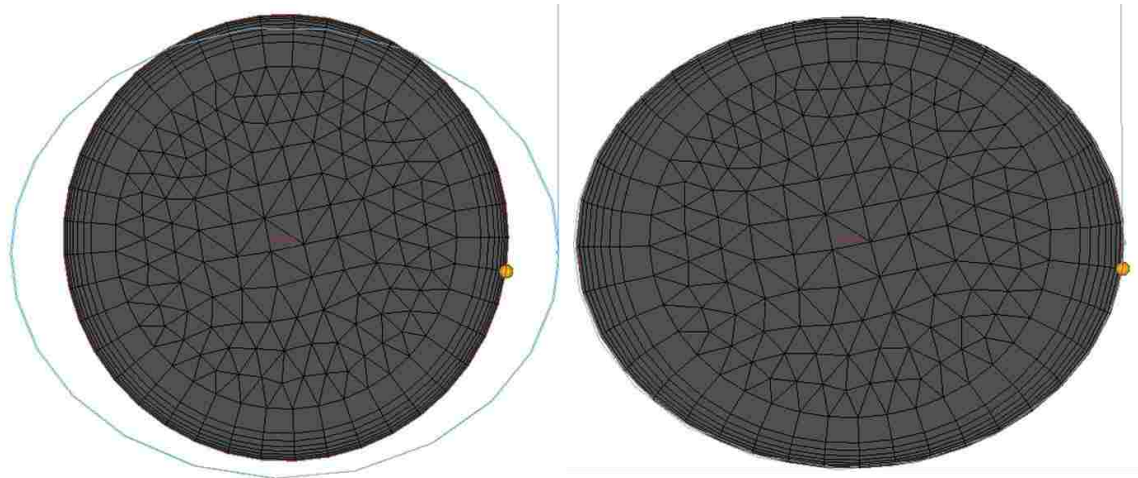
**Figure 2-3: Morphed Volume Comparison**

Direct Node Deformation is the method of deforming the mesh with the specific nodes of the mesh. This method allows very specific movements of the mesh to take place so that detailed changes can be made at a very low level. Nodes can also be moved together in groups while keeping others stationary with specified constraints. A simple example is shown in Figure 2-4 which shows how a single node can be deformed while keeping the rest stationary.



**Figure 2-4: Node Deformation Example**

Domains and Control Points deformation enables areas of a mesh to be assigned to a specific group, and control points can be used to manipulate these groups, or the group can be morphed to a specified shape. This also can be used to morph a base mesh to a new shape that can be imported from a CAD model. An example of this method is demonstrated in Figure 2-5 where the base shape on the left is morphed to the blue line from a CAD model.



**Figure 2-5: Domain Morph Example**

Different methods may be used together to make the necessary changes that are needed for the desired design changes on an analysis model. This enables many different uses of morphing for various changes to an analysis model. Changes made to the morph can be either somewhat arbitrary within a specific range, or the changes made can match specific geometric changes that are generated in a CAD model.

Mesh morphing may also be used to make significant changes to an analysis model. Using mesh morphing, a model can be directly modified to be used in analysis by a CFD analyst to make changes while waiting for a new design change or model. This would enable a CFD analyst to participate in the design process at an earlier stage of design and development. This model could also be used for an optimization process. An example of this applied to vehicle development is given by Singh [13].

Another time saving results when a morphed CFD model begins the new solution iterations from the previous solution resulting in large reductions in the amount of time needed for the new solution [17]. As a result of the substantial time reductions gained in CFD analysis and the ability to more quickly move from previous designs to new design iterations, mesh

morphing enables significant time and cost reductions in the overall design and analysis process. This effect is only increased when integrated with other CAx tools to greatly reduce CFD pre-processing, analysis time and the total time to an optimal design.

## **2.5 Computer Aided Design and Computer Aided Engineering Tools Integration**

The integration of CAD and Computer Aided Engineering/Manufacturing (CAx) tools is made possible by the use of API's and the ability to run a CAx program in batch or as an executable. The advantage to this is that a user can program the software to perform the specific tasks needed. Many examples of this can be found in the literature [18-20]. Once the different software packages have been programmed they may be integrated with an optimization framework so that different tasks may be performed iteratively, thus converging on a solution [19]. This allows flexibility for the engineer to determine the parameters or regions which need to be optimized, set the parameters to be studied or optimized and, then let the optimization framework determine the design. Using other methods previously mentioned would allow the designer or engineer to work on new work rather than continually monitoring and controlling the optimization process, making changes and re-running analyses manually, as well as post-processing the information.

### **2.5.1 Attribute Management**

One of the key elements in being able to integrate CAx programs is assigning attributes to the CAD model. Attributes are properties that can be assigned to different features in the CAD model that may represent properties, materials, boundary conditions, etc. An example of the use of attributes in a CAD model is described by King in the development of a CAD-centric model to CFD [21]. An example of using attributes can be shown with a property of a flow inlet.

The feature in CAD that represents a flow inlet can be assigned an attribute, such as a velocity. This attribute is then passed with this feature identified as a flow inlet. This may be used to set up the mesh and the flow condition for the CFD solver. Using attributes enables one to programmatically use CAx programs together, passing essential information about the original model to downstream programs. This enables the ability to automate many different commands because those attributes in the model can be used programmatically, eliminating the need to manually reassign those same attributes in subsequent programs for each case.

## **2.6 Optimization**

Optimization seems to be the essence of modern day engineering with any product development, improvement or redesign. By definition, optimal means “most desirable possible under a restriction expressed or implied” [22]. There are multiple optimization frameworks available today which apply algorithms that can point to the best possible solution given a set of requirements or constraints. Some of these frameworks include Isight, Optdes-x, Frontier, HyperStudy and modeFRONTIER to mention a few. Optimization methods are being used extensively in every aspect of engineering, and large improvements have been made as a result of using optimization algorithms and software integration.

Optimization frameworks can be used to integrate or control multiple programs to perform an optimization. As an example, a specific part can be optimized using an optimization framework in the following process. The optimization framework can be given the design variables that will be studied, such as hole radius and plate thickness. The framework will begin the optimization by sending a command to an analysis program to perform an analysis and return the results to a specified location. Those results can then be analyzed and the optimization program can then start a CAD program to create new geometry with different parameters. The

new geometry would then be sent to a meshing tool, meshed, then the analysis performed again and the results analyzed by the framework. To continue with the optimization, new parameters would then be passed to the CAD program, new geometry created, and the cycle would repeat until the optimal result is achieved. This optimization process can be used on something as simple as an I-beam, or as complex as an airframe structure and assembly for an aircraft. While the preparation to set up such an optimization process would have a significant initial time investment, the advantage would be that further design iterations would be automated and optimized by the framework.

Optimization frameworks can use many different algorithms to achieve the optimum for a given case. Many different optimization algorithms are available and implemented into available optimization frameworks. The type of problem that needs to be optimized will determine the type of algorithm that may be used. Types of optimization algorithms include gradient-based algorithms, genetic algorithms, simulated annealing algorithms, branch and bound and many others with variations of each type. Some of these are for a continuous design space while others are for discrete design space. Here, it is not intended to give a description of each type of algorithm, but to give an overview of the options and some of the details needed in setting up an optimization framework. For more information and background on optimization algorithms, suitable types for different design problems as well as optimization theory, various sources may be referred to such as “Optimization Based Design” by Alan Parkinson [23] and others [24], [25].

## **2.7 Flow Optimization Methods**

To develop a model for optimization of inlet port design Blaxill et al [26] used a parametric CAD model defining the desired design space to test multiple CFD cases. A base

case was used to determine the best grid and necessary CFD parameters for the remaining designs, then a script was used to create the remaining designs using the best-determined CFD setup. Once each case was created, the models were then run in batch mode, each of the 45 cases taking approximately eight hours to complete, for about 360 hours total CFD processing. The grid size of each case was approximately 60,000 cells and CFD processing was done in parallel on a dual core Silicon Graphic Power Challenge. This amount does not include the time that was taken to mesh each model and perform the necessary setup needed. Because of the automated creation of the remaining models after the first, there was a significant amount of time savings, but as can be seen in the following examples, the amount of time needed to perform both the gridding and analyses can be reduced and automated further.

In another related example [14], intake port optimization was performed that followed a different automation technique that included the use of a mesh morphing tool. The optimization process was started with a single CFD model. The model was then imported into a mesh-morphing tool, which was used to set how the CFD mesh could be changed. The process of setting up a model like this was referred to as arbitrary shape deformation (ASD). This process saved the time of creating multiple CAD models that were individually meshed and then set up in a CFD model. The set up time for the mesh morphing process was approximately two days performed manually. This process requires checking the model to verify that as the model is deformed to the different configurations, the mesh stays within the needed limits for good numerical results. Similar to the previous example, the cases that were made after the ASD process was finalized were done with a script, and seventy-seven additional CFD models were made. The amount of time it took to make and check these additional “cases took less than 3 man hours in total” [14]. One of the main advantages shown in this example was a product of

using a base CFD model. As a result, the number of iterations after the first were lowered by about 90% because each of the following models was able to use the converged information from the previously run model. The analysis model used was approximately 2.7 million cells for the 124 cases during which more than one 12xCPU PC array was used during run. The total CPU time was just over 169 hours. While neither of these cases is fully automated, the second is far superior in the amount of saved time. The drastic time difference between the two examples is emphasized by the fact that the first used a symmetric model and created 45 cases whereas the second used a full model and created seventy-seven cases. Results for time saved using baseline results is shown by Gleason et al [11], where the time saved from using base line results reduced the overall computation time by 36 hrs.

The most comprehensive method found for an engine airflow optimization was by Gaikwad et al [27]. Their method optimizes two cases of steady state flow for an intake port using an automated process with optimization performed with Isight. Using pre-determined parameters for meshing, a model is updated in the CAD software and then sent to the meshing tool for analysis setup before the job is submitted. After the model has been optimized, it is then sent to a different meshing utility to prepare the model for transient analysis. The process presented included minimal human interaction for the whole process. This process utilized the traditional method and no mesh-deformation software was implemented. The advantage to this process is that both steady state and transient cases are used for the optimization process.

A closely related flow example is given by Hoke and Doroudian [12]. The method presented was an example of how to optimize an instrument panel/console duct. The flow model was manually meshed and then imported into a mesh-morphing utility so that the shape of the ducting to each of the outlets could be optimized. This case involves steady state flow where the



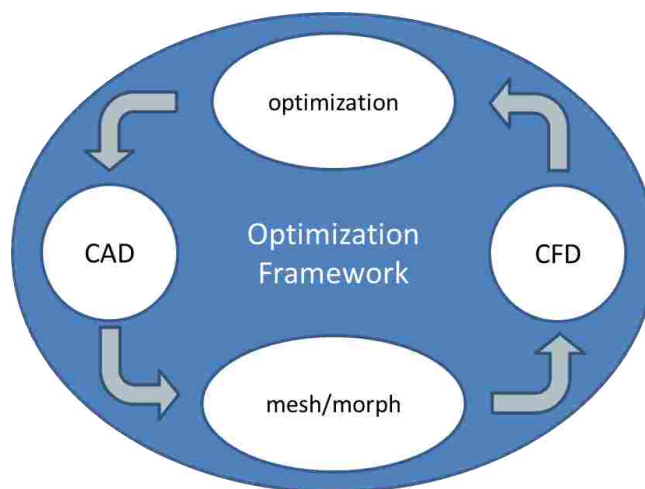
space constraints are very restrictive due to the available space underneath the instrument panel and console. As a result of using an optimization package in conjunction with the mesh-deformation tool, an improvement of 250% in flow through the duct entrance as well as a 9% pressure drop reduction in the console duct were achieved.

An optimization method using CFD to optimize a vehicle shape with mesh morphing describes a CFD tool that has the mesh morphing tool and optimization tool built in [28]. Because of this, the optimization is simplified with the initially meshed model being used as the base for subsequent cases. The geometric changes, design space and optimization method are all contained within the CFD tool to perform the optimization. While this greatly simplifies the process for the setup of the optimization with the analysis model, it does not implement a way to link the final parameter changes to the CAD model. In addition, the design space is redefined within the CFD analysis tool when the space should already be established and set in the CAD model. While this process is an improvement over the traditional method and non-integrated optimization processes, it does not allow for feedback into the CAD package so that parametric modeling may be used to account for the changes made.

### 3 METHOD

In this chapter, the general method for how the design optimization process will be accomplished is presented. Below is a short outline of the steps required to accomplish this method represented by Figure 3-1.

1. Develop parametric flow path model
2. Generate mesh for the flow path model
3. Create mesh morphing model from mesh
4. Perform CFD analysis
5. Optimize flow path by changing mesh geometry



**Figure 3-1: Optimization Framework Overview**

The remainder of this chapter generally describes these steps such that they can be implemented regardless of the software used. Chapter 4 will present a working prototype that shows a specific implementation of the method.

### **3.1 Parametric Flow Path Model**

The first step to performing the optimization process is to create the flow path to be optimized. To fully explore the design space allowed for the flow path, a parametric model will be needed. How this model is developed will determine what geometric parameters will be used to optimize the flow path. Each of these parameters can be identified by a term. Here, these parameters will be referred to generally as  $P_{x_n}$  and identified for each parameter as  $P_{x_1} \dots P_{x_n}$  depending on the number of parameters or constraints that will be used. Here the “x” refers to some parameter group, “P” signifies that it is a parameter, and the subscript “n” identifies each parameter in a group. Each parameter has a base starting value that will be identified by  $P_{x_n}$ . The bounds of each of these constraints will be identified by two other terms for each parameter.  $P_{x_n}^H$  will be the maximum value for a specific parameter while  $P_{x_n}^L$  will be the minimum value allowable.  $P_{x_n}$  will be the base value used for optimization routines.

To prepare the model for flow analysis, other features will need to be assigned to the model. These features may also have accompanying attributes to be associated with a specific feature that can be assigned within the CAD model. To identify the features needed for a flow model, such as an inlet, outlet, wall, or fluid a symbol will be assigned to each of these conditions for downstream setup and analysis. In addition, attributes may be assigned to each of these features along with any other required information. A wall may be assigned an attribute that specifies the type of mesh, mesh size, or fluid conditions such as pressure, velocity or temperature. Here Table 3-1 shows how these will be referred to in the method. When referring

to flow features and attributes generally  $F_{X_n}$  and  $F_{X_{nAj}}$  will be used. Here flow features represent geometry related to flow boundaries or regions. Velocity, pressure, type of fluid, and other similar information will be passed as attributes.

**Table 3-1: Flow Feature Identification**

<b>Flow Features</b>	<b>Identifier</b>	<b>Attributes</b>
inlet	$FI_{1\dots n}$	$FL_{nA1}\dots FL_{nAj}$
outlet	$FO_{1\dots n}$	$FO_{nA1}\dots FO_{nAj}$
symmetry	$FS_{1\dots n}$	$FS_{nA1}\dots FS_{nAj}$
wall	$FW_{1\dots n}$	$FW_{nA1}\dots FW_{nAj}$
fluid	$FF_{1\dots n}$	$FF_{nA1}\dots FF_{nAj}$

Other geometric features from the model may also need to be referenced for operations performed when meshing, other than what will be used for the base mesh model.

Geometric features may be used for the mesh morphing process such as a flow path centerline, surface, or edges that will be used for morphing. Similarly, if an attribute needs to be assigned to a feature in addition to its identification this can also be done in a like manner as shown. Table 3-2: Geometric Feature Identification shows how these will be referred to in the method. Following the same pattern, referring generally to geometric features the term  $G_{X_n}$  will be used.

**Table 3-2: Geometric Feature Identification**

<b>Geometric features</b>	<b>Identifier</b>	<b>Attributes</b>
Centerline	$GC_{1\dots n}$	$GC_{nA1}\dots GC_{nAj}$
Surface	$GS_{1\dots n}$	$GS_{nA1}\dots GS_{nAj}$
Edge	$GE_{1\dots n}$	$GE_{nA1}\dots GE_{nAj}$

While it is additional work in the beginning to create these attributes for the features, it enables the ability to automate other applications in the process without repeating known information.

### **3.2 Flow Path Mesh Generation**

Using flow features ( $F_{X_n}$ ) and attributes ( $F_{X_nA_j}$ ) makes it possible to automate the meshing process. To follow the method described, a meshing tool that allows direct import of the part file from the CAD program and automatic meshing is required. If a tool is used that does not import parts directly, an adjustment will need to be made to this approach using exported geometry from the CAD package.  $F_{X_n}$  and  $F_{X_nA_j}$  will be used to define the mesh. Using these enable the size, shape, type, and other options needed to be stored for use. If the information needed to create the grid is known, that may be stored in  $F_{X_nA_j}$  and used. Otherwise, to determine the mesh parameters, a base analysis should be performed. Once the results of the base analysis are satisfactory, the parameters needed for the mesh creation are known.

After all the needed information has been established, a script or program is then written for the meshing tool using the API. This program will import the flow geometry, features and attributes ( $F_{X_n}$  and  $F_{X_nA_j}$ ) and use that information to mesh the flow path for analysis. At this stage the mesh that has been created can be used for flow analysis.

### **3.3 Morphing Procedure**

To make changes to the analysis model created, a mesh morphing procedure will be used. The analysis model will be morphed to match the shape of the geometry that has been created in the CAD package. Changes to the model are made by using  $P_{X_n}$  that will be used for design optimization.

Here a general process will be described to morph the analysis model to the new geometry. There are different morphing methods used to change the shape of the analysis model to the new shapes needed to find the optimal point. The method used to morph the analysis model must be determined by the user. A requirement is that the morphing package be able to import the geometry needed from a CAD package. As mentioned,  $G_{x_n}$  and  $G_{x_nA_j}$ , which the CAD model has assigned to it, make it possible to programmatically determine details about the geometry. While not used in all types of mesh morphing procedures, they may be needed so that specific control over certain areas of the mesh may be morphed to specific geometric features. In addition, the geometry that is imported must also use attributes so that geometry imported can be appropriately assigned or named for morphing control. Previous iterations or original geometry may need to be imported for distance calculations for the mesh to be morphed to the new shapes. This also needs to be identified programmatically so that the geometry components imported will be kept organized. Once the new geometry has been imported, and the old if needed, then the morphing procedure may be continued.

Given a certain geometric model and analysis model, the difference between the two needs to be eliminated for the shape to be matched. While the specific method for how to accomplish this will vary from one software package to another, the basis is the same. Once the difference between the analysis model and the new model design is eliminated then the shape change is complete. The difference between the two different geometric shapes may be considered as some  $\epsilon_x$ . With the current analysis geometry or feature being represented as  $G_{x_{old}}$  and the new geometry being represented as  $G_{x_{new}}$ . With many different features and changes which need to be made, an equation can express the change from the previous geometry to the new, shown below.

$$\begin{bmatrix} G_{1\_old} \\ \vdots \\ G_{x\_old} \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_x \end{bmatrix} = \begin{bmatrix} G_{1\_new} \\ \vdots \\ G_{x\_new} \end{bmatrix} \quad (3-1)$$

Once the analysis model has been matched to the new geometry, the morphing process has been completed. This enables the same mesh to be utilized while also enabling the shape changes needed for optimization.

This process will need to be adjusted depending on the software package being used. The mesh morphing tool may be integrated with the CFD package and the meshing package, as well as another program, if desired. The exact procedure will vary depending on what is used, but the underlying procedure of associating the CAD model features and then passing those attributes to downstream applications with the needed data is still applicable.

### 3.4 CFD Procedure

To perform the CFD automatically, guidelines must be established for how the model will be analyzed. Once this has been established, a command file may be created which contains the necessary data for how an analysis should be performed. Using the API, the command file can be used to import the analysis model, perform the analysis, and return the results that are needed for the optimization framework. Here this process will be presented in more detail.

Guidelines must be established for the analysis of flow models for an automated process. If previous guidelines have not been established from a knowledge base, an initial analysis will be used to determine the correct size of the grid so that the flow detail is independent of the grid size (grid independence). During the initial analysis, the CFD setup will also be checked to make sure that the solution methods are working correctly. Once the proper grid resolution and CFD setup has been determined, they will be used for subsequent models that will run

automatically. This will enable the automation of similar models while ensuring that the correct solution methods are selected. In addition to the solution method, the monitors desired will need to be determined for the optimization framework. The method for the solution needs to be saved in a file that the solver can then import. This file will have the commands needed to assign the type of solvers, conditions, monitors, etc. that are needed. This file will then be imported to the CFD program that is used for the cases run for the optimization. This process enables the subsequent cases to be performed without the necessity of inputting the needed information at every iteration. As an example, this process may be read in by Fluent when the conditions are laid out in a journal file. When imported, this file defines the needed information for the analysis. This and the case file are then used for the analysis of the model. The first run is a baseline run that will establish the initial model values. Once the analysis is completed, the desired output values are written to file which will be used for the subsequent optimization of the design space.

### **3.5 Optimization**

While there are many details that take place in an optimization, a short summary will be given here of the overall process. Following, more detail will be covered about specific aspects of optimization.

To perform an optimization on a specific model, the optimization framework needs to be setup. The setup includes defining what the design variables are and their associated design constraints, which are  $Px_n$ ,  $Px_n^H$ , and  $Px_n^L$ . The optimization framework works by controlling the CAx programs. To perform the optimization, each analysis program needs to run and provide useable results. As each analysis program runs they write the results specified by the



user in the command file, script or program. The optimization framework then retrieves the results for needed calculations.

The type of algorithm that will be used, as well as any associated settings for that particular algorithm, will need to be set up before the optimization process begins. Then, a base model will be tested. Once the initial model is solved, this information will be passed to the optimization framework. The optimization framework will parse the results file for the needed information regarding the design variables, constraints and results. Using  $P_{X_n}$ ,  $P_{X_n}^H$  and  $P_{X_n}^L$ , the optimization framework will then drive toward the optimal solution using the results from the analyses to both determine how the design model will be changed and finally the optimal  $P_{X_n}$  for the design model.

The type of problem will determine the algorithm to be used within the optimization framework. Gradient-based algorithms are very efficient and will generally be the fastest method to achieve the optimum. Two gradient based algorithms are the Generalized Reduced Gradient (GRG) method and the Sequential Quadratic Programming (SQP) method. A GRG algorithm is very robust and can obtain an optimum within a specific design space, staying within specific constraints. SQP is known to be faster than the GRG method, but does not operate as well within a specific design space, meaning test cases may violate the bounds of the design space during the optimization. This method would not work for a model which will only work within a specific range. This means using this algorithm may not always be the best choice due to limits of an analysis model. A genetic algorithm can also be used for the optimization process. While they are not as efficient as gradient based, genetic algorithms are well suited to a large design space, or multi-objective cases. It is important to consider the type of model, constraints, objectives and number of variables when choosing an optimization algorithm.

A method to run an optimization framework in batch mode is to set up the optimization process in a standard manner, meaning to open up the optimization program and to set up the needed information to complete an optimization process. This optimization can then be saved and run in batch mode, controlling the needed programs. This process is acceptable for optimization cases that do not take significant amounts of time to set up, negating the need for a more automated process of setting up the optimization framework. For complex or large optimization routines, the setup of an optimization framework can take a significant amount of time. Some optimization frameworks can also be programmed using an API similar to the CAD and meshing tools. This enables it to be set up using a file with the commands needed to set up the optimization process (command file) to run in batch mode so that the graphical user interface GUI is not needed and the entire process can be automated. Using a command file for the optimization framework enables one to read in the design variables, constraints, results file, type of optimization used and its associated settings. This process of automatic set up will take advantage of  $Gx_n$  assigned in the original model as well as  $Px_n$  variables and limits. In addition to the attributes assigned to the original model other parameters may be imported that will further guide the optimization process with best known practices when desired. The best known practices referred to would be using a “Knowledge Based” engineering approach. Once the inputs and set up of the optimization framework are completed then the optimization can be performed. Though the development of this command file, to set up the optimization algorithm, will be time-intensive in the beginning, the amount of time saved can be large in comparison to when similar optimization routines are performed.

### **3.6 Summary**

Here a method has been presented that integrates existing CAx tools with an optimization framework, specifically using mesh morphing. Mesh morphing is used in industry currently, both in automated and manual optimization processes. The important difference here is that the method presented demonstrates how after a model has been optimized with a morphing process, the original model does not need to be manually matched to the optimized shape, which is the current practice.

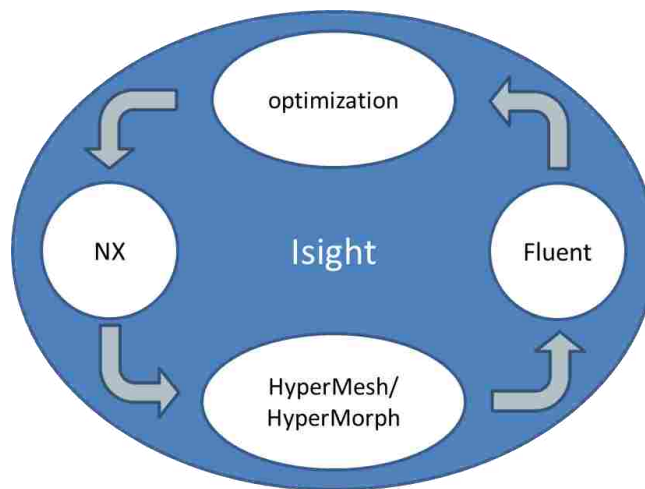
Developing an integrated optimization framework takes a substantial initial time investment, with the amount of time depending on the complexity and detail of the process being developed. The goal of spending the time to develop this process is to greatly reduce the time spent manually performing the process repeatedly for design and optimization. The time spent to develop the process should pay off by saving significant time using the process that has been developed..

## 4 IMPLEMENTATION

This chapter will discuss the implementation of the optimization process developed.

Following is an outline of this process represented by Figure 4-1.

1. Develop parametric flow path model in NX
2. Generate mesh for the flow path model using HyperMesh
3. Create mesh morphing model from mesh using HyperMorph
4. Perform CFD analysis with Fluent
5. Optimize flow path by changing mesh geometry with Isight



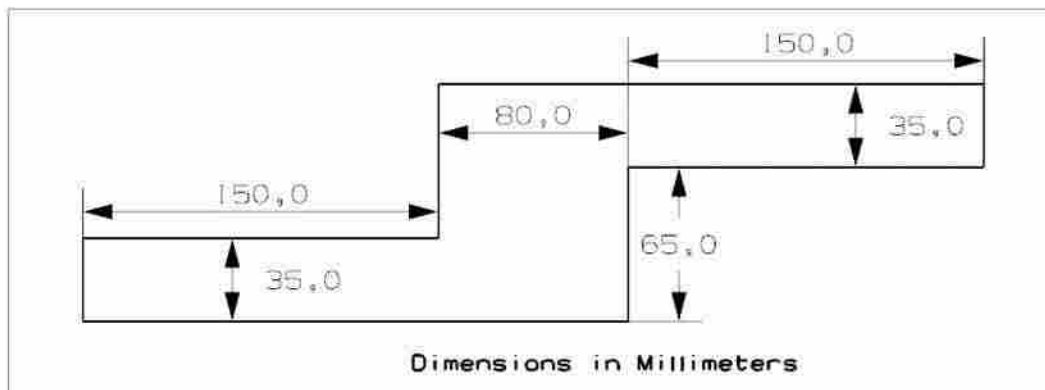
**Figure 4-1: Optimization Framework Implementation**

The optimization process developed used NX 5.0, HyperMesh 11.0, Fluent 12 and Isight 3.5. Following the method outlined in chapter 3, these were implemented in the programs' respective areas.

The flow model presented here is a simple design study to be used in the integration process for an optimization being driven by a CAD model. While this geometry is not complex, the method and implementation shown is generally applicable. The changes possible are only limited by the model used and the capabilities of the software. While there are many different tools that could have been selected, the tools used were chosen primarily because of support from the Partners for the Advancement of Collaborative Engineering Education (PACE) Program and their use by General Motors.

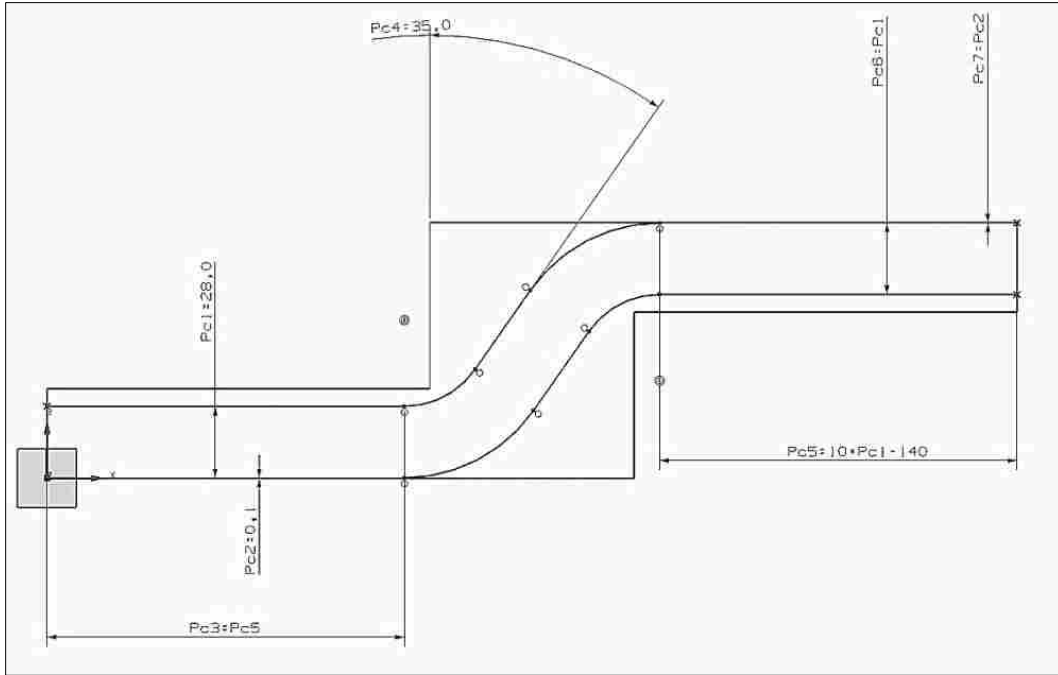
#### 4.1 Parametric Model

To test and demonstrate the capability of the method, an S type shape design space was optimized within the envelope defined by Figure 4-2. The pipe in this space is to be optimized for the highest mass flow and lowest pressure drop from inlet to exit, while changing the dimensions of the parametric part.

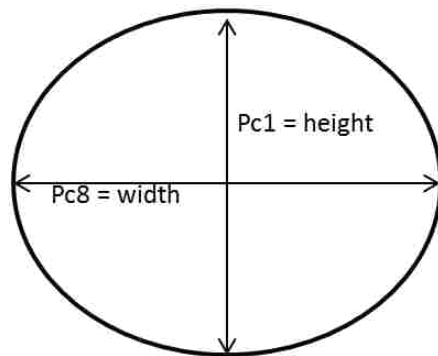


**Figure 4-2: Design Space**

The defining parameters for the possible shapes for the pipe are the center angle ( $Pc_4$ ), entrance/exit length ( $Pc_3$  &  $Pc_5$ ), height ( $Pc_1$ ) and width ( $Pc_8$ ).  $Pc_3$  &  $Pc_5$  are related to  $Pc_1$  by a linear function so the arcs will not go out of bounds on the turns. The parametric setup on the sketch and the dimensions used are illustrated in Figure 4-3 and Figure 4-4.



**Figure 4-3: Control Dimensions and Parameters**



**Figure 4-4: Control Dimensions End View**

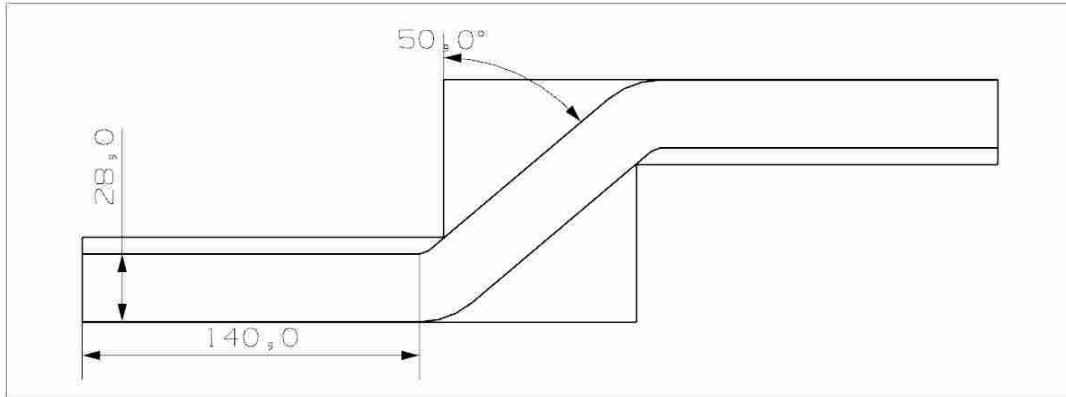
**Table 4-1: Parameter Variables and Limits**

	<b>Lower</b>	<b>Base</b>	<b>Upper</b>
<b>Pc1 &amp; Pc6 (mm)</b>	26	28	28
<b>Pc3 &amp; Pc5 (mm)</b>	120	140	140
<b>Pc4 (angle)</b>	35	45	50
<b>Pc8 (mm)</b>	26	32	32

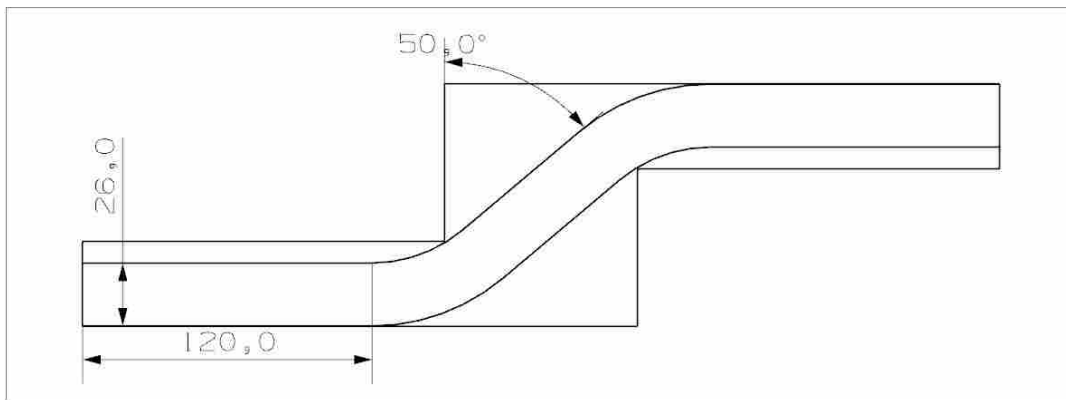
Many combinations are possible with this setup and the dimensions that can be changed. One of the tradeoffs that can be seen from just changing  $Pc_1$  is the smoothness of the pipe bend transition. A larger  $Pc_1$  can be used, but the path does not transition as smoothly. This can be seen in Figure 4-5. Here the maximum angle of 50 degrees is used where the pipe does not cross the boundaries around the turns. Figure 4-5 and Figure 4-6 demonstrate how the linear relation works between  $Pc_3$ ,  $Pc_5$  and  $Pc_1$ , as well as how those dimensions affect how the model can fill the design space. This linear relation is based on formula (4-1).

$$Pc_5 = 10 * Pc_1 - 140 \quad (4-1)$$

This pipe does not represent any particular intake runner; the reason this geometry was created was to optimize flow through a shape with a non-trivial solution. While the method presented may work with any geometry, and the parametric configuration could be much more complex, this was not the focus of the research or method. This was also the reason why  $Pc_3$  equals  $Pc_5$  and  $Pc_1$  equals  $Pc_6$ . This geometry was used to demonstrate the capability of the method that has been developed. For a more complex design such as an intake manifold or port design the amount of possible combinations becomes very large.



**Figure 4-5:  $Pc_1 = 28\text{mm}$**



**Figure 4-6:  $Pc_1 = 26\text{mm}$**

The flow path shape changes were in the ZY plane and the cross sectional shape of the test pipe is constant from beginning to end, which varies from circular to elliptical using  $Pc_1$  and  $Pc_9$ . How these bounds were used in the optimization will be discussed in 4.5.

With a parametric flow model that has specific design limitations, the next process is to have the model prepared so that the remainder of the geometric design process may be automated. To transfer the needed information from the CAD model to pre-processing and analysis, attributes or relations need to be established. This model accomplished this by using layers within NX. Features or parts in an assembly may be moved to a layer or group. Each layer may be assigned additional attributes as well so that the information will be passed on to



downstream processes. For this process, the outer pipe surface, inlet surface, and outlet surface were each moved to a specific layer. In addition to the flow path shape, the centerline of the runner was also moved to a layer. With these defined in the CAD model, the boundary conditions can be easily set when meshing the part and setting up the analysis.

## **4.2 Mesh Generation**

HyperMesh is the meshing tool that has been implemented for the meshing of the CAD model for analysis. Similar to other programs, HyperMesh has two features that greatly simplify the automation of the meshing process and creation of a CFD mesh. As mentioned previously, the CAD model is divided into layers that are used to define the boundary types of the CFD model. The HyperMesh import tool can directly import the part file and separate the imported file according to layers assigned in NX. This functionality was implemented into a TCL script, which is the language used by the HyperMesh API. As the part is imported, the layers are renamed to the boundary type they represent. Here, layers 9-11 represent the inlet, wall and outlet respectively, which in the method were referred to as flow features. There are many layer numbers available to assign within NX. The numbers used for a particular layer can be determined by the user. Layers 9-11 were used to identify the flow features. Other layer numbers may be used to represent other features, groups, or other methods of organization and identification. Table 4-2 demonstrates how the flow features were identified in the process.

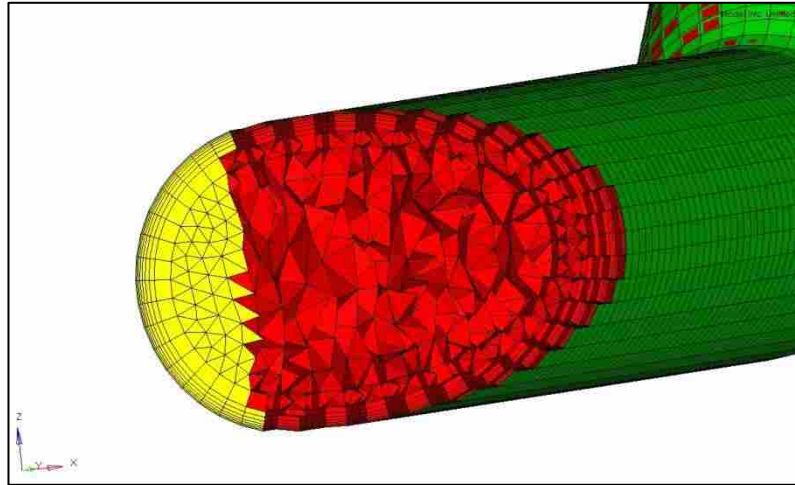
**Table 4-2: Model Flow Features**

Flow Feature (Generic)	Layer (NX)	Name (HyperMesh/Fluent)
FI1	9	inlet
FO1	10	outlet
FW1	11	wall

These components inside of HyperMesh were then used to define how the mesh was created and for identifying those components as needed by the CFD analysis package. For the inlet and outlet, the mesh used a float method for creating the mesh. This means that the boundary layer elements only propagated from the wall and that the inlet and outlet were used as boundaries for the mesh being created from the wall.

The mesh generation for the flow model used a mixed grid along the wall with primarily hexahedral elements and some triangular elements on the surface of the pipe. This enabled the best quality with the shape being used for the base analysis model. From this model, the other models were morphed to create the other test cases. The reason for a grid that is structured is because they are able to handle memory more efficiently and generally are faster to solve [29]. To create the mesh, the surface is first meshed using a mixed mesh (triangle and square elements) and an optimizer for the mesh layout that is built into HyperMesh. Once the surface mesh was created, these elements were then used to generate the interior 3D elements for the model. The Boundary layer grows in thickness from the wall, and the base mesh model has a thin boundary layer that has a growth rate into prism elements. HyperMesh has different options for how the mesh will transition into the interior tetrahedral mesh. In this case a smooth transition was used which consisted of hex cells, pyramid and prism cells that transitioned into

the interior tetrahedral elements. An end view and cut away view can be seen in Figure 4-7 which shows the result of this process.



**Figure 4-7: Mesh Sample**

The size of the outer mesh, ratio of the smooth transition to the tetrahedrals, growth rate of the boundary layer, thickness and size of the inner tetrahedral elements can be set when generating the mesh for the flow model. The sizes used were implemented into the TCL script for the traditional loop and the initial mesh generation. Mesh sizes in hypermesh are dimensionless, this means since the model was created in millimeters the sizes given are in millimeters. The size of the interior and transition elements were interpolated from the wall and boundary layer to the interior using an algorithm which is built into HyperMesh. The settings for this process were set as automatic so the size of the tetrahedral elements and pyramids were determined by the algorithm. The size mesh settings can be seen in Table 4-3.

**Table 4-3: Mesh Type and Size**

	Type	Size
Surface	mixed	2
1st boundary layer thickness	hex	0.09
growth rate (8 cells thick)	hex	1.3
transition elements	pyramid	Interpolated
interior	tetrahedral	Interpolated

To automate the meshing process, the commands needed were all implemented into a file using the TCL language. This file includes importing the geometry, cleaning the geometry, meshing the geometry and then exporting a Fluent case file that can then be used for Fluent. One of the steps which was included in “cleaning” the geometry was to equivalence any edges so that the model had no openings. Depending on the use of the case file after it was created, it may be exported with information from a previous case that has been set up for Fluent, or a case file only containing the mesh and boundary definitions can be exported. This makes it possible to reuse information from setups previously made, or also to use data that has been stored from previous analyses. This functionality and its use will be discussed more in section 4.5.

### **4.3 Mesh Morphing Process**

The mesh morphing process is performed after meshing the part has been completed. Mesh morphing in this study was performed using HyperMorph using the *domains and handles* method. The Fluent case file, containing the mesh created, was imported by HyperMesh to use for the morphing process. To morph the model, different domains needed to be created that correlated to the mesh. When the mesh is created for Fluent, it is organized to correlate to the solver with the associated boundary identities, such as wall, inlet and outlet. While there are other possibilities that are used in a CFD solver, the components created in this mesh are the

inlet, outlet, wall and fluid. When the model is imported, a morphing domain is assigned to each of the components; one group for the wall, another for the inlet, outlet and fluid respectively. Assigning the 2D components individually and then creating the 3D domain for fluid was the most robust method found during this study. There were two primary reasons why each component was assigned individually. First, was to control the domain and handle creation. When each component had a morph domain assigned to it, this prevented extra handles from being automatically assigned to a morphing domain that were not needed. Second, assigning each component in a specified order provided a way to keep track of the morphing domains. Once the domains were created they were reassigned new numbers in a specified order so that each domain could be readily identified or used in the script. These domains are identified by numbers, so specifying the order and starting point enabled identification for use later in the code. This allows morphing to be performed on specific domains of the mesh in groups that represent the conditions in the mesh.

Once the groups were created, the original flow centerline, the new centerline and the new surface were imported. The original centerline (Gc1) was imported from the base part file and the new centerline (Gc2) and surface (Gs1) were imported from the new part file with the updated dimensions to HyperMorph using a layering scheme in the CAD model to identify the geometries so they may be named accordingly. The geometric features, associated layers and names used are seen in Table 4-4.

**Table 4-4: Model Geometric Features**

Geometric Feature (Generic)	Layer (NX)	Name (HyperMesh)
Gc1	2	old_line
Gc2	2	new_line
Gs1	3	new_surface

Using the *map to geometry* function in HyperMorph, the mesh from the case file was morphed to the flow path line using the line difference method. The old line and new line are used to calculate the change that needs to be made for the flow path of the model. To morph to the surface robustly, a two-step process needs to be followed. If this process was not followed the wall of the mesh would morph to the correct shape, but the entrance and exit of the pipe would only partially morph, resulting in an unacceptable model. First, the nodes on the edges of the pipe were morphed to the edge of the surface. Following this, the nodes from the outside of the geometry (in the wall domain) were then morphed to the surface that was imported. This process created the geometry changes desired in the analysis model. Once the morphing process was complete, the model was then exported to a case file for Fluent to be used for analysis.

To make sure that over time the mesh would not become deformed by the many morphing operations performed on the same analysis mesh, the original mesh created was used for each shape change. During initial testing it was found that if changes too large for the model were made, that HyperMorph would simply state that the change is too large and to try a smaller operation. In HyperMorph the node and element numbers would remain the same throughout the morphing procedures performed during the optimization routine. Because these remained the same, Fluent will see a similar mesh so that the analysis can be started from the previous run.

This enables the model to converge faster because the model can start from where the previous analysis stopped.

#### 4.4 Flow Model Analysis

For the flow analysis, Fluent was used as the CFD solver. To optimize the flow through the pipe a velocity inlet case was used with an *outflow* exit boundary type.. While a pressure drop case is typically what is used in industry, t a velocity inlet setup was chosen to simplify setup and reduce the time to convergence. The *outflow* boundary condition is a special case where the exit conditions are calculated based on the flow through the pipe. To use this boundary condition, the flow needs to be incompressible, single phase and steady state. To satisfy these conditions air was single phase, the Mach number was 0.04 and the flow study was steady state. Here the focus was to demonstrate how the morphing process would be implemented in a flow optimization, so a pressure drop case was not used. Outlined in Table 4-5 are the boundary conditions used for the flow model in the optimization.

**Table 4-5: Flow Boundary Conditions**

inlet	10 m/s
outlet	Outflow
fluid	Air
ambient pressure	101,325 Pa
temperature	300 K
Wall	stationary, no slip, 0.5 roughness constant, adiabatic

To perform the optimization, a model needed to be created which would return good results but would not be too computationally expensive. A grid independence study was performed to verify the size of mesh that would result in consistent solutions for the pressure

drop and the mass flow through the initial pipe design. The results of the grid independence tests can be seen in Table 4-6 and Table 4-7.

**Table 4-6: Grid Independence Mesh Size and Flow Results**

surface grid size (mm)	cells	faces	nodes	Pressure Drop (Pa)	mass-flow (kg/s)
3	63512	149338	28699	50.58313	0.006446654
2	162874	376152	67980	51.33856	0.006479774
1.5	302013	691865	121479	51.32079	0.006490538

**Table 4-7: Grid Independence Results Comparison**

surface grid size (mm)	Pressure Drop	Mass Flow
3 mm to 2 mm	1.471%	0.511%
2 mm to 1.5 mm	-0.035%	0.166%
3 mm to 1.5 mm	1.437%	0.676%

Because of the small difference between an initial surface size of 2mm and 1.5 mm, it was decided due to mesh generation time for the traditional loop and the computation time that 2 mm would be used for the study. Once the size of the mesh was decided, the type of mesh used consisted of a structured boundary layer along the wall of the pipe, and then a smooth transitional mesh using pyramid elements. The remainder of the elements were tetrahedral. To make sure that the mesh size and type would work, the quality of the mesh after morphing was checked using the outer bounds of the design space to check for abnormal or poorly shaped elements. For this process the checks were performed manually, though to perform this automatically would consist of writing a script and implementing it into the process. The quality checks of the mesh were performed manually with the mesh checking utility within Fluent. Performing this study ensured that the model would have good quality within the bounds tested during the optimization.



A realizable k epsilon (RKE) turbulence model was used for the turbulence in the solving method. This model was found to be the most robust and reliable in a previous study for internal flow optimization by Mistreanu [30]. To optimize the flow through the pipe there were two objectives, one was to reduce the pressure drop from the inlet to the outlet, and the other was to maximize the flow of the pipe. These criteria were measured after each run and output from Fluent. To establish a good flow model, multiple runs were performed on the base model for the flow path. The RKE turbulence model was used during these tests. It was found while performing the analysis of the base shape that there was reversed flow in some of the elements, which was incorrect. Because of this, each new shape analysis was started as a laminar case and then, once this converged within the assigned range, a first order turbulence model was used. After the turbulence model converged with first order equations, second order equations were used for the final part of the analysis. Once this reached a prescribed residual convergence, the flow analysis was completed for a geometric design and the results were then written to a text file to be read by the optimization software. The details for the flow analysis setup can be seen in Table 4-8. This setup was used for every flow test case during the optimization routines, including each of the morphing cases.

**Table 4-8: Flow Solver Setup**

<b>Pressure Velocity Coupling Scheme</b>	SIMPLE
	realizable K-epsilon (2 eqn)
	standard wall functions
<b>Gradient Solver</b>	Least Square Cell Based
<b>Second Order Solvers</b>	Pressure
	Momentum
	Turbulent Kinetic Energy
	Turbulent Dissipation Rate
	Energy
<b>Solution Method</b>	Laminar using 1st order (300 iterations)
	Turbulent using 1st order (300 iterations)
	Turbulent using 2nd order (800 iterations)
	Convergence residual check k and epsilon 5e-7

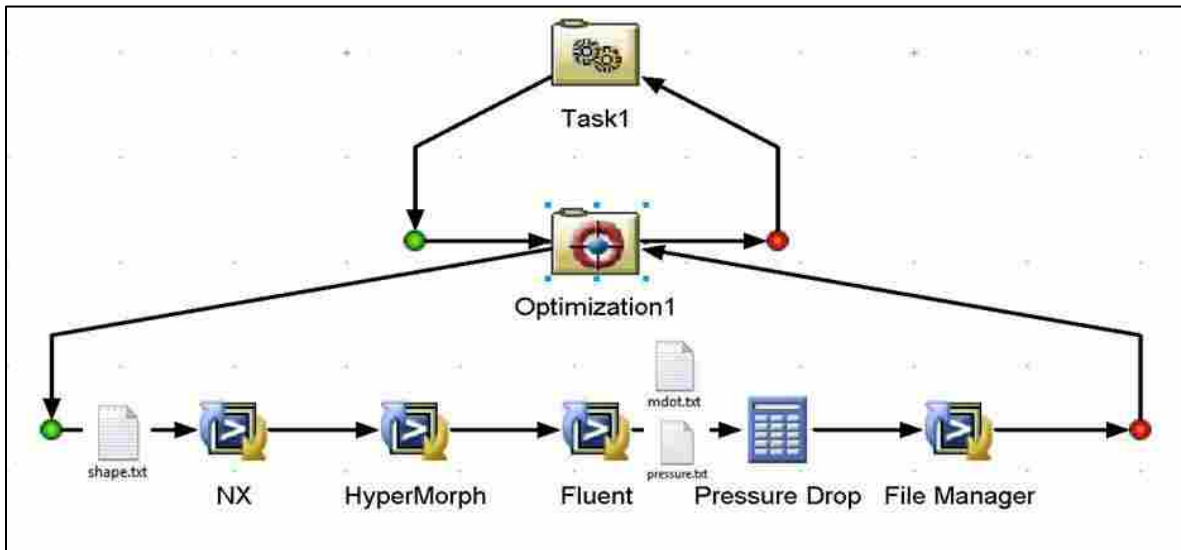
To automate the flow process, a journal file was created for Fluent that contained the operations needed to be performed within Fluent for the optimization. For the morphing process, the first case read in would use the results of the test case, providing the data from that model. The boundary conditions would then be read in for the laminar case. Following convergence, the process would continue similarly with the following two cases. This generated good results with no errors during the flow analysis. To perform the analysis for the standard mesh loop optimization, the Fluent journal file was written to read in the case file that contained the mesh with the boundary definitions. The journal file would then read in the boundary conditions, initialize and perform the analysis in the same manner as the morphing process. The main difference between these methods in CFD analysis was how the flow analysis was started. Each morphed case would iterate from the previous solution. For the meshing optimization routine, each run would begin with an initialization and then perform the analysis with the same convergence criteria. The comparative differences and detail on the optimization routines will be discussed further in chapter 5.

## 4.5 Optimization Framework

The goal of this process is the optimization of the flow path through integrating the CAx programs. This was accomplished using Isight as the optimization framework for the process. How Isight was used to perform this optimization will be presented here.

Isight was used to integrate NX, HyperMesh, HyperMorph and Fluent. These programs are used by Isight to generate the designs and perform the analysis needed for the optimization algorithm being used. Isight determines the values of the parameters that need to be tested, and then as the output is generated from the programs being run, will find the optimum based on the results output. Isight is a text based optimization framework, meaning that the programs being used by Isight communicate via text files. Isight will write values to a text file, and then the text file is parsed and the contents used for each optimization run by the programs which need that information. This means that the programs, which need to communicate with Isight, will need text parsing functionality. As an example, and to explain its implementation in this optimization loop, NX was used to make the geometric changes, be it for the meshing or morphing optimization loop process. To begin, initial sets of values, from the first iteration of the optimization loop, are written to a text file. In this case for the optimization loop, there were three variables written to this file. Those variables included pipe width, pipe height, and the angle of the center of the pipe to the vertical wall. These variables have constraints within the design space that are established in Isight. A program was written with the C++ API for NX that would parse the text file for the dimensions to be changed. NX would then generate a new part and save this part under a different name from the original. The optimization loop would then continue with the other programs in the optimization loop. HyperMesh only needed the part from the newly created part file, so in this case a text file was not parsed for this component.

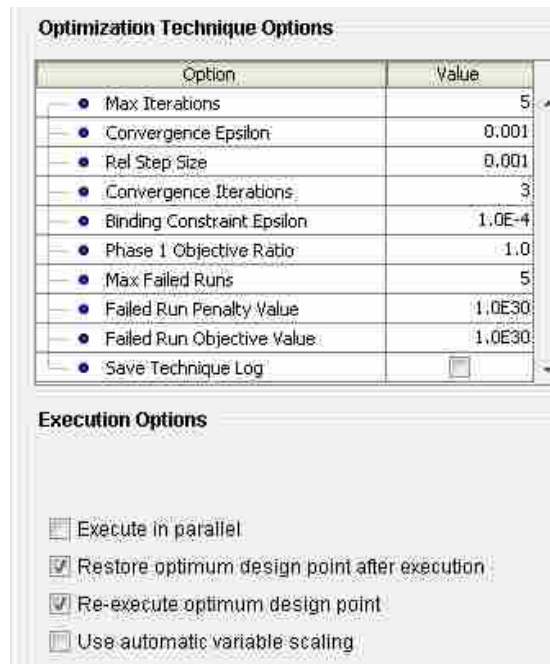
Fluent also had no need of the initial input file because the geometry changes being made would come from the new or adapted case file. For the results, a report was generated that would write to two different text files from Fluent. One would output the mass flow rate of the pipe while another would output the static pressure at the inlet and outlet of the pipe. The text files created by Fluent would then be parsed for the mass flow rate and pressure. The mass flow rate would be used directly, and the pressure results from the inlet and outlet were used to determine the pressure drop. These results were used as inputs for Isight that would be used to find the optimum. This process layout can be seen in Figure 4-8. The diagram shown is the loop from Isight. Note the text file icons shown before the NX component and after the Fluent component show where in the process text files are used.



**Figure 4-8: Optimization Loop Example**

A Large Scale Generalized Reduced Gradient (LSGRG) algorithm was used for the optimization process. There are two primary reasons why this algorithm was selected. First, this optimization was selected because gradient-based optimization methods converge to the

optimum in the least amount of time compared to other methods. Second, this algorithm was used because of the constraints on the design space; it can optimize with constraints and still achieve the optimum. To ensure that the global optimum was achieved, Isight would not only find an optimum, but continue with analysis tests exploring the complete design space using the method specified by the algorithm. As a result, more analysis runs were performed past the optimum found, but this ensured that a global, and not a local optimum was found. These characteristics make it more suitable than others that could be used. For a design space which was less sensitive to constraints that would not present problems outside of the immediate design space, the SQP algorithm would have been used. For a discontinuous design space where there are many variables and constraints, a genetic algorithm would be implemented to find an optimal solution. The settings used for the LSGRG algorithm in Isight can be seen in Figure 4-9.



**Figure 4-9: Optimization Settings**

To perform the optimization, different programs and scripts are called or run from within Isight. For components not part of Isight, such as Fluent and HyperMesh, the program is started from a batch file in a Windows operating system. Using a batch file enables the use of commands or input files that will be used for the specific program, executable programs can also be run from within Isight. In the optimization loops used in this case, the NX component was written as an executable. This was called each iteration of the optimization process. The location of the file to be parsed and the commands needed to change the part were programmed directly into this executable file. Details that will be given here will pertain mainly to the Windows operating system. Specifically, Windows 7 Enterprise 64 bit was used for the analysis performed for this optimization process. The calculator component that is built into Isight was also used. This component was used to calculate the pressure drop between the inlet and the outlet of the pipe. A parser built into Isight was used to read the results from the pressure output text file created by Fluent, and then the difference of the inlet and outlet static pressures was assigned to a variable. This pressure drop variable and the mass flow rate were checked after each optimization loop to drive the steps the optimization algorithm would use for subsequent runs.

During the optimization many different case files were created that were run in Fluent for the optimization routine. In each of the optimization loops the case and data files for Fluent were written over each loop. While this would take up only as much space as was needed for each optimization run, this did not provide a way for the analysis runs to be checked or analyzed after completion. To keep a record of each run, a batch file was written which would copy the files needed and name them as specified after each analysis. This made it possible to check the analysis runs for diagnosis and verification.

To be able to speed up the analysis runs, a high performance computer (HPC) from the college of engineering was utilized. When the HPC was utilized to run the Fluent analysis, this made it possible to free up local resources for additional optimization runs. Because HyperWorks 11 was not yet available on the HPC, the optimization process was controlled locally where the mesh and mesh morphing were performed. To use the HPC, PuTTY was used to automate the submission process of the flow analysis. PuTTY is a terminal emulator that made it possible to login to the Linux server to submit the Fluent analysis. To automate this process, a batch file was written which would execute PuTTY, log in to the remote server and submit the needed files so the flow analysis could be performed. In this case, a text file was written which contained the commands for PuTTY to execute. A command option when starting PuTTY can be called which will read a text file with the commands to be followed. This file included the log in process needed. After the log in, a .pbs file was submitted to the server that would queue the analysis. The .pbs file included the information that was needed to run Fluent, the license file to be used, number of processors, nodes, and amount of memory that needed to be allocated for the job. Using a terminal emulator in this manner makes it possible to use remote resources while also running programs locally. The ability to use this method in the optimization loop came from instructing Isight to look for the results files written to a file server. In this case, a remote storage location was used by both the local machine and the HPC. For the Fluent analysis, an option was set within Isight which would look for a file that would be the signal for the process being completed. After this file was found, Isight would continue the optimization routine.

The integration of the programs with Isight made it possible to automate the complete optimization process. Many different optimization frameworks are available to perform similar optimization routines that can significantly reduce the time to an optimal design when used

effectively. Using an optimization framework takes an initial time investment, but can help greatly reduce the amount of manual work greatly by automating the process of design iterations and using proven algorithms to drive to the optimal solution.

#### **4.5.1 Optimization Using Traditional Method**

To perform a comparison between the different ways a flow path could be optimized, there were two different optimization loops made. Each of these loops optimized the same design with the same design space limitations. The difference between them lies in how the geometry of the mesh was changed for each flow analysis. The standard or traditional method modifies the CAD model, meshes the flow path, performs the CFD and then reports the results to the optimization framework, which then carries on the optimization process with the same loop. This method is still used frequently in industry even without an optimization framework to reach the optimum. The designer changes the model and then remeshes the part and performs the analysis. This process takes place either with the same person, or with multiple people. The development of an optimization framework for this process has a great advantage over this traditional process. Using this traditional method with an optimization framework can greatly speed up the time to drive to an optimum. Because the traditional process is more commonly used, an optimization process was set up using this methodology. This provided a base for how much time it would take with the example used to determine an optimum for the design space.

#### **4.5.2 Optimization Using Morphing Method**

The major difference between the traditional and mesh morphing methods is how the analysis model geometry is changed. As mentioned in the traditional method, new geometry is



generated and then meshed. The method to change the geometry for the analysis model in this loop was mesh morphing. Details of this process have previously been covered in section 4.3.

There are different kinds of mesh morphing that can be used, and how the mesh shape is changed. A mesh can be set up so that various changes can be made to the mesh, and then those results can be input into an optimization framework so the effect of those shapes can be analyzed and the optimum found. A process used with morphing saves the changes made to the mesh during the morphing process from  $P_{X_n}^H$  to  $P_{X_n}^L$ . This makes it possible to change the shape of the mesh very quickly because the calculations have already been made. Different shape combinations can also be saved that correlate to  $P_{X_n}$  or to the mesh and design space. These are sometimes referred to as arbitrary shape changes. While this has the advantage of being able to analyze the design space freely with the possible shapes that can be used in optimization, the optimal model may not match what is in the CAD model, so the CAD model would need to be modified or recreated to match the shape of the optimal flow path. A different approach was used to eliminate this problem in this analysis as explained in section 4.3. While the method implemented does not take advantage of using pre-calculated shapes mentioned earlier, it ensures that the changes made to the model will always be CAD based, so that when the optimum is achieved the variables found to be optimum can be directly transferred back to the CAD model without recreation of the optimal model. This eliminates the need to go back and redesign the model because the parameters that have been found can be readily input to the parametric model previously created.

## 5 RESULTS

Here the results from the different optimization loops will be presented. For the results that will be presented, the optimization loops were performed on a computer with the following specifications found in Table 5-1. For the processing, Isight and Hypermesh/HyperMorph utilized a single core and Fluent used four. Each optimization loop will be presented separately and then the results will be compared between them. The optimization runs were performed with the same mesh parameters, and each model started at the same point in the optimization.

**Table 5-1: Computer Specs.**

<b>OS</b>	Windows 7 64 bit	
<b>Processor</b>		
	<b>type</b>	Intel Xeon W3520
	<b>cores</b>	4
	<b>speed</b>	2.67 GHz
<b>RAM</b>		
	<b>amount</b>	12 GB
	<b>speed</b>	DDR3 PC3-8500

### 5.1 Traditional Method Loop

The traditional method took approximately 27.5 hours and performed 53 iterations to determine the optimal flow path configuration. This resulted in each iteration taking approximately 0.52 hrs. In Table 5-2 is the comparison of the initial to the optimal case found by Isight. For the optimization, the goal was to maximize the mass flow rate while minimizing the

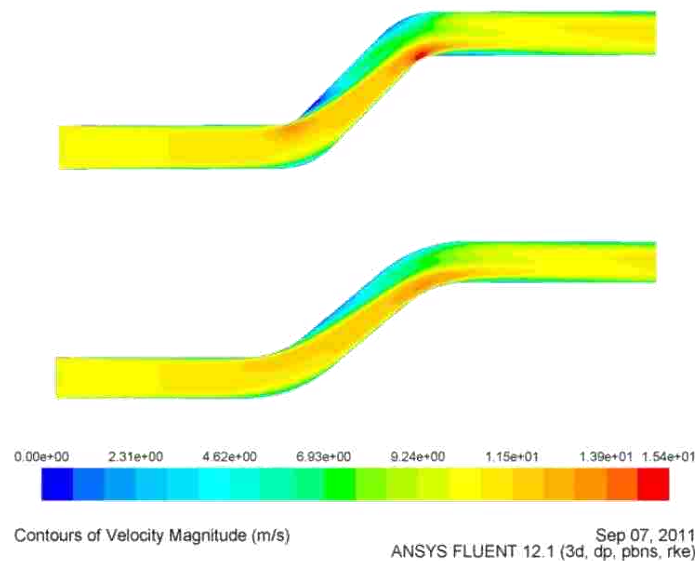
pressure drop each having an equal weight for the optimization. This configuration for an optimization could easily be changed to meet a specified pressure drop and maximize the mass flow rate, or any desired combination.

**Table 5-2: Traditional Optimization Results**

	<b>Angle (deg)</b>	<b>Height (mm)</b>	<b>Width (mm)</b>	<b>Mass Flow (kg/s)</b>	<b>Pressure Drop (Pa)</b>
<b>Starting Point</b>	45	28	32	0.008596	61.170
<b>Optimal Point</b>	43.712	26	32	0.007981	57.112
<b>Percent Difference</b>	-2.95%	-7.69%	0.00%	-7.71%	-7.11%

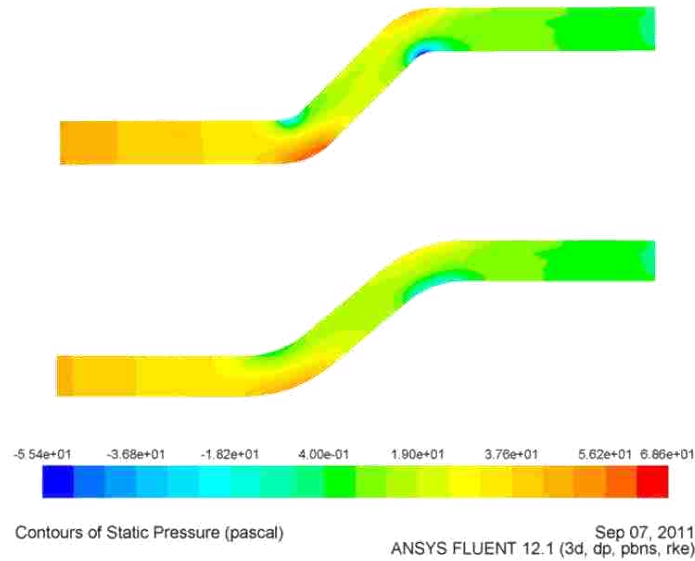
Both the pressure drop improvement and the decline in mass flow changed a similar amount in opposite directions when considering one improved and the other did not. The result of the mass flow decreasing was not expected because each had an equal weight for the optimization. After checking the optimization settings it was found that the mass flow and pressure drop were not scaled correctly. Because the scale of the mass flow rate was much smaller than the pressure drop, this resulted in the pressure drop taking precedence over the mass flow rate in the optimization routine. Taking into account the scaling issue, it was solid improvement for the pressure drop at just over 7 percent. While this optimization did take 27.5 hours, it took significantly less time than this process would have if performed manually by either an individual or group of engineers. For the contour figures which will be shown, a center plane was used that would show the middle of the flow path. While this will not show the complete picture of what is happening, it is indicative of the changes made in the flow path. In Figure 5-1 the upper pipe is the initial test case and the lower is the optimized center plane of the optimized geometry. The improvement in the flow pattern for the velocity can be seen because the areas of low velocity were reduced, as seen at and around the turns of the pipe. Flow

velocity in the optimized pipe also improved because the velocity became more uniform with lower gradients. There is still an area in the center section of the pipe that has low velocity that may be able to be improved upon. This could be accomplished with a model that has a different parametric configuration that would enable greater movement of the flow path. A possible method to accomplish this would be to have a parametric surface that has more degrees of freedom to be adjusted. This would be similar to an arbitrary shape deformation process mentioned previously.



**Figure 5-1: Traditional Velocity Contour Comparison**

Static gauge pressure was used to show the pressure changes in the pipe so that the velocity component of dynamic pressure would not dominate the pattern and it would be easier to see where pressure drops would occur in steps. Total pressure should have been used for the pressure drop calculations, but static pressure was used. In Figure 5-2 pressure contours improved from the initial geometry in the top diagram compared to the lower diagram due to greater uniformity and lower gradients, especially around the bends of the flow path. Similarly this may be able to be improved further with a different parametric control scheme.



**Figure 5-2: Traditional Static Pressure Contour Comparison**

## 5.2 Morph Method Loop

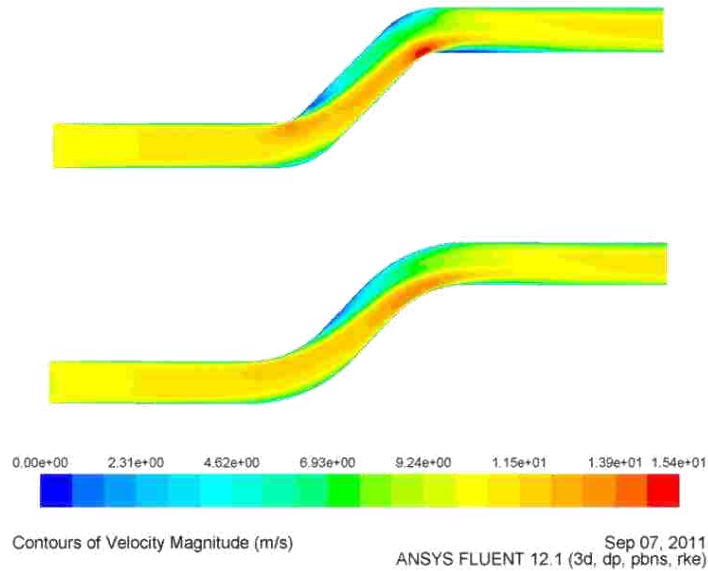
The morph method loop took approximately 9.16 hours and 39 iterations to determine the optimal point, at 0.23 hours per iteration. In Table 5-3 the beginning and optimal points are presented from the optimization performed using the mesh morphing process.

**Table 5-3: Morph Optimization Results**

	Angle (deg)	Height (mm)	Width (mm)	Mass Flow (kg/s)	Pressure Drop (Pa)
<b>Starting Point</b>	45	28	32	0.008596	59.556
<b>Optimal Point</b>	43.48808037	26.269	32	0.008053	55.495
<b>Percent Difference</b>	-3.48%	-6.59%	0.00%	-6.75%	-7.32%

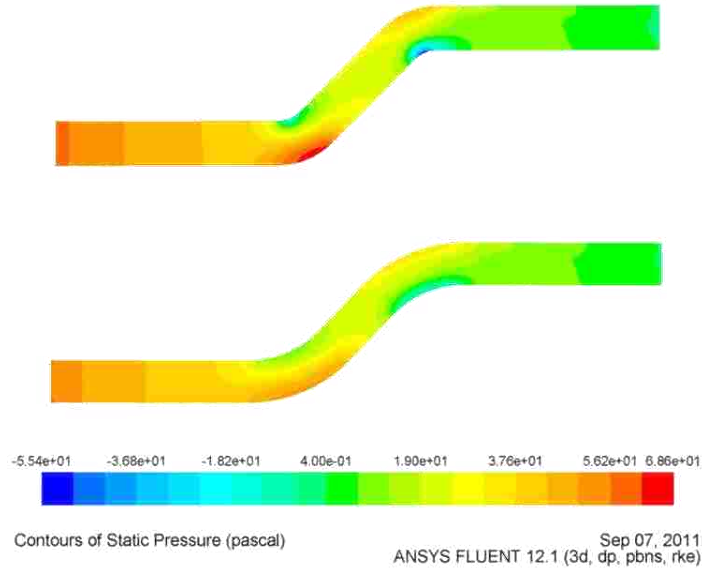
Table 5-3 shows the improvement in the pressure drop was about 7 percent while the mass flow decreased nearly the same percentage at 6.75 percent. The reason for the reduction in mass flow was a result of improper scaling in the optimization. In Figure 5-3, the differences in

the starting and optimal points can be seen in the velocity profile. Flow in the optimized flow path has a smoother flow path through the center section and more uniform flow versus the starting point. The areas of low velocity were reduced, the flow became more uniform around the turns and the gradients were also reduced around the turns resulting in the improvements made.



**Figure 5-3: Morphing Velocity Contour Comparison**

The static pressure drop difference can also be seen in Figure 5-4. This demonstrates the improvement in pressure that was obtained from the optimization, which was a result of the greater uniformity in the pressure, specifically around the turns where the gradients were reduced.



**Figure 5-4: Morphing Static Pressure Comparison**

### 5.3 Traditional and Morph Loop Comparison

Table 5-4 is the comparison of the traditional and morphing optimization loops. The difference between the optimal geometry and flow property results reached is very similar.

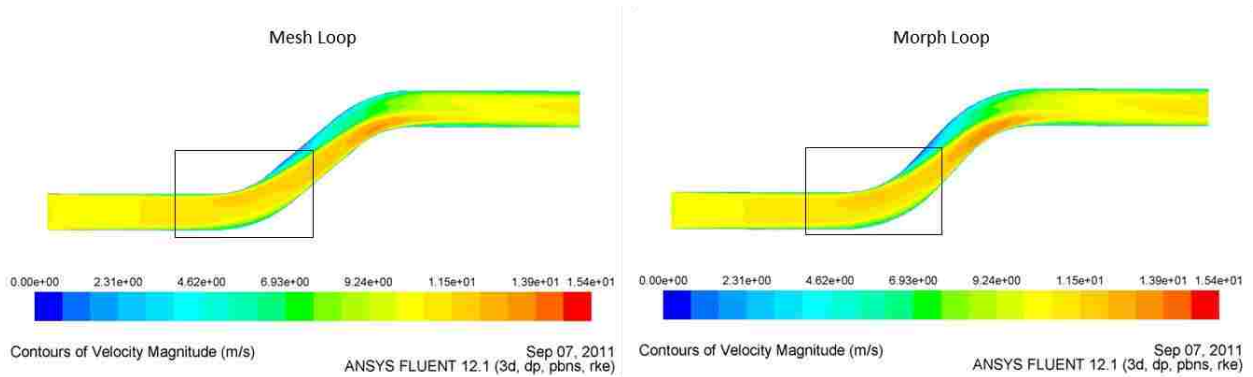
**Table 5-4: Loop Comparison**

	Angle (deg)	Height (mm)	Width (mm)	Mass Flow (kg/s)	Pressure Drop (Pa)	Optimization Time (hrs)	Iterations	hrs. /iteration
<b>Morph</b>	43.49	26.27	32	0.00805	55.49	9.17	39	0.24
<b>Traditional</b>	43.71	26	32	0.00798	57.11	27.50	53	0.52
<b>Percent Difference</b>	0.51%	1.04%	0.00%	0.90%	2.83%	67%	26%	55%

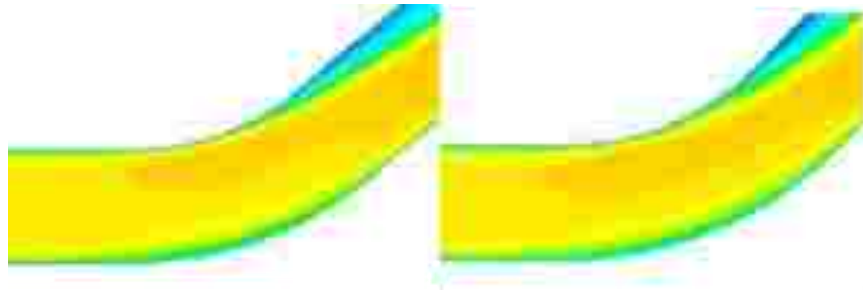
The greatest difference in flow characteristics is the pressure drop (2.83%) and the largest geometric difference is the height (1.04%). Because the morphed model has the same amount of elements from the smallest to largest case, this may account for the difference between geometric

and flow results. The largest reduction of the two different optimization loops, accounting for the difference between the number of iterations, was the hours per iteration at 55%. This represents the reduction in time for the complete optimization loop, including mesh generation as well as the flow analysis time. While this result is not as drastic as has been seen in other cases such as Hosford [17], an advantage of this method is that the CAD model is already at the optimal shape, so there is no need to go back and match the CAD to the optimal result, eliminating the time needed for manual redesign.

Figure 5-5 is a comparison of the different velocity flow patterns of the analysis for the optimal geometry for each optimization loop type.



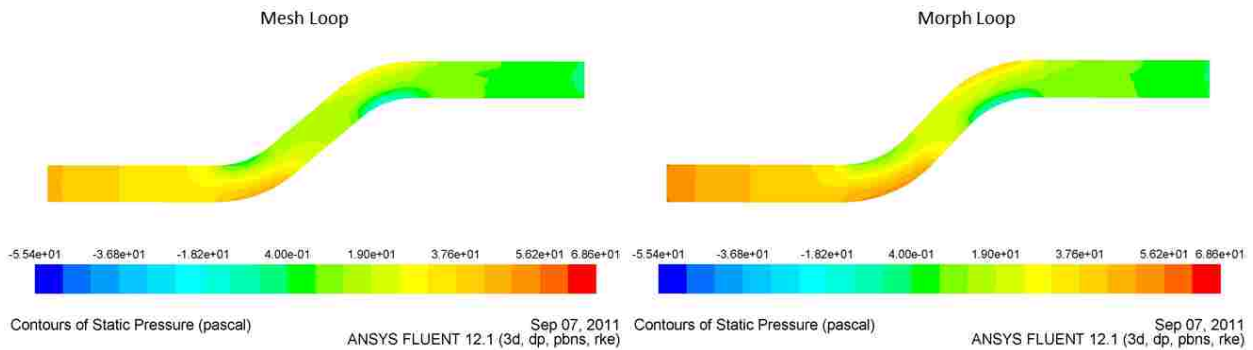
**Figure 5-5: Optimal Velocity Comparison**



**Figure 5-6: Elbow Velocity Profile**



Based on the comparison between the different velocity profiles, there is very little difference between the two. A small difference between them can be seen by the size of the velocity gradient at the first turn on the inside, which shows a larger area of higher speed flow shown in Figure 5-6. Generally the results agree closely with small differences in the contours.



**Figure 5-7: Pressure Profile Comparison**

Similar to the differences between the velocity profile, small differences can be seen in the size of the pressure gradient changes in Figure 5-7 as well as some differences in the pressures in the entry area. While there is this discrepancy, the overall patterns agree very well, and the static pressure drop difference between the two is only 2.83%.

An explanation for the differences between the two optimization loops is the grid resolution at different geometric configurations. The grid has the same number of elements for the morph case regardless of the change in the geometry. This accounts for part of the calculation speed increase between the flow analyses. This also may be the reason for the small differences in the flow profiles. Overall, this difference does not greatly affect the results. While the difference is not as significant as some, greater improvement could be made with the implementation of pre-calculating the mesh shape changes. This would greatly reduce the

amount of time that was needed for the optimization loops to perform the shape change. With the current morphing method, to change the shape each time takes approximately 2 minutes. This time could likely be reduced down to few seconds, which would increase the time savings by that amount each iteration. In addition to this improvement, the analysis could also be sent to a HPC where the analysis time for the fluid flow could be significantly reduced.



## 6 CONCLUSIONS

The process of automating the optimization of a flow path through the use of mesh morphing mapped back to the original CAD part/model has been presented, i.e. the resulting optimal discretized flow path's geometric parameters are mapped back to the original CAD model. This process has been shown to reduce the per-iteration time by 55%. A result of this process is the elimination of the need to match optimal geometry created outside of the CAD package.

In summary, the method presented uses an optimization framework to find the optimal flow path for an intake runner. CAD, a meshing/morphing tool and CFD were used together to find the optimum within a specified design space. The objectives of this thesis were as follows.

1. Using an optimization framework, integrate a pipe flow analysis process from CAD, to a meshing/morphing utility, to a CFD package.
2. Determine an effective way to optimize the shape of the intake runner and have optimization performed in the loop with full or partial automation as desired by the designer.

Objective one was completed by demonstrating the implementation of an optimization framework. This made it possible to automate the design and optimization of a flow path. The second objective was achieved by using a parametric model from a CAD package, and using  $Px_n$  of that model in the optimization framework. Through the use of the optimization framework a

designer can determine the best starting point for the optimization. The use of the optimization framework and other programs also may be completely automated through the use of a batch file, program or script, or they may be used interactively to the degree desired by the designer.

The method and the implementation thereof have shown that with the example shown, savings of 55% per iteration are possible. Developed further, this process will have even greater advantages when used to optimize an intake manifold or flow path network which are much larger.. Using this method, the time to repeatedly mesh an analysis part can be eliminated. In the example given, the time to mesh a flow model would take about a minute to complete. Time spent redesigning a flow path for differing applications and changes can be completely eliminated, making it possible to drastically increase the amount of time spent on new development and research. For the current example, the time it would take to match the optimal shape is minimal, but for much more complex flow paths, the time to match a complex optimized model could be substantial. . These advantages will result in a large reduction in the time needed for flow path development, here it has been shown that the time to find the optimum was reduced by over 50%.

## **6.1 Future Work**

As mentioned previously, this work is the basis for an ideal manifold development tool. While this thesis has presented a base for intake flow optimization there are many areas that could be developed further to work toward an ideal intake development and optimization tool.

1. Develop a method and tool for automated geometry creation based on a given design space.
2. Implement the optimization process with a complete intake manifold and intake system.
3. Determine how to link CAD parameters directly to geometric control for shape optimization in a morphing tool
4. Implement unsteady analysis into the optimization process
5. Develop user interfaces for each tool within the CAD package for complete automation from CAD.

Work in each of these would further enhance the usefulness of the preliminary tool presented in this thesis. A comprehensive intake manifold tool would significantly reduce initial design time while radically reducing the time in redesigning the manifold for a similar application but different design envelope.



## REFERENCES

- [1] J. B. Heywood, *Internal Combustion Engine Fundamentals*. New York: McGraw Hill Publishing Co, 1988, pp. 215-220.
- [2] C. R. Ferguson and A. T. Kirkpatrick, *Internal Combustion Engines: Applied Thermosciences*, 2nd ed. New York: John Wiley & Sons, Inc., 2001.
- [3] J. J. Shah and M. Mantyla, *Parametric and Feature-Based CAD/CAM Concepts, Techniques, and Applications*. New York: John Wiley & Sons, Inc., 1995, pp. 78-79.
- [4] I. Zeid, *Mastering CAD/CAM*. New York: McGraw Hill Publishing Co, 2005, p. 424.
- [5] C. M. Hoffmann, "Constraint-Based Computer-Aided Design," *Journal of Computing and Information Science in Engineering*, vol. 5, no. 3, p. 182, 2005.
- [6] C. M. Hoffmann and K. J. Kim, "Towards Valid Parametric CAD Models," *Computer-Aided Design*, vol. 33, no. 1, pp. 81-90, Jan. 2001.
- [7] D. Orenstein, "QuickStudy: Application Programming Interface (API)," *Computerworld*, 2000. [Online]. Available: [http://www.computerworld.com/s/article/43487/Application\\_Programming\\_Interface](http://www.computerworld.com/s/article/43487/Application_Programming_Interface). [Accessed: 26-Mar-2011].
- [8] A. Hepworth, "Methods to Streamline Laminate Composite Design, Analysis, and Optimization," M.S. thesis, Mech. Eng., BYU, Provo, UT, 2010.
- [9] K. J. Mix, "Integrating Engineering and Communication Tools for the Automation of Design Rationale Capture," M.S. thesis, Mech. Eng., BYU, Provo, UT, 2010.
- [10] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics The Finite Volume Method*, Second. Harlow, England, UK: Pearson Education Limited, 2007, p. 3.



- [11] X. Zhu, M. E. Gleason, and K. Sbeih, "A CFD Application of Surface Morphing for Vehicle Exterior Development," *SAE International World Congress*, no. 2007-01-0110, 2007.
- [12] P. Hoke and M. Doroudian, "Optimization of an IP and Console Duct using Advanced Shape Deformation Technology," in *3rd European Automotive CFD Conference*, 2007. Available: <http://www.ansys.com/industries/automotive/advanced-shape-deformation-9972.pdf>
- [13] R. Singh, "Automated Aerodynamic Design Optimization Process for Automotive Vehicle," *SAE International World Congress*, no. 2003-01-0993, 2003.
- [14] M. Cross and D. Smith. (2005, Feb 14). *Generic Automotive Engine Intake Port Optimisation*. Available: <http://www.scribd.com/doc/24992703/Whitepaper-Engine-Intake-Optimization>
- [15] X.-W. Gao, P.-C. Chen, and L. Tang, "Deforming Mesh for Computational Aeroelasticity Using a Nonlinear Elastic Boundary Element Method," *AIAA Journal*, vol. 40, no. 8, pp. 1512-1517, Aug. 2002.
- [16] Altair, "Altair HyperMesh Users Guide Version 11.0." 2011.
- [17] A. T. Hosford, "Mesh Deformation Approach for Aerodynamic Optimization Using a Genetic Algorithm," B.S. honors thesis, Mech. Eng., BYU, Provo, UT, 2008.
- [18] F. Campos, P. Geremia, and S. Weston, "Fast Robust Design Optimisation of an Underhood Cooling Duct," in *NAFEMS World Congress*, 2007. Available: [http://www.iconcf.com/media/publications/F.Campos\\_etal\\_NAFEMS2007.pdf](http://www.iconcf.com/media/publications/F.Campos_etal_NAFEMS2007.pdf)
- [19] M. Usan, "Automotive component product development enhancement through multi-attribute system design optimization in an integrated concurrent engineering framework," M.S. thesis, Eng.& Mgt., MIT, Boston, MA, 2005.
- [20] V. A. Jairazbhoy, L. Wang, M. Shahabi, R. Nimbalkar, and S. Earla, "OPTIMIZATION OF AIR FLOW THROUGH A HEAT EXCHANGER IN AN AUTOMOTIVE HVAC," in *3rd ANSA &  $\mu$ ETA International Conference*, 2009, pp. 1-26.
- [21] M. L. King, "A CAD-Centric Approach To CFD Analysis With Discrete Features," M.S. thesis, Mech. Eng., BYU, Provo, UT, 2004.

- [22] “WordNet Search 3.0 ‘Optimal’,” *WordNet*. [Online]. Available: <http://wordnetweb.princeton.edu/perl/webwn?s=optimal>. [Accessed: 22-Apr-2011].
- [23] A. Parkinson, *Optimization-Based Design*. 2010, Provo: BYU.
- [24] R. L. Fox, *Optimization Methods For Engineering Design*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1971.
- [25] R. Fletcher, *Practical Methods of Optimization*, Second. Tiptree, UK: John Wiley & Sons, Inc., 1987.
- [26] H. Blaxill, J. Downing, J. F. Seabrook, and M. Fry, “A Parametric Approach to Spark-Ignition Engine Inlet-Port Design,” *SAE International Congress and Exposition*, no. 1999-01-0555, 1999.
- [27] S. Gaikwad, K. Arora, V. M. Korivi, and S. K. Cho, “Steady and Transient CFD Approach for Port Optimization,” *SAE International World Congress*, no. 2008-01-1430, 2008.
- [28] A. D. Khondge and S. Sovani, “Automation of Vehicle Aerodynamic Shape Exploration and Optimization using Integrated Mesh Morphing and CFD,” *SAE International*, no. 2011-01-0170, 2011.
- [29] N. Wyman, “CFD Review | State of the Art in Grid Generation.” [Online]. Available: <http://www.cfdreview.com/features/01/04/28/2131215.shtml>. [Accessed: 22-Aug-2011].
- [30] A. M. Mistreanu, “Computational Methodology for Large Scale Simulation of Flow in the Intake System of a V-8 Engine,” Ph.D. dissertation, Mech. Eng., Clemson Univ., 2003.





```

using std::cerr;
#else
# include<sstream.h>
# include<iostream.h>
#endif
#include<uf.h>
#include<uf_ui.h>
#include<uf_exit.h>
#include<uf_defs.h>
#include<NXOpen/NXException.hxx>
#include<NXOpen/Session.hxx>
#include<NXOpen/Expression.hxx>
#include<NXOpen/ExpressionCollection.hxx>
#include<NXOpen/PartSaveStatus.hxx>
#include<NXOpen/Part.hxx>
#include<NXOpen/PartCollection.hxx>
#include<NXOpen/Session.hxx>
#include<NXOpen/Unit.hxx>
#include<NXOpen/UnitCollection.hxx>
#include<NXOpen/Update.hxx>
#include<fstream>
#include<string>
#include<stdio.h>
#include"tokenizer.h"
#include<windows.h>
#include<sstream>
// added following for opening part
#include<NXOpen/BasePart.hxx>
#include<NXOpen/PartLoadStatus.hxx>
usingnamespace NXOpen;
usingnamespace std;
#define UF_CALL(X) (report_error( __FILE__, __LINE__, #X, (X)))
staticint report_error( char *file, int line, char *call, int irc)
{
if (irc)
{
char err[133],
msg[133];
sprintf(msg, "*** ERROR code %d at line %d in %s:\n+++ ",
irc, line, file);
UF_get_fail_message(irc, err);
UF_print_syslog(msg, FALSE);
UF_print_syslog(err, FALSE);
UF_print_syslog("\n", FALSE);
UF_print_syslog(call, FALSE);
UF_print_syslog(";\n", FALSE);
}
return(irc);
}
// functions for parsing
// declare identifiers for lines to be parsed
enum paramType
{
width_,
diameter_,
centralAngle_
};
paramType WhichParamAmI(std::string paramName)

```

```

{
    if(paramName == "W")
        return width_;
    if(paramName == "D")
        return diameter_;
    if(paramName == "A")
        return centralAngle_;
}
/* Main routine */
externvoid main( char argc, char *argv[] )
{
/* Initialize the API environment */
if( UF_CALL(UF_initialize()) )
    {
/* Failed to initialize */
return;
    }
/* TODO: Add your application code here */
std::string line;
std::string file;
file = "C:\\MorphLoop\\shape.txt"; /* */
ifstream infile;
infile.open(file.c_str());
//set variable
string CentAngle;
string Diameter;
string width;
//define an STL vector of strings to hold the result in loop to clean up for each
object
vector<string> tokens;
    while (getline(infile, line, '\n'))
    {
        //delimiters defined
        string delimiters(", =;\t");
        // break the string into a list of tokens
        tokenize(line, tokens, delimiters);
        // skip blank lines
        if(tokens.empty())
            continue;
    }
infile.close();
if((int)tokens.size()>=3)
{
    width = tokens[1];
    Diameter = tokens[3];
    CentAngle = tokens[5];
}
Session *theSession = Session::GetSession();
BasePart *basePart1;
PartLoadStatus *partLoadStatus1;
basePart1 = theSession->Parts()->OpenBaseDisplay("C:\\MorphLoop\\sA_NX5.prt",
&partLoadStatus1);
Part *workPart(theSession->Parts()->Work());
Part *displayPart(theSession->Parts()->Display());
delete partLoadStatus1;
Session::UndoMarkId markId1;
markId1 = theSession->SetUndoMark(Session::MarkVisibilityVisible, "Change Display
Part");

```

```

PartLoadStatus *partLoadStatus2;
PartCollection::SdpsStatus status1;
status1 = theSession->Parts()->SetDisplay(workPart, true, true, &partLoadStatus2);
workPart = theSession->Parts()->Work();
delete partLoadStatus2;
theSession->Parts()->SetWork(workPart);
workPart = theSession->Parts()->Work();
Session::UndoMarkId markId2;
markId2 = theSession->SetUndoMark(Session::MarkVisibilityVisible, "Enter
Gateway");
Session::UndoMarkId markId3;
markId3 = theSession->SetUndoMark(Session::MarkVisibilityVisible, "Enter
Modeling");
// -----
// Menu: Tools->Expression...
// -----
Session::UndoMarkId markId4;
markId4 = theSession->SetUndoMark(Session::MarkVisibilityVisible, "Expression");
Expression *expression1(dynamic_cast<Expression *>(workPart->Expressions()-
>FindObject("angle")));
Unit *unit1(dynamic_cast<Unit *>(workPart->UnitCollection()->FindObject("Degrees")));
workPart->Expressions()->EditWithUnits(expression1, unit1,
(char*)CentAngle.c_str());
Expression *expression2(dynamic_cast<Expression *>(workPart->Expressions()-
>FindObject("diameter")));
Unit *unit2(dynamic_cast<Unit *>(workPart->UnitCollection()-
>FindObject("MilliMeter")));
workPart->Expressions()->EditWithUnits(expression2, unit2, (char*)Diameter.c_str());
Expression *expression3(dynamic_cast<Expression *>(workPart->Expressions()-
>FindObject("width")));
workPart->Expressions()->EditWithUnits(expression3, unit2, (char*)width.c_str());
int nErrs1;
nErrs1 = theSession->UpdateManager()->DoUpdate(markId4);
// -----
// Menu: File->Save As...
// -----
remove("C:\\MorphLoop\\ItWorks.prt");

PartSaveStatus *partSaveStatus1;
partSaveStatus1 = workPart->SaveAs("C:\\MorphLoop\\ItWorks.prt");
delete partSaveStatus1;
/* Terminate the API environment */
UF_CALL(UF_terminate());
}

```

## APPENDIX B. HYPERMORPH TCL SCRIPT FOR MESHING

```
# Imports NX part file layers 9 inlet 10 wall 11 outlet. Meshes pipe then exports CFD .cas file.
# Importing commands when part is imported naming collectors and organizing parts with
layers.
*createstringarray 12 "SELECTIONS 0 1 0 0 0 1 0 0 0 1" "" "BEGIN_PARTS" "END_PARTS"
"" "BEGIN_LAYERINFO" "DISABLE" "ENABLE 9-11" "END_LAYERINFO" ""
"COMPONENT_NAME custom <Layer>" "";
# to make the following line work "#ug\ug"
"C:/Users/William/Documents/Research/NX/TestPipe.prt" 1 0 -0.01 0 0 1 12 1 0 needed to be
changed to the following line with the brackets ug command refers to ext api function in
hypermesh
*feinputwithdata2 {#ug\ug} {C:/MeshLoop/ItWorks.prt} 1 0 -0.01 0 0 1 12 1 0 ;#
*settopologydisplaytype 1
#change name of collectors from numbers to names
*retainmarkselections 1
*renamecollector components "11" "outlet"
*retainmarkselections 0
*retainmarkselections 1
*renamecollector components "10" "wall"
*retainmarkselections 0
*retainmarkselections 1
*renamecollector components "9" "inlet"
*retainmarkselections 0
# commands to clean up geometry to get ready for 2D meshing
*createmark surfaces 1 "by collector name" "wall" "inlet" "outlet"
*selfstitchcombine 1 18 0.0254 0.0254
#commands used to 2d automesh part with QI optimize mixed mesh type element size 2.00 flow
align checked
*setedgedensitylink 0
*createdoublearray 89 1 1 0.00602 0.005418 0.002408 0.001204 0.0006019 1 1 0.00602
0.007223 0.009029 0.01204 0.01806 1 1 1 2 4.4 5 10 1 1 0 5 13 15 30 1 1 90 110 134 140 160 1
1 90 70 46 40 20 1 1 60 80 112 120 150 1 1 60 50 34 30 15 1 1 0 10 34 40 70 1 1 1 0.9 0.7 0.6
0.3 0 1 0 0.3 0.8 1 2 1 1 0 6 10 15 20 0 0 0.8 1 10
*reset_qa_calculator 1 89
*reset_elemsize_criteria 2
*elementorder 1
*createmark surfaces 1 "by collector name" "wall" "inlet" "outlet"
```



```

*optimized_mesh 1 "dummy" 2 2 1 30 1 1 1
#checks for T-connections
*createmark components 1 "wall" "inlet" "outlet"
*findedges components 1 1
#checks for edges
*createmark components 1 "wall" "inlet" "outlet"
*findedges components 1 0
# *createstringarray 2 "tet: 67 1.05 -1 0 1 0 0" "cfd: 118 0.09 8 1.3 0"
# *createmark components 1 "wall"
# *createmark components 2 "inlet" "outlet"
# *tetmesh components 1 3 components 2 0 1 2
# 3d mesh with boundary layer
# *createmark components 1 "wall"
# *createmark components 2 "inlet" "outlet"
# *tetra_cfdmesh_transition components 1 components 2 1.2 0.75 0.3 1.1 5 3 1 0 310 0.8
# following line is original method for meshing in first completed run, higher res boundary layer
and smooth boundary development
#*createstringarray 2 "tet: 67 1.05 -1 0 1 0 0" "cfd: 118 0.09 8 1.3 0"
*createstringarray 2 "tet: 35 1.05 -1 3.5 0.8 0 0" "cfd: 118 0.08 8 1.3 0"
*createmark components 1 "wall"
*createmark components 2 "inlet" "outlet"
*tetmesh components 1 3 components 2 0 1 2
# *createstringarray(2) "tet: 35 1.05 -1 3.5 0.8 0 0" "cfd: 118 0.2 3 1 0"
# *tetmesh(elements,0,-1,elements,0,-1,1,2)
# *createstringarray(2) "tet: 35 1.05 -1 3.5 0.8 0 0" "cfd: 118 0.2 3 1 0"
# *createmark(components,1) "10"
# *createmark(components,2) "9" "11"
# *tetmesh(components,1,3,components,2,0,1,2)
# create fluid component by renaming cfd tetramesh
*retainmarkselections 1
*renamecollector components "CFD_tetramesh_core" "fluid"
*retainmarkselections 0
# create inflow and outflow collectors
*createmark materials 1
*clearmark materials 1
*collectorcreateonly components "inflow" "" 64
*createmark components 1
*clearmark components 1
*createmark components 1 "inflow"
*materialupdate components 1 ""
*createmark components 1
*clearmark components 1
*createmark materials 1
*clearmark materials 1
*collectorcreateonly components "outflow" "" 64
*createmark components 1 "inflow"

```

```

*clearmark components 1
*createmark components 1 "outflow"
*materialupdate components 1 ""
*createmark components 1
*clearmark components 1
# move cfd boundary layer to fluid
*createmark elements 1 "CFD_boundary_layer"
*movemark elements 1 "fluid"
# delete cfd_boundary_layer collector
*retainmarkselections 1
*createmark components 1 "CFD_boundary_layer"
*deletemark components 1
*retainmarkselections 0
# find faces of fluid and make faces collector
*createmark components 1 "fluid"
*findfaces components 1
# isolate faces collector
*retainmarkselections 1
*createmark components 1 "^faces"
*createmark surfaces 1 "fluid"
*displaycollectorsallbymark 1 "isolateonly" 1 1
*createmark groups 2
*displaycollectorsbymark groups 2 "isolate" 1 1
*retainmarkselections 0
# moved element faces by
# need to create variables that specify the plane for both the inflow and the outflow this will be
based on the layer of the place in CAD
*createmark elements 1 "on plane" 1 380 1 0 1 0 0.1 1 1
*movemark elements 1 "inflow"
*createmark elements 1 "on plane" 1 0 1 0 1 0 0.1 1 1
*movemark elements 1 "outflow"
# delete remaining faces component
*retainmarkselections 1
*createmark components 1 "^faces"
# "inlet" "outlet"
*deletemark components 1
*retainmarkselections 0
# display only needed components
*displaycollectorwithfilter components "on" "wall" 1 0
*displaycollectorwithfilter components "on" "fluid" 1 0
*displaycollectorwithfilter components "on" "inflow" 1 0
*displaycollectorwithfilter components "on" "outflow" 1 0
# un edited export of model to cas file
*retainmarkselections 0
*entityhighlighting 1
*entityhighlighting 0

```

```

*displaycollectorwithfilter components "none" "" 0 1
*createmark components 1 "displayed"
*clearmark components 1
*createmark nodes 1 "displayed"
*clearmark nodes 1
*createmark elements 1 "displayed"
*clearmark elements 1
*createmark elements 2 "fluid"
*clearmark elements 2
*createmark elements 2
*clearmark elements 2
*createmark elements 1 "wall"
*createmark elements 2 "wall" "inflow" "outflow"
*markintersection elements 1 elements 2
*createmark elements 1 "wall"
*clearmark elements 1
*createmark elements 1 "fluid"
*createmark elements 2 "wall" "inflow" "outflow"
*markintersection elements 1 elements 2
*createmark elements 1
*clearmark elements 1
*createmark elements 1 "inlet"
*createmark elements 2 "wall" "inflow" "outflow"
*markintersection elements 1 elements 2
*createmark elements 1 "inlet"
*clearmark elements 1
*createmark elements 1 "outlet"
*createmark elements 2 "wall" "inflow" "outflow"
*markintersection elements 1 elements 2
*createmark elements 1 "outlet"
*clearmark elements 1
*createmark elements 1 "displayed"
*clearmark elements 1
*createmark components 1 "fluid"
*collectormarkmove components 1 1 0
*createmark components 1 "displayed"
*clearmark components 1
*createmark elements 1 "wall" "inflow" "outflow"
*findedges elements 1 0
*createmark elements 1
*clearmark elements 1
*createmark nodes 1 "displayed"
*clearmark nodes 1
*nodecleartempmark
*cfoutput "C:/MeshLoop/mesh.cas" "ASCII" "FLUENT" "0" ""
*entityhighlighting1

```

## APPENDIX C. HYPERMORPH TCL SCRIPT

```
# code to import cas file and modify cas file then export new case file with solving method and
previous solved data
# import CAS file
*view "rear"
*entityhighlighting 0
*createmark components 1
*clearmark components 1
*feinput "C:/Program Files/Altair/11.0/io/model_readers/feinput/bin/win64/fluent_cas_msh.exe"
"C:/MorphLoop/morphOrig.cas" 1 0 -0.01 1 0
*createmark components 1 "all"
*clearmark components 1
*entityhighlighting 1
# I will need to make sure that the file name can be adjusted or that the saving method will match
what is determined
# import original geometry
*createstringarray 12 " SELECTIONS 0 1 0 0 0 1 0 0 0 1" "" "BEGIN_PARTS"
"END_PARTS" "" "BEGIN_LAYERINFO" "DISABLE " "ENABLE 2" "END_LAYERINFO"
"" "COMPONENT_NAME custom <Layer>" ""
*feinputwithdata2 {#ug\ug} {C:/MorphLoop/sA_NX5.prt} 1 0 -0.01 0 0 1 12 1 0 ; #
# rename imported
*retainmarkselections 1
*renamecollector components "2" "old_line"
*retainmarkselections 0
# import new geometry
*createstringarray 12 " SELECTIONS 0 1 0 0 0 1 0 0 0 1" "" "BEGIN_PARTS"
"END_PARTS" "" "BEGIN_LAYERINFO" "DISABLE " "ENABLE 2-3"
"END_LAYERINFO" "" "COMPONENT_NAME custom <Layer>" ""
*feinputwithdata2 {#ug\ug} {C:/MorphLoop/ItWorks.prt} 1 0 -0.01 0 0 1 12 1 0 ; #
# change name of new geometry component
*retainmarkselections 1
*renamecollector components "2" "new_line"
*retainmarkselections 0
*retainmarkselections 1
*renamecollector components "3" "new_surface"
*retainmarkselections 0
#scale imported geometry to match flow model
```

```

*createmark components 1 "old_line" "new_line" "new_surface"
*scalemark components 1 0.001 0.001 0.001 0
>window 0 0 0 0
#create domains 2D then 3D
# only 2 domain groups created in following method
*morphstoredomains 1
*createmark elements 1 "by comps" "wall" "inflow" "outflow"
*morphcreatedomaindc elements 1 2 0 0 1 0
*morphstoredomains 1
*createmark elements 1 "by comps" "fluid"
*morphcreatedomaindc elements 1 3 0 0 1 0
# -----get mark id of lines-----
*createmark lines 1 "by comp" "old_line"
*createmark lines 2 "by comp" "new_line"
set mee [hm_getmark lines 1]
set mee2 [hm_getmark lines 2]
*morphstoremorphvolumes 3
*createmark nodes 1 "fluid"
*createmark nodes 2
if { [ catch {eval *createlists lines 1 $mee} ] } {
#Handle error
}
*createlists nodes 1
if { [ catch {eval *createlists lines 2 $mee2} ] } {
Handle error
}
*createlists nodes 2
*createplane 1 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
# entity type(nodes) mark id(1,2) f_entity_type(nodes) f_mark_id(1,2) i_line_list(1,2)
i_node_list(1,2) f_line_list(1,2) f_node_list(1,2) r_plane(1,2) rotate(0 linear mapping, 1 rotate
nodes) axis(0-5) sym(0) con(0,1) blend mbias(value) fbias(value)
# *morphmapdifference(nodes,1,nodes,2,1,1,2,2,1,1,5,0,1,0,1,1)
if { [ catch {*morphmapdifference nodes 1 nodes 2 1 1 2 2 1 1 5 0 1 0 1 1 } ] } {
#Handle error
}
# Morph to edges at ends first
# method with original morph case to morph outflow to edge of surface
*morphstoremorphvolumes 3
*createmark nodes 1 86342-86382
*createmark handles 1
*createlists lines 2 21
*createlists nodes 2
*createvector 1 1.0000 0.0000 0.0000
*morphmaptolinevec nodes 1 handles 1 2 2 0 1 1 1 1
# method with original morph case to morph inflow to edge of surface
*morphstoremorphvolumes 3

```

```

*createmark nodes 1 1 2 91859-91897
*createmark handles 1
*createlist lines 2 11
*createlist nodes 2
*createvector 1 1.0000 0.0000 0.0000
*morphmaptolinevec nodes 1 handles 1 2 2 0 1 1 1 1
# Morph to surface commands
# Using map to surfaces and then using map nodes by collector this works
*morphstoremorphvolumes 3
*createmark nodes 1 "by comps" "wall"
*createmark handles 1
*createmark surfaces 1 "by comps" "new_surface"
*createvector 1 1.0000 0.0000 0.0000
# *morphmaptosurfacevec(nodes,1,handles,1,surfaces,1,0,1,1,1,1)
*morphmaptosurfacevec nodes 1 handles 1 surfaces 1 0 1 1 1 1
#export cas file using previous cas information
*retainmarkselections 0
*entityhighlighting 1
*entityhighlighting 0
*displaycollectorwithfilter components "none" "" 0 1
*createmark components 1 "displayed"
*clearmark components 1
*createmark nodes 1 "displayed"
*clearmark nodes 1
*createmark elements 1 "displayed"
*clearmark elements 1
*createmark elements 2 "fluid"
*clearmark elements 2
*createmark elements 2
*clearmark elements 2
*createmark elements 1 "wall"
*createmark elements 2 "wall" "inflow" "outflow"
*markintersection elements 1 elements 2
*createmark elements 1 "wall"
*clearmark elements 1
*createmark elements 1 "fluid"
*createmark elements 2 "wall" "inflow" "outflow"
*markintersection elements 1 elements 2
*createmark elements 1
*clearmark elements 1
*createmark elements 1 "inlet"
*createmark elements 2 "wall" "inflow" "outflow"
*markintersection elements 1 elements 2
*createmark elements 1 "inlet"
*clearmark elements 1
*createmark elements 1 "outlet"

```

```
*createmark elements 2 "wall" "inflow" "outflow"  
*markintersection elements 1 elements 2  
*createmark elements 1 "outlet"  
*clearmark elements 1  
*createmark elements 1 "displayed"  
*clearmark elements 1  
*createmark components 1 "fluid"  
*collectormarkmove components 1 1 0  
*createmark components 1 "displayed"  
*clearmark components 1  
*createmark elements 1 "wall" "inflow" "outflow"  
*findedges elements 1 0  
*createmark elements 1  
*clearmark elements 1  
*createmark nodes 1 "displayed"  
*clearmark nodes 1  
*nodecleartempmark  
*cfdoutput "C:/MorphLoop/morph.cas" "ASCII" "FLUENT" "1" "C:/MorphLoop/morph.cas" 1 1  
*entityhighlighting
```

## APPENDIX D. FLUENT JOURNAL FILE

### D.1 Mesh Fluent Journal

```
rc C:/MeshLoop/mesh.cas
;to scale mesh to mm
mesh/scale .001 .001 .001
file/read-bc
C:/MeshLoop/bclam
in/i-f
it 300
file/read-bc
C:/MeshLoop/bc1st
in/i-f
it 300
;file wcd C:/MeshLoop/fluent.dat y
file/read-bc
C:/MeshLoop/bc2nd
in/i-f
it 800
report/surface-integrals/area-weighted-avg inflow outflow () pressure yes "pressure.txt" n y
report/fluxes/mass-flow yes yes "mdot.txt" n y
file wcd C:/MeshLoop/fluent.dat y
exit
```



## D.2. Morph Journal File

```
rcd C:/MorphLoop/morph.cas
file/read-bc
C:/MorphLoop/bclam
solve/iterate 300
file/read-bc
C:/MorphLoop/bc1st
solve/iterate 300
file/read-bc
C:/MorphLoop/bc2nd
solve/iterate 800
report/surface-integrals/area-weighted-avg inflow outflow () pressure yes "pressure.txt" no yes
report/fluxes/mass-flow yes yes "mdot.txt" n y
file/wcd C:/MorphLoop/morph.dat yes
exit
```