Brigham Young University

BYU ScholarsArchive

2010-03-12

Recognizing Parametric Geometry from Topology Optimization Results

Shane H. Larsen
Brigham Young University - Provo

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

Part of the Mechanical Engineering Commons

www.manaraa.com

Recognizing Parametric Geometry from

Topology Optimization Results

Shane H. Larsen

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
Scott Thomson
Denise Halverson

Department of Mechanical Engineering

Brigham Young University

April 2010

ABSTRACT

Recognizing Parametric Geometry from

Topology Optimization Results

Shane H. Larsen

Department of Mechanical Engineering

Master of Science

Topology Optimization has been proven to be a useful tool in discovering non-intuitive optimal designs subject to certain design constraints. The results of Topology Optimization are either represented as a tessellation object composed of thousands of triangular surfaces, or as a point cloud. In either case, the results of Topology Optimization are not suited for use in subsequent steps of the design process which require 3D parametric CAD (Computer Aided Design) models. Converting Topology Optimization results into parametric CAD geometry by hand is an extremely tedious and time consuming process which is highly subjective. This thesis presents a shape recognition algorithm that uses a feature by feature CAD-centric approach to convert Topology Optimization results into parametric CAD geometry. This is accomplished by fitting 2D cross section geometry to various parts of a given feature through the use of Shape Templates and then constructing 3D surfaces through the set of 2D cross sections. This algorithm aids in measuring the geometric approximation error of the generated geometry as compared to the optimal model, and standardizes the process through automation techniques. It also aids the designer / engineer in managing the direct tradeoff between closeness of geometric approximation (measured by volumetric comparison) and model complexity (measured by the number of parameters required to represent the geometry).

Keywords: Shane H. Larsen, topology optimization, shape templates, shape recognition

ACKNOWLEDGMENTS

TABLE OF CONTENTS

LIST OF TABLES

x

LIST OF FIGURES

1   INTRODUCTION

Topology Optimization (TO) is a process used to distribute material in an optimal manner throughout a predefined design domain. It has been used heavily in mechanical design [15][8], electromagnetic design [5], and a myriad of other industries to generate optimal configurations. It is capable of generating design concepts that are, many times, outside of the realm of human intuition since predefined parameters or model features are not needed [16]. This optimization process is performed on finite element models (FEM) comprised of nodes and elements and therefore, the results are also in the form of nodes and elements and are generally represented in IGES or STL format.

After the TO process is complete, the resultant optimal topology is used as a visual reference from which a parametric CAD model can be created. A parametric CAD model is the preferred format for all subsequent design processes including parametric optimization and manufacturing process planning. A parametric CAD model can be controlled and adapted by a finite set of parameters that control the entire model. This makes it possible for the designer to change and update the model quickly and automatically as new discoveries are made or new customer needs are revealed.

1.1   Problem Statement

The current method of manual interpretation of TO results by a human designer is very time intensive and lacks standardization. Under these circumstances the use of TO in industry is

1

limited due, in part, to the expense of interpreting the results once the optimization run is complete. A number of researchers have also pointed out the need for an automated method for interpreting the results of TO. [8][11][16] If a more automated, standardized approach to TO results interpretation and implementation were developed TO could be utilized in more industries allowing more efficient designs to be created.

1.2 **Objectives**

It is my objective to implement methods which improve the process of converting TO results into Parametric CAD geometry through an improved shape recognition algorithm. This algorithm would operate in a CAD-centric environment, use intuitive standard CAD features, and allow the designer to control the complexity of the resulting features in terms of number of defining parameters required to represent them. Ideally, this process would be automated, be repeatable, and reduce interpretation time requirements. It would also provide the appropriate setup to parameterize the CAD model in such a way that parametric optimization could be performed after the conversion to CAD is finished. The main points of this thesis are:

1. Create automated processes to evaluate a given set of points and surfaces from TO results and determine the best fit parametric CAD feature to represent the set of surfaces and points.

2. Implement the automation processes in a CAD-centric environment.

3. Show that the resultant CAD feature is a good approximation of the optimal topology by volume comparison.

2

To demonstrate the proposed methods, a set of basic cantilevered beam cases will be used. These cases are well known in the field of structural optimization. Graphical representations of these cases are presented in Figure 1-1.
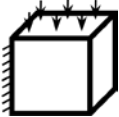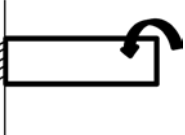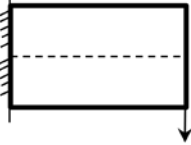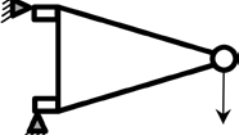
Case I: Cantilevered Cube with distributed load	Case II: Cantileverd Beam under torque load	Case III: Cantilevered Beam with point load and symmetry	Case IV: Automobile A-Arm with a point load

Figure 1-1: Thesis case studies

1.3 **Statement of Scope**

The scope of this research is most easily delineated within the framework of the design process. A typical design process sequence utilizing TO is

1. Geometric Design

2. Optimization Pre-Processing

3. Topology Optimization (TO)

4. Generate IGES/STL of TO results

5. Import IGES/STL into CAD

6. Conversion to Parametric CAD

7. Parametric Optimization

8. Manufacturing Process Planning

This research assumes that TO has been completed and that a discreet model has been obtained (steps 1-5). This can be done using any TO software so long as the TO results can be

3

transferred to a CAD software package in the form of standard STL or IGES file formats. The main focus of the research is on step 6 of the design process, Conversion to Parametric CAD.

Although the methods proposed in this research are application independent, the implementation will be done within CATIA V5 R18 using the CAA RADE API interface. While this API is very advanced, the implemented methods use functionality within this API that is common to most CAD application APIs.

As will be seen within this thesis, the process of converting TO results into CAD geometry is split into a number of sub-steps. Some of the sub-steps are not included in the scope of this research due to resource limitations and time constraints.

1.3.1   Feature Segmentation/Selection

Feature segmentation is the process of recognizing one feature from another within the same model. Although some method of feature segmentation is needed in order to implement the actual research findings in this thesis, different methodologies for accomplishing this task were not researched or addressed. The research assumes that the recognition algorithm is given a set of surfaces that are all part of a single feature to be recognized.

Feature selection is the process of selecting a subset of points or faceted surfaces from the TO results that define a single feature of the model. This is related to feature segmentation and is also not treated in the current research.

1.3.2   Parametric Optimization (Shape Optimization)

After the conversion from TO results to parametric CAD geometry, it is assumed that a subsequent process of parametric optimization will be performed on a given part. As this process

4

is well defined in industry, it is not treated within the scope of this research. This is, however, an active area of research.

1.4 **Guiding Principles**

This research offers a number of benefits to industry standard practices. It is hoped that the resultant methods can easily be implemented within the current design architecture that is being used in industry without great disruption to that architecture. The following section will provide the reader with insight into the ideology that guided the development of the methods presented in this thesis.

1.4.1 Simple

The methods and processes proposed in this research are intended to provide a simple way to interpret TO results that is intuitive to a human designer. The benefits of simplicity are hard to quantify, but are far reaching. A simple process is easy to teach to new employees and can be executed by lower level employees thus achieving cost savings in business processes. The more complex a tool becomes, the less likely it is that the tool will have the intended beneficial impact.

The methods outlined in this process also allow the designer to maintain control over the level of complexity of the resultant model in terms of number of parameters required to create the CAD geometry. This results in computational savings in subsequent steps of the design process. [11]

5

1.4.2 Parametric

As Parametric CAD models have become the standard for product design, it is essential that new design processes support this architecture. The methods proposed by this research improve the link between TO and parametric CAD.

1.4.3 Automatic

The most apparent benefit of this thesis will likely be the time savings from an automated approach to TO results interpretation. It is expected that, even on simple parts, some time savings will be realized while on complex parts the time required for conversion to parametric CAD could be reduced substantially.

Automation also improves repeatability. Repeatability in TO results interpretation refers to the ability for a single person to produce the same design multiple times from the same given starting point. The current method of manual TO results interpretation reduces repeatability due to human involvement in every small parameter and aspect of the geometric design. When more of the fine details of the design are moved to an automated algorithm, as this research proposes to do, the repeatability of the design is improved.

1.4.4 Standardized

Standardization refers to process similarity from one person to the next. Currently, there is no industry standardization in the process of TO results interpretation outside of a company's 3D modeling standards. Due to this lack of standardization two co-workers may follow very different approaches to TO results interpretation which could lead to vastly different business outcomes. This causes discontinuities in the design process as personnel in a company change

6

from time to time. It is beneficial to know that when a certain engineer or designer leaves their current position, the person who comes in to replace them will be able to produce similar results. This research provides a standard approach to TO results interpretation to facilitate continuity of design over time.

1.4.5 Measured

Another drawback of the current process is the lack of a fitness measure to know if the resultant geometry matches the optimal topology. In order to make decisions about the level of complexity of a given model, information about the fitness of the model compared to the optimal geometry obtained from the TO is required. This research incorporates a least squares fitness of each feature of the model thus providing the designer with the information needed to make decisions about the inherent tradeoff between model complexity and geometric fitness.

1.5 Thesis Organization

As Topology Optimization gains momentum in industry and is used more widely, it will become ever more important to automate the process of interpreting the optimization results. The following chapters of this thesis will delve into the details of how this can be done in accordance with the above guiding principles. Chapter 2 will provide background information regarding how others have attempted to solve this problem in the past, as well as foundational work in different fields that relate to the methods in this research. Chapter 3 will then present in detail the proposed methods. Chapter 4 will present the CATIA V5 R18 specific implementation used as a proof of concept of the methods from Chapter 3 along with the results of the four case studies.

Chapter 5 will conclude the research by analyzing the results from Chapter 4 to determine if the research objectives have been met.

2   LITERATURE REVIEW / BACKGROUND

The following literature review includes the relevant foundational work in the fields of Topology Optimization and Shape Recognition that this research builds upon. It is intended to inform the reader regarding what has been accomplished in these fields. A basic understanding of this foundational work is required to fully understand the proposed methods.

2.1   Topology Optimization

This research does not directly affect the process of Topology Optimization, but only the post-processing and interpretation of TO results. The process of TO, however, affects how post-processing and interpretation can occur. Therefore, it is imperative to understand the basics of the process of TO. This section presents the aspects of TO that are pertinent to this research.

2.1.1   The Soft Kill Method

Though there are many different methods of topology optimization [14], this research will focus on the Soft Kill Method of TO [1]. TO is applied to a finite element model (FEM) which defines a design space representing the maximum area or volume that the model is allowed to exist in. Once boundary conditions, model constraints, and optimization constraints are defined the optimization process begins [2]. The optimization algorithm for the Soft Kill Method adjusts each element's material properties such as density or Young's Modulus between 0% and 100% of the actual material property value and then observes the resultant change in

9

element stresses throughout the whole model. If the objective of the optimization is to reduce material mass, the algorithm will attempt to push all of the element densities to zero. When any given element reaches some predefined stress constraint the algorithm will discontinue reducing its density. Through iterating on this process, the optimal material distribution for the given loading conditions is found.

2.1.2 Tessellation and Visualization of Topology Optimization Results

Once the optimization process has converged to a solution, the topology must be viewed to decide if the result is acceptable and feasible. Using the Soft Kill method, there are truly infinite possible solutions due to the variation in density or Young's Modulus throughout the model. To view the solution, a density threshold is chosen from which an iso-surface can be displayed representing the optimal part. This iso-surface represents the boundary around all elements with a density equal to or greater than the density threshold. To illustrate the concept of an iso-surface imagine that the beginning design space for a problem was a cube and that a cylindrical rod is the optimal solution. Figure 2-1, shows a cross section representing this problem. The radial gradient represents the element densities of the solution. The lighter color of the gradient represents a less dense material. The designer can decide what density threshold to use to create an iso-surface. Several possible iso-surface solutions are represented in Figure 2-1.

It is important to note that this surface is based on the original design space finite element model. This means that the resultant iso-surface is similar to the finite element model in that it is made of a combination of lines and vertices which form quadrilateral or triangular surfaces (not a smooth curve as shown in Figure 2-1).

10
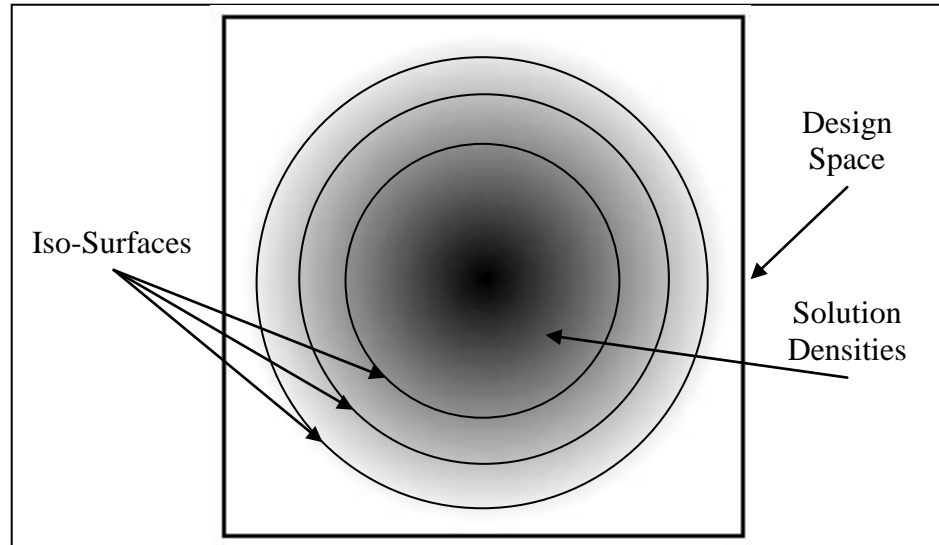
Figure 2-1: Example Iso-Surface (cross section view)

This is sometimes referred to as a tessellation object. Smoothing algorithms can be used to reduce the number of lines and vertices required to create the tessellation object, but the result will still be a tessellation object. Tessellation objects are not compatible with parametric CAD because there are no part features defined only thousands of points in space connected by lines that have no relationship to one another.

2.2 **TO Results Interpretation - Previous Approaches / Architectures**

On a basic level, TO results interpretation is similar to other fields of research that attempt to extract geometry from or represent geometry with point clouds such as reverse engineering, medical imaging, and computer graphics. There have been many different approaches used in these various fields to identify features from point clouds. A few recent approaches are presented below.

11

2.2.1 Application Linked Architecture

Most commercial CAD packages available today do not include TO solvers and pre-processors. In fact, historically, a different software application was used to mesh, solve, and post-process Finite Element Analyses (FEA) followed by another software application to create the results of the analyses in a CAD environment. Many Engineering firms operate in this type of environment today. Blattman [4] presented an overall process for converting TO results to parametric CAD that linked these different applications together through API programming, system calls, and stand alone C++ executable files. He also pointed out the need for more robust shape recognition algorithms which would enhance this overall process, although that was not his focus directly. Blattman's work was a feature by feature approach to TO results interpretation that gave the designer instantaneous feedback within the CAD environment about the progress of the interpretation process.

Since this time of linked architecture, the industry has seen much by way of consolidation. The move is clearly toward a CAD-centric architecture [10] where all types of analysis and optimization are performed in one multi-faceted software application. This means that geometric design, meshing, analysis and optimization are all performed in a single 3D CAD environment.

Although many of the principles of the conversion from TO results to CAD incorporated into the Application Linked Architecture are extremely valuable, it is clear that a CAD-centric design environment is preferred in industry. Accordingly, one objective of the methods presented in this thesis is to perform as much of the interpretation process within the CAD application as possible.

12

2.2.2 Mesh Refinement

Mesh refinement algorithms come mainly from laser scanning applications used in reverse engineering. They are used in most cases to eliminate redundant data points that don't add any unique geometric information [3][7] . A very simple example of a mesh refinement algorithm would be a program that recognizes when three points are approximately collinear. In this case, the point in the middle of the three points can be eliminated without losing any significant geometric data since a line is completely defined by 2 points, not three.

Mesh refinement algorithms are currently applied in commercial TO applications when exporting the chosen iso-surface to IGES or STL formats to create tessellations with fewer point densities. It has not been used in the conversion from TO to CAD geometry however. Even though mesh refinement algorithms are readily applied to TO results, the result of a mesh refinement is usually still a mesh; not parametric geometry. While mesh refinement has many benefits in terms of computational savings, it does not address the fundamental problem of converting TO results into parametric CAD geometry.

2.2.3 Image Processing

Image processing is typically applied in a 2D realm where a model is represented by a gray level image and segmented into square pieces as shown in Figure 2-2. A TO results file can readily be converted into this kind of gray level segmented image format based on finite elements and densities. The "elements" are then considered existent or non-existent based on a grayness threshold set by the user. The model is then converted to a binary black and white image based on this threshold. This effectively filters out the unimportant information from the Gray level image and the algorithm can then perform its function.

13
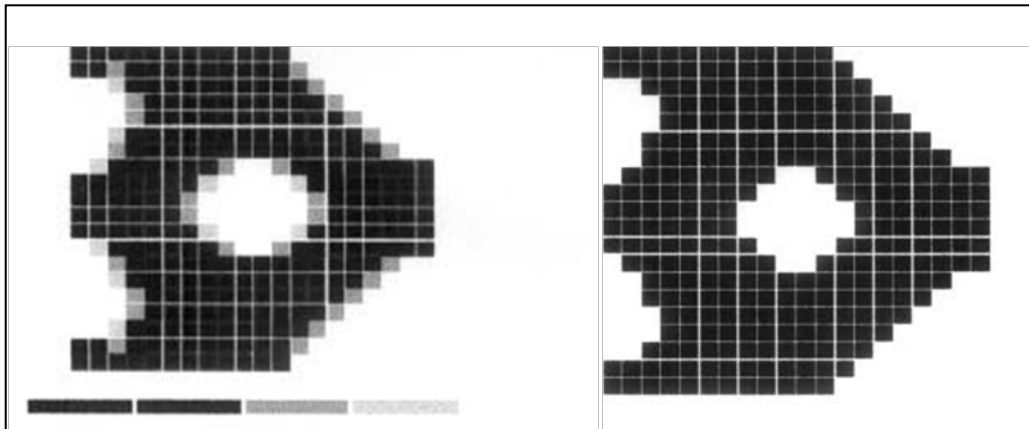
Figure 2-2: Conversion of gray level image to binary black and white

Image processing is used widely to identify part edges and boundaries and to obtain geometric points from those boundaries. Those points can then be used in any of the various shape recognition methods which will be covered in the following sections.

The technique of image processing has been widely used in TO results interpretation due to the compatibility of TO results with image processing input data. Even though it is best suited for 2D problems [11], it has also been used in 3D TO results interpretation with great success [15].

One of the main issues that is encountered with image processing is the reliance upon the finite element mesh. Current commercial TO applications provide the user with many different options for exporting the results of TO. As was mentioned earlier, one popular method is to apply mesh refinement algorithms which approximate the solution with fewer data points than the finite element mesh. However, this popular option removes the nicely organized mesh and reduces the number of data points available such that image processing techniques will have little data to work from. These problems are complicated further when extrapolated to the 3D realm. It

14

is the view of the author that industry standard practices that have emerged in recent years have rendered image processing a less effective tool in the TO results interpretation process.

2.2.4   2D Shape Templates

Lin and Chao [11] utilized image processing in conjunction with Shape Templates to automatically create 2D truss geometry from TO results files. After converting the gray level image to a discretized black and white image as described above, the outer edges of the model were created by fitting a B-Spline to the exterior points and then the truss members were formed by inserting voids that matched the white space of the image. To determine the shape and size of these voids, Lin and Chao utilized shape templates. An example of their results is shown in Figure 2-3.
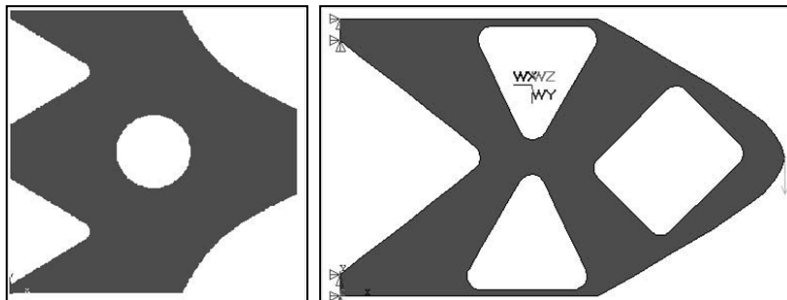
Figure 2-3: Examples of shape template fitting employed by Lin and Chao.

To create a shape template, a suitable, generic, convex shape is drawn and the distance (L) between a reference point within the shape and the edge of the shape is measured. This is repeated for various points around the periphery of the shape at different angles (θ). A polar map of the shape can then be graphed as shown in Figure 2-4.

15
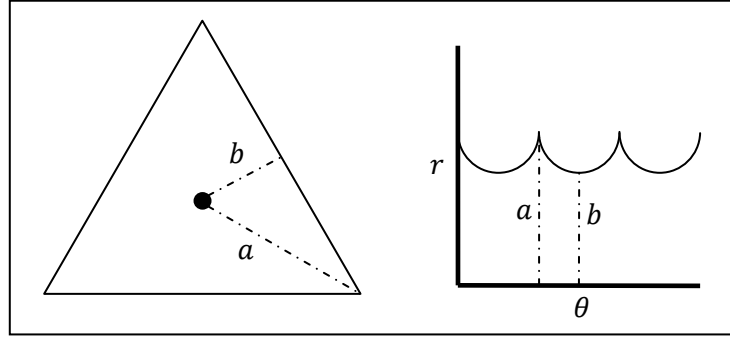
Figure 2-4: Example shape template

After the desired shape templates have been defined, samples from the TO results can be compared to the shape templates and the most appropriately fitting shape template can be chosen to use for that shape. Lin and Chao accomplish this task by calculating the standard deviation of measurements from the centroid of a void to its periphery and comparing this standard deviation to that of the known shape templates. This method has many strong points within the realm of the problems that Lin and Chao addressed. However, there are also several reasons why this method may not be ideal. For example, if the mesh is not uniform, the standard deviation will be skewed toward the higher density areas of the mesh making comparison of the standard deviations error prone. This method also does not take advantage of the fact that an analytical solution to this problem is readily available.

Shape templates are a convenient way to represent the topological data of features due to the ease of comparing sample data to shape templates in polar space. They also offer the ability to define different levels of complexity based on the number of parameters the designer is willing to allow the template to assume. If a more perfect fit to the optimal part is required, the designer can make the decision to allow templates with more defining parameters which will fit more perfectly.

16

2.2.5 B-Spline Cross Section Fitting

Tang and Chang [15] also used image processing but they extended it to 3D applications of TO results interpretation. This was accomplished by applying a 2D process to many different cross sections throughout the part and constructing a surface through the cross sections. The 2D cross sections were formed by fitting B-Splines to the resultant points of the image processing step.

Through this process, Tang and Chang were able to automatically reconstruct a manufacturable, optimized Roadarm of a tracked vehicle. Figure 2-5 shows the initial design of the Roadarm which was taken directly from the TO results, as well as the optimal design after shape optimization occurred.
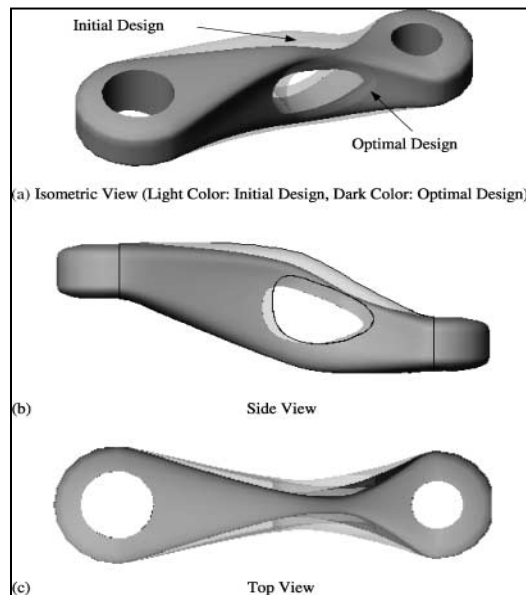
Figure 2-5: Reconstructed road arm from TO results

Tang and Chang had great success using B-Splines to approximate the TO results and created novel methods to transition between different numbers of B-Splines per cross section.

17

Much of their work acted as inspiration and as a starting point for this thesis. It is the intent of the author to add to what has been done by accomplishing a similar task on a feature by feature basis to allow the designer more control of the process. It is also the author's intent to use simpler CAD objects than B-Spline curves to reduce the number of parameters required to represent the whole model.

2.2.6 Shape Similarity Matching Algorithms

Shape similarity matching algorithms are used to search through databases of 3D geometric part designs to find specific types of parts. For example, if you were designing a part that needed a fastener similar to a screw or a bolt, but didn't know exactly which one would be ideal for your given design problem it would be useful to be able to perform a search on a 3D part database and retrieve all parts that are similar to a screw. This may retrieve nails, screws, bolts, rivets, rods, etc. From the results, the best option can be chosen to fulfill the needs of the given situation.

Shape similarity matching algorithms can be used in TO feature recognition by generating a library of known manufacturable shapes that can be created as parametric CAD features. Then, in the same way that the algorithms were intended to be used, a search can be performed to find the shape from the library with greatest similarity score to the TO results sample. Then, this shape can be used in the parametric CAD model to approximate the optimal shape from the TO results. Since most 3D models are represented as tessellation objects, shape similarity matching algorithms are especially useful for TO results processing. Many of the issues faced in TO results interpretation are the same problems found in the field of computer graphics and are taken into account in most algorithms. These techniques have yet to be applied directly to TO results interpretation, but offer great insight into possibilities for the future.

18

One method of shape similarity matching is referred to as shape distribution comparisons. [13] A shape distribution is a plot of the probability of getting different values for a certain random measurement of a shape. For example, the shape distributions of the distances measured between two random points on familiar shapes are presented below in Figure 2-6.
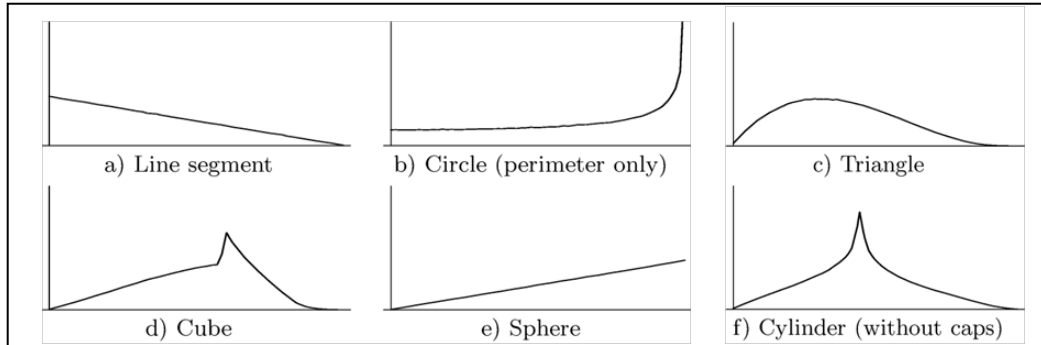
Figure 2-6: Shape distributions of familiar shapes

2.3 **API Programming**

An Application Programming Interface, or API, allows access to most internal functionality of an application. For example, Excel, a popular spreadsheet application from Microsoft, provides access to all of its functionality through a VBA (Visual Basic for Applications) API. In Excel, most anything that can be done interactively through the mouse and keyboard can be done through the VBA API. This makes it possible to create customized functionality that can improve the efficiency of the user. There may be several different APIs available for a single application. For example, you may access functionality of a given application using VBA, C++, of Java programming languages.

API programming is an essential part of implementing the methods in this research. Theoretically, the API does not affect the proposed methods. In some cases, however, an API

19

could be limiting in the scope of access it provides to the core functionality of the application and, therefore, would limit the implementation of a given set of methods.

3   METHODS

This chapter presents general methods for shape recognition to convert TO results information into parametric CAD features. These methods allow the engineer/designer to maintain control over the tradeoff between fitness to the optimal model and feature complexity. A process overview will first be presented to familiarize the reader with the general steps involved in the methods. In this section many important terms and concepts will also be defined. After the process overview, the details of the shape recognition algorithm will be presented.

Although much of the language in this thesis regarding TO refers to structural optimization, the methods are applicable to any use of TO resulting in a geometric solution.

3.1   **Process Overview**

The shape recognition algorithm employed in this research is intended to be used on the design space as defined in the TO within the CAD environment since the algorithm itself constructs geometry. This creates instantaneous feedback for the designer to see that the TO results interpretation process is going as desired.

The algorithm takes a set of B-Rep (Boundary Representation) surfaces as input along with a few geometric parameters that describe the feature orientation. It then recognizes the best *feature* to create to approximate the volume enclosed by the surfaces (a *feature* will be defined in detail later in this section).

21

The algorithm finishes by constructing the best fit feature and performing a Boolean operation between the design space and the newly constructed feature in order to either add to or remove material from the design space. This process is then repeated for each desired feature to create a 3D parametric solid model that is ready for parametric optimization and manufacturing planning. This overall process is represented in Figure 3-1.
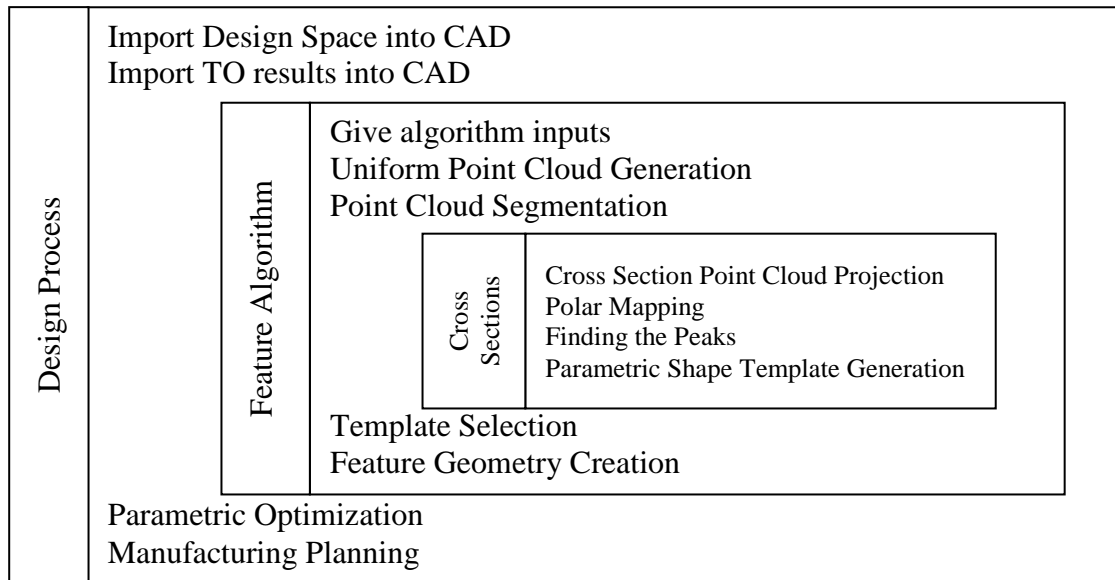
Figure 3-1: Overall process

In order to perform TO, a design space must first be established within which material can be distributed in any manner. This design space is typically a very simple geometric form and can readily be imported into a CAD application and converted into a solid model. Throughout this thesis the subset of $\mathbb{R}^3$ defining this design space will be represented by DS. DS entirely defines the available solution sets to the problem. Given this fact, any subset of $\mathbb{R}^3$ forming the optimal solution set S_{opt} resulting from TO must be a subset of the original design space. This

concept is represented mathematically by:

$$S_{opt} \subseteq DS \subset \mathbb{R}^3 \tag{3-1}$$

As final output of TO, a set of B-Rep surfaces O are generated to create a visual representation of the optimal topology. S_{opt} is defined by the volume that these surfaces bound such that S_{opt} is equal to the boundary of O.

$$S_{opt} \approx bO \tag{3-2}$$

Although bO may represent the optimal solution in terms of performance, it is not usually economically feasible to manufacture a part based directly on bO. For this reason, a more simplified approximation of S_{opt} must be obtained through the shape recognition algorithm. This solution set will be known as S_{sim} to denote a simplified approximation of S_{opt}. S_{sim} is created through a sequence of Boolean operations between DS and a set of n solid features f_n obtained from the shape recognition algorithm. These features can represent either positive space or negative space such that by unifying them with the design space a material addition or subtraction can be performed. This process is represented mathematically below

$$S_{sim} = DS \bigcup f_1 \bigcup f_2 \bigcup \dots \bigcup f_n \tag{3-3}$$

The word *feature*, as used in this thesis, refers to a solid object constructed by sweeping a closed surface through CS cross sections. A cross section is made of m curves linked at their end points to create a closed loop (see Figure 3-2).

23
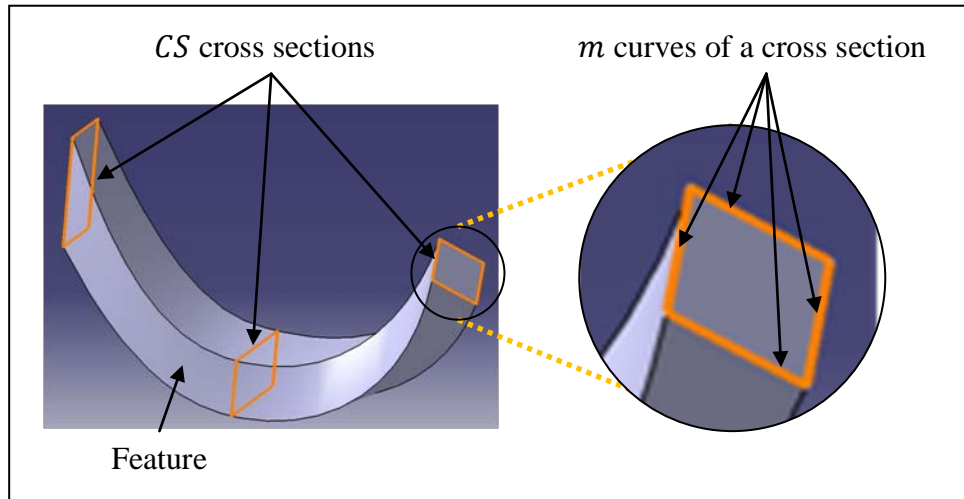
Figure 3-2: Definition of a feature

The geometric data needed to construct a *feature* is obtained through the shape recognition algorithm which forms the core contribution of this research. A subset of surfaces from O referred to as O_f to signify feature surfaces, is used as input to this algorithm. Figure 3-3 illustrates the relationship between DS, O, and O_f.
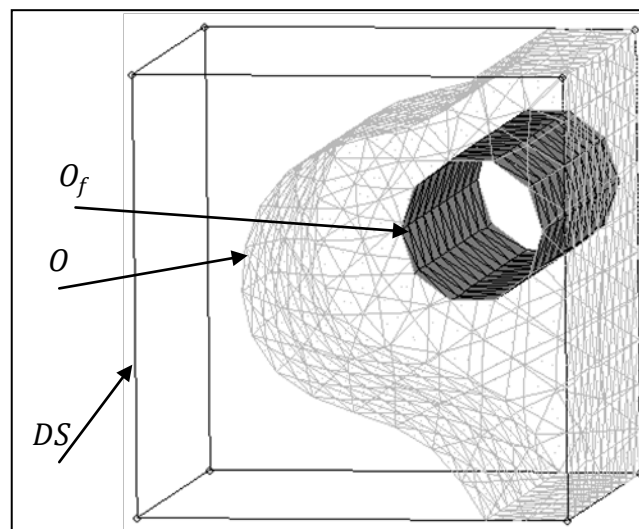
Figure 3-3: Design Space DS, Solution Set O, and Feature Surfaces O_f

In the figure, O has been identified through TO performed on DS as the optimal solution to the problem. O_f is a subset of O which defines a *feature*. The *feature* appears to be a cylindrical hole through the design space DS. To clarify, DS is a solid object and there is currently no hole in DS. Rather, O_f will be used as input to a shape recognition algorithm and an actual hole that approximates O_f, will be created as a result of the algorithm. This is then repeated for each *feature*.

3.2 **Shape Recognition Algorithm**

This section will step through the shape recognition algorithm in sequential order and explain the detailed methods employed in each step. We begin by defining the inputs to the overall algorithm.

3.2.1 **Inputs**

The shape recognition algorithm requires four inputs. They are introduced together here so as to familiarize the reader with the symbols used to represent the inputs and because they are the first data that the algorithm handles.

- Feature surface set O_f (required)

- Feature orientation geometry G (required)

- Number of cross sections CS (required)

- Design space intersection surface set DS_x (optional)

This thesis does not address how these inputs are obtained, but assumes that they are known and fed directly into the algorithm at the beginning. An example of how they may be obtained is presented, however, within the Implementation section (Chapter 4).

Feature Surface Set - The feature surface set was introduced above in section 3.1. It is used to define the subset of the optimal solution that the algorithm is trying to approximate.

Feature Orientation Geometry – The main objective of the algorithm is to generate cross sections that can be used to create a 3D feature within the design space. In order to orient these cross sections, a 2D plane is needed. The Feature Orientation Geometry, symbolized by G, is a generalized curve in $\mathbb{R}^3$ used to locate and orient the cross section planes of the feature. It is also used to control the path of the feature from one cross section to the next.
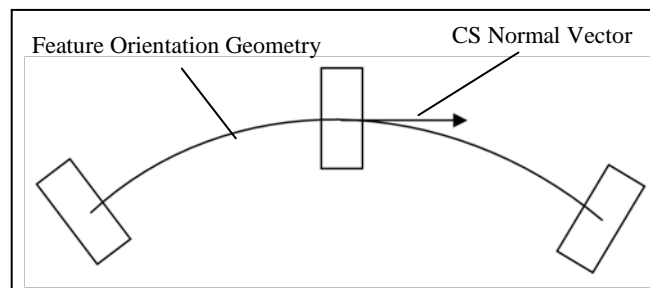
Figure 3-4: Feature Orientation Geometry

Number of Cross Sections – This defines the number of cross sections used to create the feature. The more cross sections used, the better the approximation to the optimal geometry, but the greater resulting model complexity and computational cost in subsequent steps of the design process.

Design Space Intersection Surface set – Many times when a feature is subtracted from the design space, it may intersect one or more surfaces of the design space in such a way that the topological entities of the design space are changed. The efficiency and accuracy of the

26

algorithm are greatly improved when this is identified at the beginning of the recognition process. Figure 3-5 depicts a common example of such an intersection. In the model on the left, the "Intersection surface" is one surface with 4 sides. After the *feature* is subtracted from the cube, the "Intersection surface" is split into two smaller surfaces.
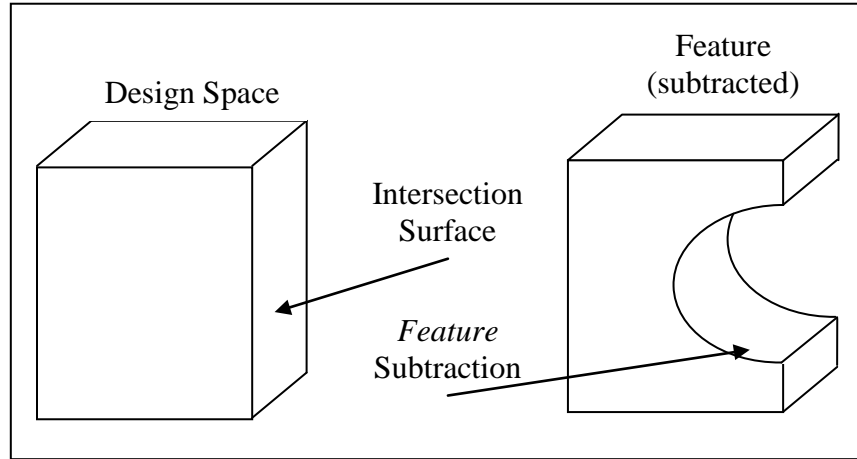
Figure 3-5: Design space intersection surface

3.2.2   Uniform Point Cloud Generation

The first step in the algorithm is to generate a uniform density point cloud on O_f. The reason for doing this is to eliminate any dependence on the surfaces from the TO run. This point cloud is generated by looping through the surfaces of O_f and generating random points on each surface. The locations of these random points are determined by a linear combination of the vertices (V) of the surface. The number of random points generated on each surface is dependent upon the area of that surface. This feature point cloud C_f is then represented by

$$C_f = f(A_{O_{fi}}, \rho, V_{O_{fi}}) \tag{3-4}$$

27

Where $A_{O_{fi}}$ is the area of the ith surface in the surface set O_f, ρ is a predetermined point per area density value, and $V_{O_{fi}}$ is the set of vertices associated with the ith surface of O_f.

Figure 3-6 (a) shows a set of surfaces representing O_f and (b) the points that would be generated on the surfaces during this step. The resulting point cloud (c) is then used for subsequent steps of the shape recognition algorithm thus eliminating the need to refer to the feature surfaces after this point.
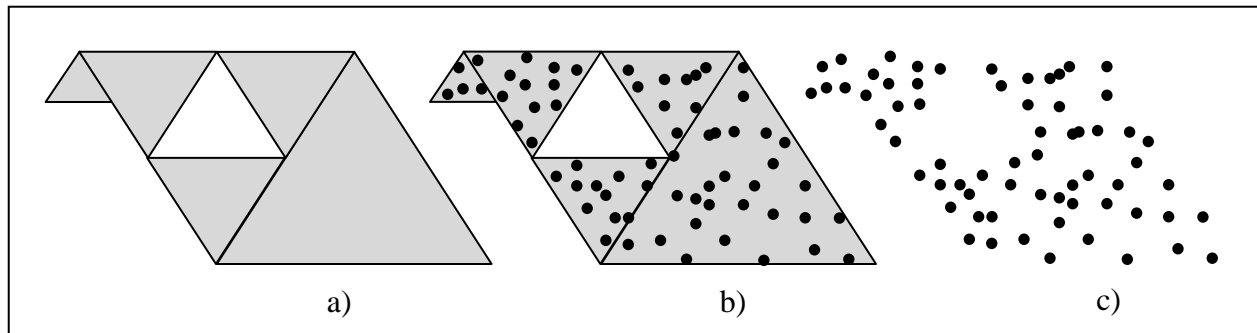
Figure 3-6: a) Example O_f, b) Points on O_f, c) Resultant feature point cloud C_f

3.2.3 Point Cloud Segmentation

The feature point cloud C_f is then segmented into CS smaller point clouds based on the orientation geometry G. To accomplish this, each point P_i in C_f is projected onto G along a normal vector to G as shown in Figure 3-7.

28
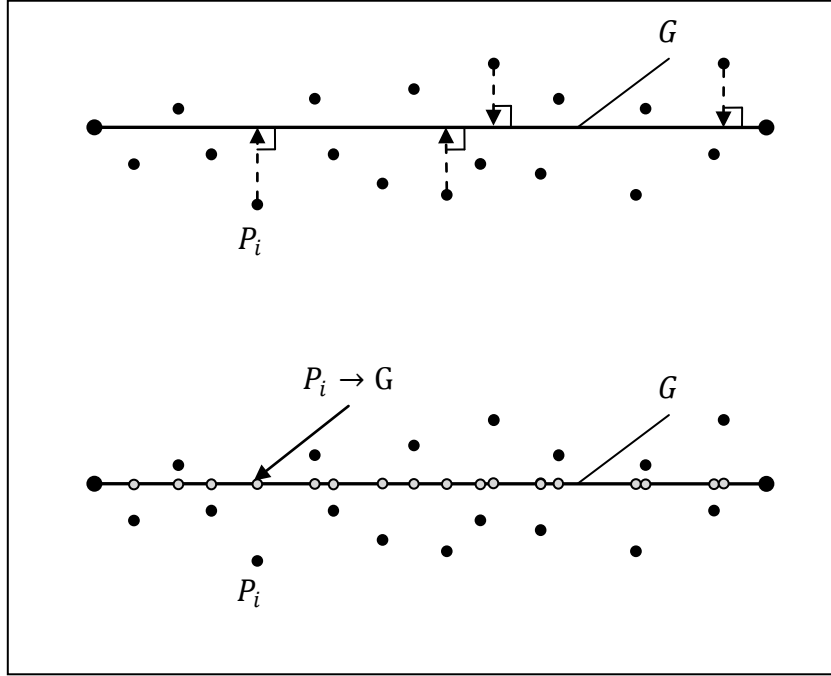
Figure 3-7: Projecting C_f onto G

The arc length S_i of the projection of $P_i \rightarrow G$ is then measured from the end point of G. Each point of C_f can then be sorted into separate cross section point clouds (C_j) based on its location along the arc length of G (represented as $\hat{G}$) according to the following equations where j is an integer between 0 and $CS - 1$.

$$\text{if} \quad j \cdot \frac{\hat{G}}{CS} \leq S_i < (j + 1) \cdot \frac{\hat{G}}{CS} \quad , \qquad P_i \in C_j \tag{3-5}$$

$$\text{if} \quad S_i = \hat{G} \qquad\qquad\qquad , \qquad P_i \in C_{(CS-1)} \tag{3-6}$$

For example, assume that the arc length of G is 10 ($\hat{G} = 10$) and the projection of a certain point P_i onto G is located at an arc length of 3 along G ($S_i = 3$), and the user has specified to use 4 cross sections to approximate the feature. Substituting all of these values into equation 3-5 indicates that P_i would be sorted into cross section point cloud C_1

$$1 \cdot \frac{10}{4} \leq 3 < (1 + 1) \cdot \frac{10}{4}$$

29

Figure 3-8 shows a simple example of the point cloud segmentation step just described assuming that $CS = 2$.
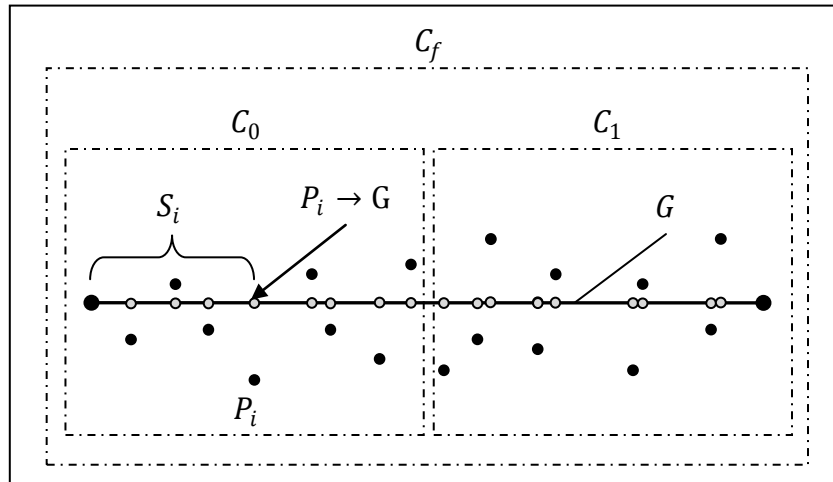
Figure 3-8: Segmentation of C_f into multiple cross section point clouds C_0 and C_1

3.2.4 Cross Section Point Cloud Projection

Now that there are CS cross section point clouds, each point cloud can be used to determine the cross section of the feature in that general vicinity. This is done by projecting the point cloud onto a 2D plane. This plane will be denoted by ΔC_j representing the projecting plane of cross section point cloud j. To locate and orient ΔC_j in space, a point (D) and plane normal vector $(\vec{N})$ must first be identified.

It is ideal if the plane is located as close to the center of the point cloud as possible so that the distance any one point is projected is minimized, which minimizes the approximation error introduced by projecting the points in the first place. To do this, the same arc lengths computed in the previous step (S_i) are subtracted from one another until the two points $(P_{min}$ and $P_{max})$

30

that are furthest apart along G are located, yielding ΔS_{max}. D is then located starting at the projection of $P_{min} \to G$ and moving along G the arc distance $\frac{\Delta S_{max}}{2}$ as shown in Figure 3-9 a.
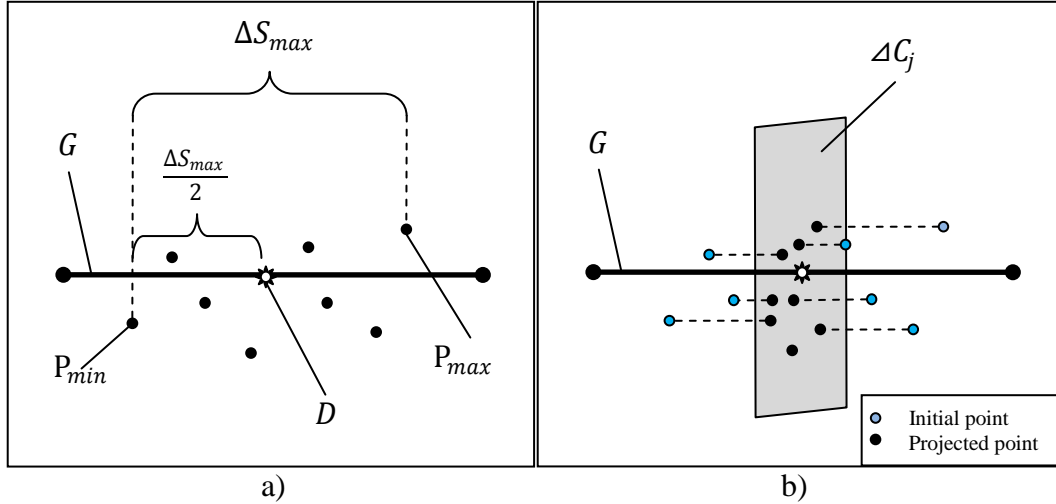
Figure 3-9: a) Locating a point to define $\varDelta C_j$, b) projecting C_j onto $\varDelta C_j$

Once D is located, the tangent vector of G at point D ($\vec{T}_{G_D}$) is used as the normal vector for the projecting plane $\varDelta C_j$. Each point in C_j is then projected along $\vec{T}_{G_D}$ to $\varDelta C_j$. The projected cross section point cloud will be distinguished by $\overrightarrow{C_j}$. An example of this projection is seen in Figure 3-9 b.

3.2.5 Polar Mapping

In order to compare the point data to Shape Templates (defined in subsequent sections) a polar map of $\overrightarrow{C_j}$ must be constructed. This is done by first obtaining an arbitrary reference point (A) from which to measure the polar coordinates of each point in $\overrightarrow{C_j}$. A point near the center of the point cloud is best in order for the algorithm to maintain mathematical stability, but any reference point on $\varDelta C_j$ will theoretically work (see Figure 3-10 a). The polar coordinates (r, θ)

31

of $\overrightarrow{C_j}$ with respect to A are then measured and the points are sorted into θ "bins" which represent groupings of points from $\overrightarrow{C_j}$ with similar θ coordinates. The points in each θ bin are then averaged together so that there is one point representing each θ bin as seen in Figure 3-10 b. The resulting averaged point cloud will be symbolized by $\overline{\overrightarrow{C_j}}$ to represent the averaged, projected cross section point cloud. The polar coordinates of $\overline{\overrightarrow{C_j}}$ are then measured and recorded as shown in Figure 3-10 c.
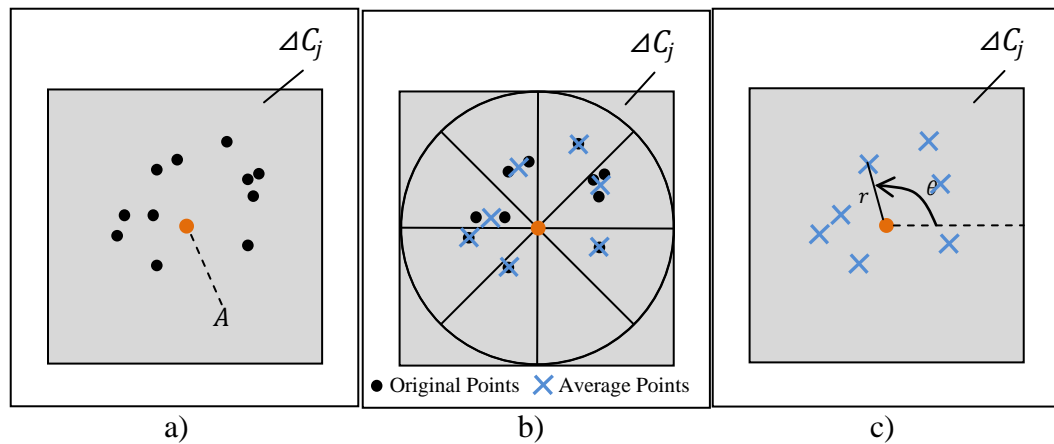
Figure 3-10: a) Reference point *A*, b) Average points, c) Polar measurement

Plotting the polar measurements from the previous step yields a polar map of the periphery of the shape as described by Lin and Chao [11]. The set of points comprising this polar map will be referred to throughout the rest of this thesis as $C_j^{'}$. Figure 3-12 presents several polar maps of known shapes as visual reference for the reader. As can be seen in the figure, the max and min points on $C_j^{'}$ represent key geometric data needed to represent a given shape.
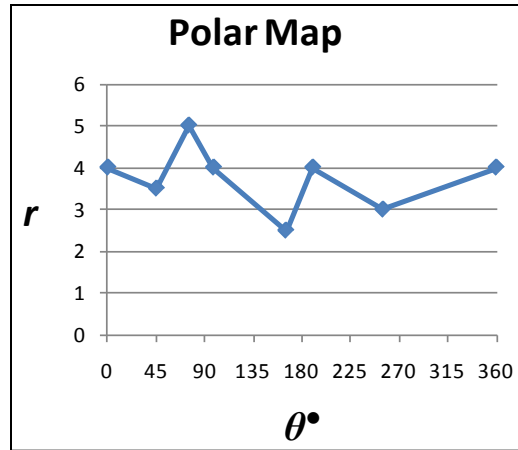
Figure 3-11: Polar map of sample data

a) circle   b) equilateral triangle   c) square   d) isosceles right triangle
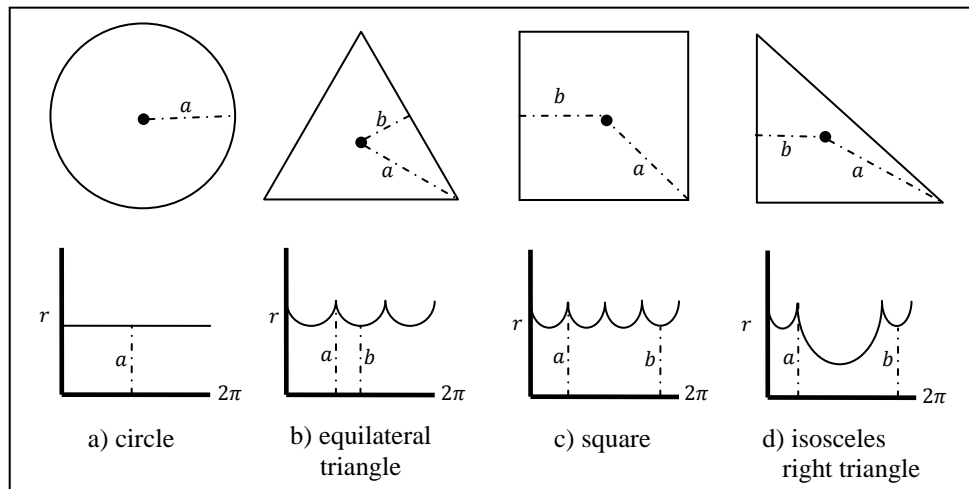
Figure 3-12: Polar maps of known shapes

3.2.6   Finding the Peaks

At this point in the algorithm, the points within $C_j^{'}$ with maximum r values need to be identified so that $C_j^{'}$ can be compared to different pre-defined shape templates in a subsequent step. This section describes how this is to be done.

33

The most complex shape template that has been defined will determine pk_{tar}, the target number of peaks (or local maxima) that will be identified from $C_j^{'}$. For example if the most complex template defined is a 5 sided polygon, then $pk_{tar} = 5$ because 5 local maxima will be needed from $C_j^{'}$ in order to define a 5 sided polygon. The number of peaks actually identified by the algorithm is pk_{act}.

The search algorithm proceeds through all of the points in $C_j^{'}$ one by one to determine if that point is a local maximum. Let the single point under consideration at any given time be called P_{cur}. The search algorithm compares the r values of all the points within a given θ range (rng) before and after P_{cur}. If the r value of P_{cur} is the largest among this subgroup of $C_j^{'}$ it is declared as a peak and pk_{act} is incremented by one. The algorithm then moves to the next point and repeats this process.

Figure 3-13 illustrates some of the details of the peak search algorithm. If P_{cur} is Point 4 in Figure 3-13 (indicated by a circle), and $rng = 1$, then the r values of Points 3, 4, and 5 are compared to each other. If Points 4 has the highest r value among the points included in the comparison, then it is considered a maximum. In this case, Point 4 is a maximum so pk_{act} is incremented by 1. If $rng = 3$, however, then Point 4 would no longer be considered a maximum because Point 7 would be included in the r value comparison and it has a higher r value than Point 4. The point with the next higher θ value (in this case Point 5) is then considered to be P_{cur} and the same process is repeated. In this example, with rng equal to 1 or 3, Points 4 and 8 are identified as maxima.

After each point in $C_j^{'}$ has been considered, if $pk_{act} > pk_{tar}$ then rng is incremented by one and each point is considered again within a larger range of neighboring points until $pk_{act} < pk_{tar}$ at which point the set of peaks found in the previous iteration is used.

34
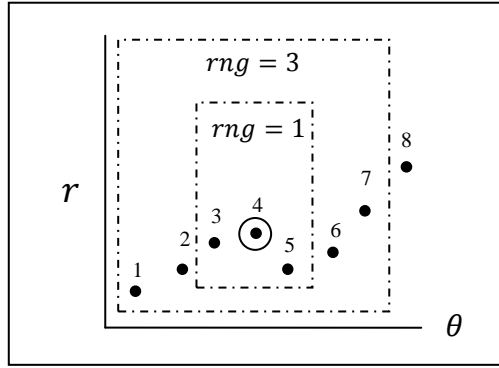
Figure 3-13: Peak search method

This process is shown graphically in Figure 3-14 using a sample data set from a triangle. Notice that there is a bit of "noisy data" within the sample at $\theta = 180$. This is a typical problem contained within TO results. When the value of rng is low, this noise is mistaken for a maximum. However, as the value of rng increases, these mistaken maxima are filtered out.

In the final iteration, shown in Figure 3-14 e, pk_{act} is less than pk_{tar}, therefore the 3 maxima identified in d (the previous iteration) are used for template generation and fitting.
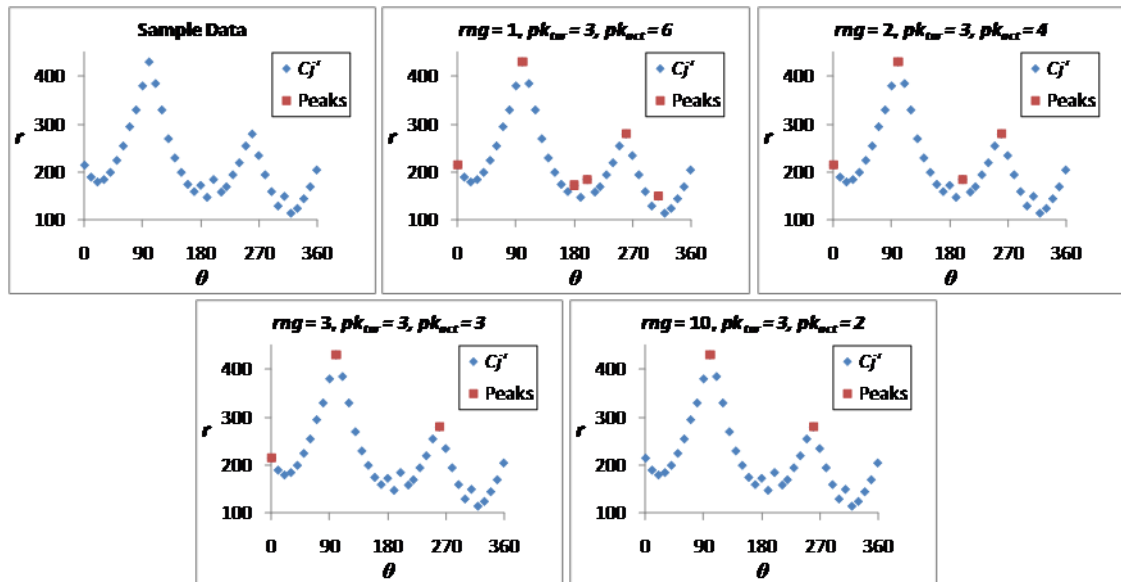
Figure 3-14: Peak search algorithm

35

3.2.7 Parametric Shape Template Generation

A Shape Template ST, for purposes of this research, is a two dimensional, closed, convex shape of n sides. Each side of the shape is defined in polar coordinates by a function of the form $r = g(\theta)$. Thus, the entire shape can be defined by a piecewise function $G(\theta)$.

$$r = G(\theta) = \begin{cases} g_1(\theta), \theta_0 \leq \theta < \theta_1 \\ g_2(\theta), \theta_1 \leq \theta < \theta_2 \\ g_3(\theta), \theta_2 \leq \theta \leq \theta_3 \\ \quad\vdots \qquad\qquad \vdots \\ g_n(\theta), \theta_{n-1} \leq \theta \leq \theta_n \end{cases} \qquad \sum_0^n \theta_n = 2\pi \qquad (3\text{-}7)$$

Each function g_n represents one side of ST and can be compared to the sample data between two peaks identified in the previous step. This is done by plugging the θ values of pk_{act} into the piecewise function limits in Equation (3-7). Then, for each (r, θ) pair in $C_j^{'}$, a comparison r value is generated from the template equations. It is then a trivial matter to subtract the template r values from the $C_j^{'}$ r values to calculate the 2D approximation error ε_{2D} of that specific template using those specific peaks. ε_{2D} for each possible combination of peaks is calculated and the combination of peaks which minimizes ε_{2D} is identified. This process is a simple exhaustive search optimization method. Any other defined templates are then processed in the same way.

To illustrate how a parametric shape template is generated we will now look at an example. Assume that there is a shape template defined with 4 sides. Sides 1 through 3 are straight lines, with side 4 being an arc as shown in Figure 3-15.

36
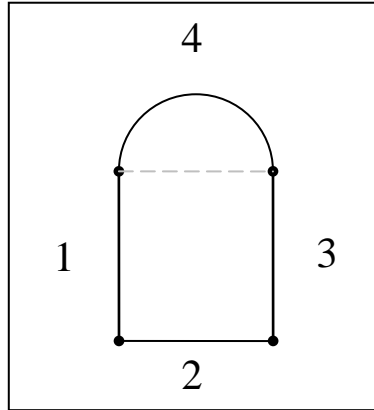
Figure 3-15: Example shape template

The equation of a line in Cartesian space is

$$y = mx + b \tag{3-8}$$

Then, substituting the standard polar transformation equations in for *x* and *y* and solving for *r*

$$y = r\sin\theta \tag{3-9}$$
$$x = r\cos\theta \tag{3-10}$$

We end up with the polar equation of a line where *m* and *b* are the slope and *y*-intercept, respectively, of the line in Cartesian space.

$$r = \frac{b}{\sin\theta - m\cos\theta} \tag{3-11}$$

The shape template equations g_1, g_2, and g_3 would be of the same form as Equation 3-11 but with differing values for *m* and *b* since these pieces of the template are all straight lines. To get the general form of g_4, we take the general equation of a circle and, in a similar manner, substitute for *x* and *y* using Equations 3-9 and 3-10. This results in Equation 3-12.

37

$$r = \frac{2r_0 \cos(\theta - \varphi) \pm \sqrt{[2r_0 \cos(\theta - \varphi)]^2 - 4(r_0^2 - a^2)}}{2} \qquad (3\text{-}12)$$

where (r_0, φ) is the polar location of the circle center and a is the radius of the circle. In the case of the example template shown in   Figure 3-15, the center of the circle must lie on the line formed between the end points of lines 3 and 1. This means that a is equal to half the length of the gray dotted line in Figure 3-15. The full shape template definition as presented in Equation 3-7 is

$$r = G(\theta) = \begin{cases} \dfrac{b_1}{\sin \theta - m_1 \cos \theta}, & \theta_0 \leq \theta < \theta_1 \\[2ex] \dfrac{b_2}{\sin \theta - m_2 \cos \theta}, & \theta_1 \leq \theta < \theta_2 \\[2ex] \dfrac{b_3}{\sin \theta - m_3 \cos \theta}, & \theta_2 \leq \theta \leq \theta_3 \\[2ex] \dfrac{2r_0 \cos(\theta - \varphi) \pm \sqrt{[2r_0 \cos(\theta - \varphi)]^2 - 4(r_0^2 - a^2)}}{2}, & \theta_3 \leq \theta \leq \theta_4 \end{cases} \qquad (3\text{-}13)$$

Now assume that Figure 3-16 a) represents the sample data from a model. Images b, c, d, and e illustrate graphically what occurs when the different peak θ values are substituted into Equation 3-13. It is obvious that the combination of substitutions made which yielded image e is the best orientation for the Shape Template geometry.

This process is repeated for each cross section point cloud and each defined shape template. This should result in a set of templates being generated for each cross section along with the 2D approximation error measurement ε_{2D} associated with each template.
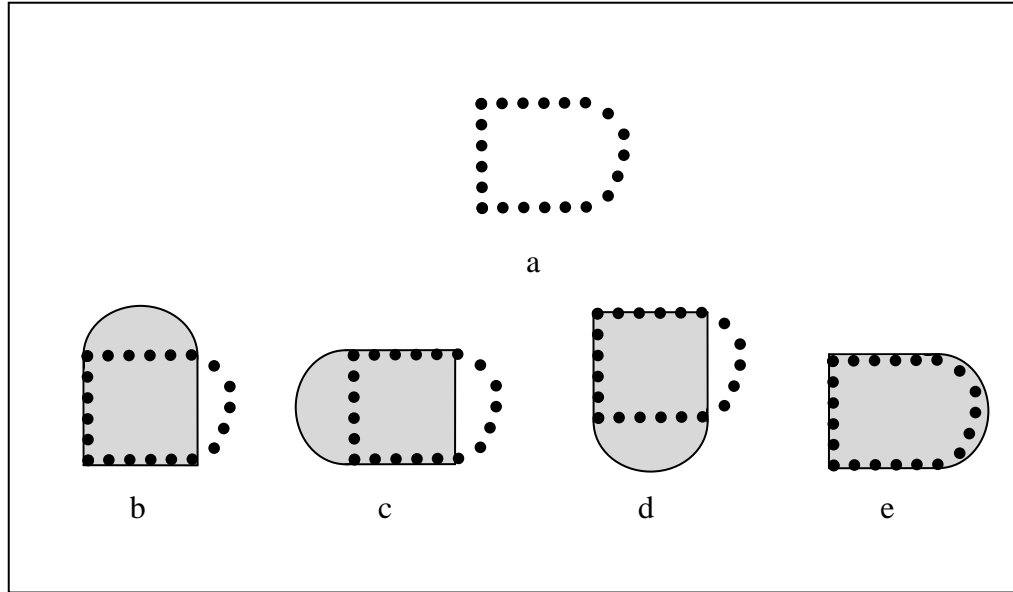
Figure 3-16: Parametric shape templates with different combinations of peaks

3.2.8 Template Selection

At this point in the design process it is advantageous to bring the designer in the loop. It may be that manufacturing capabilities or constraints are such that the designer would like to use a specific template to create the feature regardless of optimal geometric fit. Only the human designer can decide the tradeoff between model complexity and model fitness for every situation. In accordance with this fact, the designer is presented with the findings of the algorithm up to this point which include the 3D approximation error ε_{3D} for each template and the number of needed parameters $Parm$ to represent a given template as a measure of complexity. ε_{3D} is calculated by adding all the 2D approximation errors of each cross section associated with a given template as shown in equation (3-8).

$$\varepsilon_{3D} = \sum_{0}^{CS} \varepsilon_{2Dn} \tag{3-13}$$

39

Table 3-1: Fitness and Complexity of Defined Templates

Template	ε_{3D}	# Parameters
Template 1	ε_{3D1}	$Parm_1$
Template 2	ε_{3D2}	$Parm_2$
Template 3	ε_{3D3}	$Parm_3$
Template n	ε_{3Dn}	$Parm_n$

Once the designer has indicated which template is best based on the tradeoff between fitness and complexity, it can be used to create actual geometry.

3.2.9 Geometry Creation

The geometry creation portion of this process is accomplished through CAD API programming and is not part of the methods included in this research.

40

4   IMPLEMENTATION AND IMPLEMENTATION RESULTS

In order to demonstrate and test the validity of the concepts and methods just discussed, this chapter is dedicated to a sample implementation. At the end of this chapter, results from implementing the methods on four complete models along with their fitnesses will be presented. This implementation is not all-encompassing in the scope of the research, but is intended to show the efficacy of the methods in a few simple cases. To make the implementation tractable within the scope of this research, a few simplifying constraints have been put in place. The functions g_1 through g_n defining a shape template according to Equation 3-7, are limited to 1st order functions of θ. In other words the, shape templates implemented are n-sided piecewise linear polygons. Similarly, the orientation geometry G is limited to 1st order B-Spline geometry, or in other words, straight lines.

The implementation presented here will show the recognition and construction of a single feature of a model of a cantilevered cube under a distributed load on the top surface (Figure 4-1). This process mirrors that presented in Figure 3-1 in the box labeled "Feature Algorithm". This is indicative of the actual process that would be repeated for each feature of the model, but is only shown for one feature. The entire implementation occurs within CATIA V5 R18, hereafter referred to as CATIA, and utilizes the CAA RADE C++ API to accomplish tasks within the CAD application.\

41
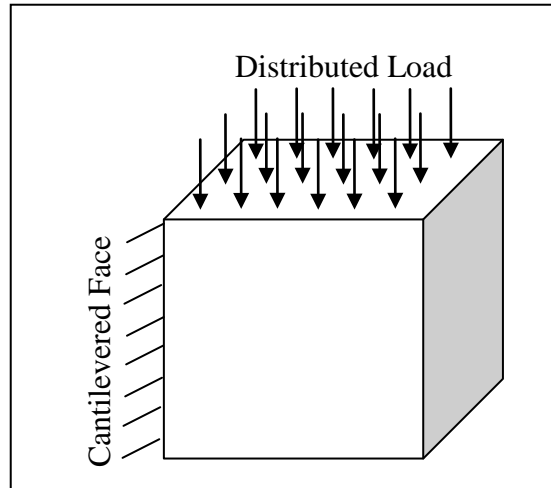
Figure 4-1: Case study for implementation

4.1 **Environment**

The process begins with setting up the environment. This is done by importing both the design space (DS seen in Figure 4-2 a) and the output surface set from TO (O) into CATIA. The translucency of DS can be adjusted so that both DS and O can be seen simultaneously as shown in Figure 4-2 b.
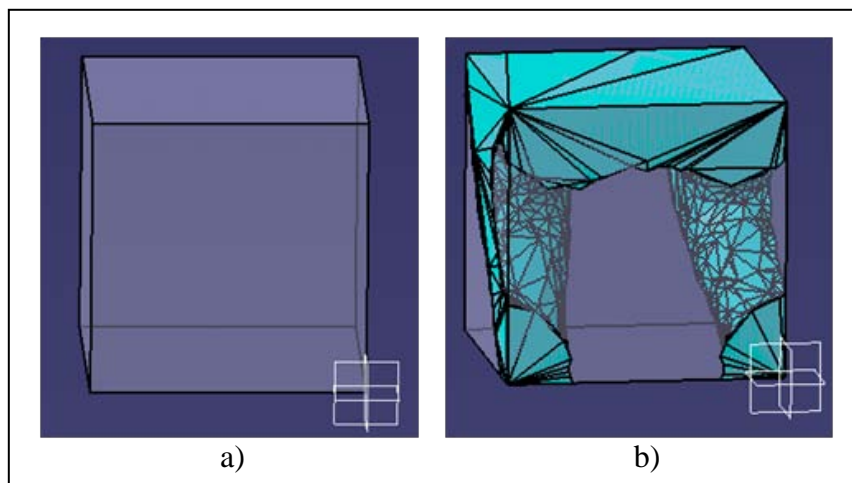
Figure 4-2: a) DS, b) DS and O with translucency

4.2 **Inputs**

Several inputs introduced in section 3.2.1 are obtained from the user through a Graphical User Interface (GUI) including the direction vector defining G and the number of cross sections to use to approximate the feature (CS). The GUI also displays the file path of DS and allows the user to specify which of the defined shape templates should be used to approximate the feature. When the "Auto" option is selected, the shape template that minimizes approximation error is used automatically. Figure 4-3 shows the GUI and the inputs used for this example. For purposes of demonstration a shape template defining a circle, a triangle, a quadrilateral, and a pentagon were defined. In this example G is a straight line oriented in the Y-Direction, 2 cross sections will be used, and the best fit template will be used to create geometry.
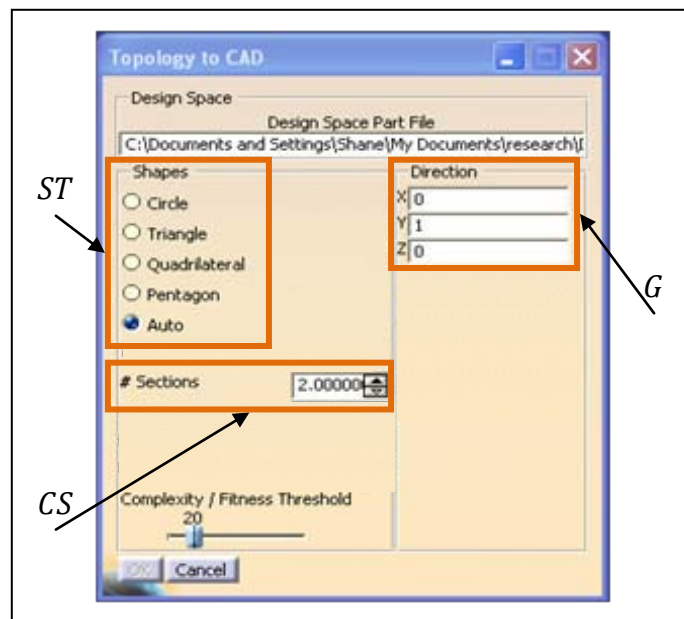
Figure 4-3: Graphical User Interface

To obtain the other inputs including the feature surfaces O_f, and the design space intersection surfaces DS_x, the user must use an interactive selection trap within the CAD

environment. Through this process the user can specify which surfaces from O are to be included in O_f and which surfaces of the design space the feature intersects (DS_x). These steps are shown in Figure 4-4. The polygon trap user selection method shown in Figure 4-4 b uses a polygon drawn freehand by the user and extrudes that polygon along the direction of the current view frame. Any object that is completely contained within that extrusion is considered "selected" and will be included in O_f.

| a) | b) | c) |
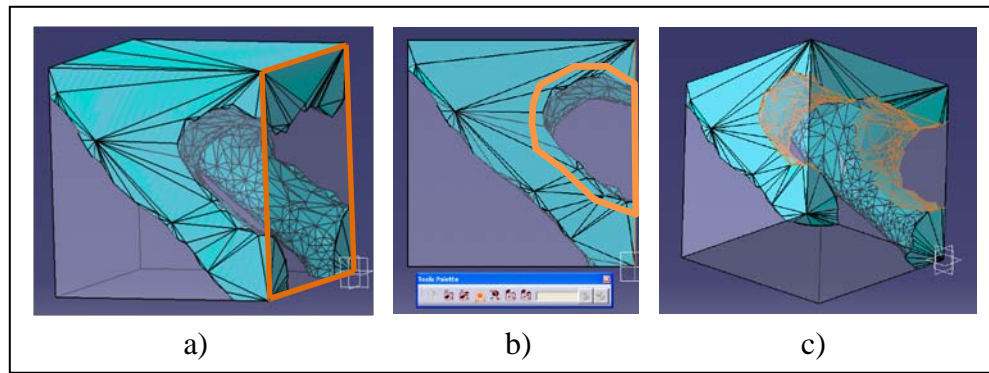
Figure 4-4: User selection of DS_x (a) and O_f(b). c) Resulting selected surfaces of O_f

All of the user interaction used to gather the inputs of the algorithm are dependent on CATIA CAA RADE object classes.

4.3 **Uniform Point Cloud Generation**

To generate the uniform feature point cloud C_f from the feature surfaces O_f, the vertices of each surface must be accessed. To do this a "cell list" is retrieved from each surface object of SurfList. The cell list contains all geometric object data for the surface including points representing the vertices, lines connecting the points that form the boundaries of the surface, and the surface itself. Retrieving the surface from the list is accomplished via the CATPathElement

44

object. Next the surface must be recast to a `CATBaseUnknown` object type in order to use the `CATIMfGeometryAccess` interface to access the geometric objects and store them in a list structure (`CellList`).

```
for(int i=0; i<SurfList->GetSize(); i++)
{
        CATPathElement * mPath = NULL;
        mPath = (CATPathElement*) (*SurfList)[i];
        CATBaseUnknown *spUnknownObj = mPath->CurrentElement();
        CATLISTV(CATBaseUnknown_var) CellList;
        CATIMfGeometryAccess* GeomAcc = NULL;
        spUnknownObj-> QueryInterface
                    (IID_CATIMfGeometryAccess,(void**)&GeomAcc);
        GeomAcc->GetCells(CellList);
```

Then, the geometric dimension of each item in `CellList` is observed to determine if it is a line (dimension 1) or a surface (dimension 2). If it is a line, then the end points represent the vertices of the surface and are stored in double arrays `a`, `b`, and `c`. If it is of dimension 2, it must be the surface face itself and the area of the face is extracted and stored in a variable called `area`.

```
for (int j=1; j<=CellList.Size(); j++)
{
        CATCell_var tmpCell = CellList[j];
        short dim = tmpCell->GetDimension();
        if (dim == 1)
            {
                    //Extract point data into a, b, and c
            }

        if (dim == 2)
            {
                    CATFace_var face = tmpCell;
                    area = face->CalcArea();
            }
}
```

Once the above routine is complete, the vertex data along with the surface area are known and the point cloud can be generated. First the number of needed points is determined based on the surface area and the point density variable obtained in an earlier step from the user. The random point is then appended to `HolePoints` which is a data structure used to store all points of C_f.

45

```
                int numPoints = (int)(area*pDens);
                if (numPoints<1) numPoints = 1;

                for (int j=1; j<numPoints; j++)
                {
                        //generate 2 random numbers
                        double r1 = (rand()%10001)/10000.0;
                        double r2 = (rand()%10001)/10000.0;

                        //generate random point coord on the triangle
                        ArrMult(a,(1-sqrt(r1)));
                        ArrMult(b,(sqrt(r1)*(1-r2)));
                        ArrMult(c,(sqrt(r1)*r2));

                        ArrPlus(d,a,b);
                        ArrPlus(p,c,d);
                        //this results in p = a+b+c
                        HolePoints.Append(p);
                }
```
ArrMult and ArrPlus are defined as

```
        void TTCStateCommand::ArrPlus(double point[3], const double
        left[3], const double right[3])
        {
                point[0] = left[0] + right[0];
                point[1] = left[1] + right[1];
                point[2] = left[2] + right[2];
        }

        void TTCStateCommand::ArrMult(double arr[3],double scale)
        {
                arr[0] = arr[0]*scale;
                arr[1] = arr[1]*scale;
                arr[2] = arr[2]*scale;
        }
```
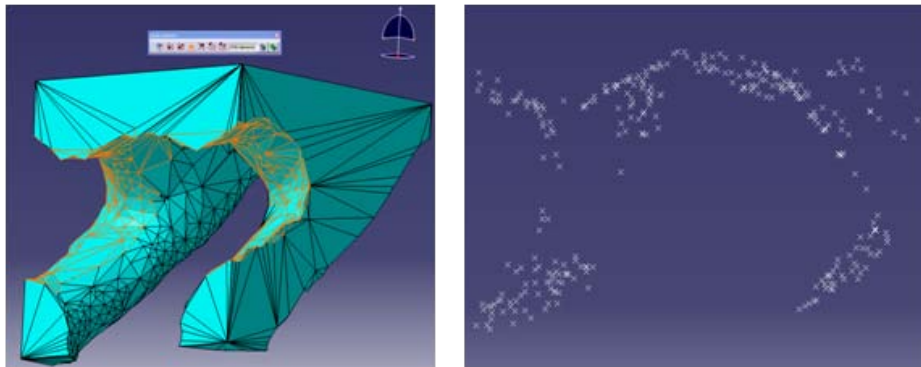
The result of this step is shown graphically in Figure 4-5.

Figure 4-5: Uniform point cloud generation results

46

4.4 Point Cloud Segmentation

To segment the point cloud into multiple cross section point clouds each point in HolePoints is projected onto G, which is equal to the Y-axis in the example. The max and min Y value of all points in HolePoints is used to separate the HolePoints into two point clouds. The result of this step is shown in Figure 4-6.
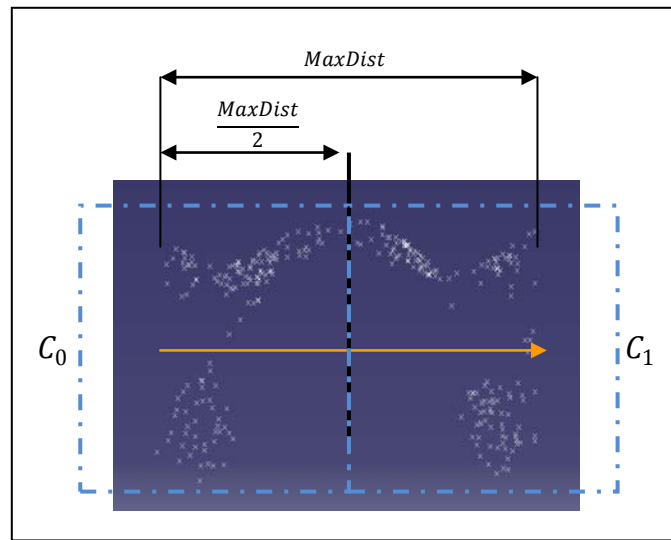
Figure 4-6: Point cloud segmentation

4.5 Cross Section Point Cloud Projection

Due to the simplification employed in this implementation, of using a straight line to represent G, the normal vector of the projecting plane ΔC_j is equal to the Y-axis direction. In order to locate the projecting plane, all the points in C_j (CSPointList) are averaged together to yield a reference point AveCSPnt.

```
for ( int i=1; i<= CSPointList.Size(); i++)
{
      AveCSPnt+= CSPointList[i];
}
AveCSPnt = AveCSPnt/CSPointList.Size();
```

47

The CATIA CAA RADE API includes many object classes that are designed for quickly doing common geometric operations. Two of these objects that are used at this point and throughout the implementation are `CATMathVector` and `CATMathPoint`. The following code demonstrates how member functions and operators of these two classes help to project the point cloud onto the projecting plane.

```
for ( int i=1; i<= CSPointList.Size(); i++)
{
        CATMathPoint* tmpPoint = new CATMathPoint(CSPointList[i]);
        CATMathVector cur2ave = CSPointList[i]-AveCSPnt;
        CATMathVector radVec = ((*GenDir^cur2ave)^*GenDir);
        *tmpPoint = *tmpPoint - (cur2ave*(*GenDir))*(*GenDir);
}
```

First a `CATMathPoint`, `tmpPoint`, is created based on the current cross section point. Then a `CATMathVector cur2ave` is instantiated by subtracting the average point from the current point. In the code snippet, `GenDir` is a unit vector in the direction of G. By taking the cross product (symbolized by `^` in the `CATMathVector` class) of `GenDir` and `cur2ave` followed by the cross product of `GenDir` again, the projection of `cur2ave` onto the projecting plan is obtained and named `radVec`. If this is the first point to be projected then it represents $\theta = 0$ and a new coordinate system (`XVec` , `YVec`) is defined based on this projected vector.

```
if (i==1)
{
        YVec = radVec;
        XVec = (*GenDir)^YVec;
        Theta = 0;
}
```

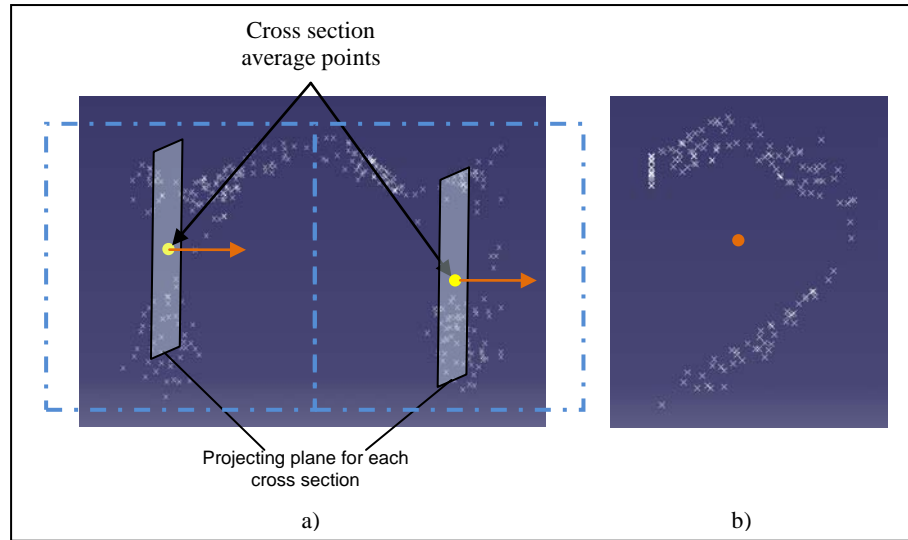The resultant 2D point cloud is shown in Figure 4-7.

48

Figure 4-7: a) Projecting planes, b) 2D projected point cloud

4.6 **Polar Mapping**

To determine the polar coordinates of each point in the current CSPointList, each point's component in the XVec and YVec directions is obtained and the angle theta is determined based on the following equation

$$\theta = \cos^{-1}(radVec \cdot YVec) \tag{4-1}$$

When using cos^{-1} however, it is important to know which quadrant the point lies in with special attention being given to when the point lies directly on the axis line. To manage these issues the following code was implemented.

```
double yCheck = radVec*YVec;
double xCheck = radVec*XVec;

//when the point is on an axis line
if(yCheck > .99999999999) Theta = 0;
else if(yCheck < -.99999999999) Theta = 2*pi;
else if(xCheck < -.99999999999) Theta = 3*pi/2;
else if(xCheck > .99999999999) Theta = pi;
//1st quadrant case
else if(yCheck > 0 && xCheck > 0) Theta = acos(radVec*YVec);
```

49

```
//2nd quadrant case
else if(yCheck < 0 && xCheck > 0) Theta = pi - acos(radVec*(-
                                                    1*YVec));
//3rd quadrant case
else if(yCheck < 0 && xCheck < 0) Theta = pi + acos(radVec*(-
                                                    1*YVec));
//4th Quadrant case
else Theta = 2*pi - acos(radVec*YVec);
```

Finally the r value of the points can be determined by taking the length of `radVec` and the θ bin that the point lies in can be determined based on the number of θ bins being used. This example uses an `interval` of 1 which creates 360 θ bins.

```
rad = radVec.Norm();
radVec.Normalize();
```

Each point is separated into its respective bin by dividing θ by `interval` and truncating the result.

```
int bin = floor(Theta / interval);
UniDistPoints.at(bin).push_back(tmpPoint);
```

Notice that the points are stored in a 2-dimensional vector (`UniDistPoints`) with the top level having a set length of 360, while the 2nd level of vectors are dynamically allocated. This makes averaging the different points in each bin a trivial nested loop and results in the creation of an average bin point (`binAvePnt`). A simplified graphical representation of this concept is shown in Figure 4-8.

The average bin point `binAvePnt` is then stored in another vector of set length 360 along with the radial distance r of `binAvePnt` called `curRad`.

```
pnts360.erase(pnts360.begin()+j);
pnts360.insert(pnts360.begin()+j,binAvePnt);

rads360.erase(rads360.begin()+j);
rads360.insert(rads360.begin()+j,curRad)
```
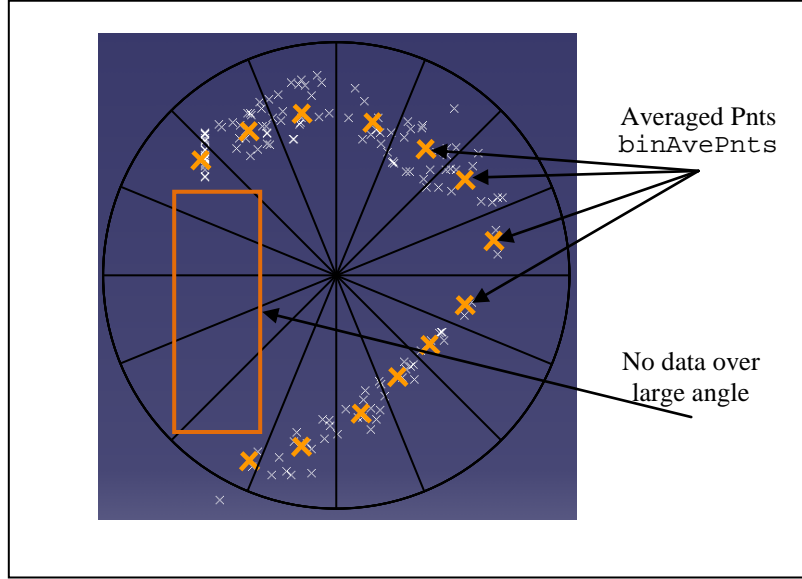
Figure 4-8: Averaged θ bin points

Many times there are large angles that have no point data to represent the cross section. This is usually due to the fact that the feature intersects the design space, as in our example case. Figure 4-8 shows this lack of point data along the left edge of the cross section. The algorithm senses this issue during the θ bin averaging by keeping track of consecutive θ bins that have no point data in them to average. When a set number of consecutive θ bins are detected with no point data, each of the user specified intersection surfaces DS_x are tested to see if the current θ direction intersects any of these surfaces. If there is an intersection, a point is generated at the intersection and added to $\overrightarrow{C_j}$ to represent that angular position of the polar map of the cross section. The implementation of this process is shown below where `SpecListOfFaces` is DS_x, and `spSearchDir` is the direction of the current θ bin. First, a copy of the cross section reference point `AveCSPnt` is projected in `spSearchDir` direction until it intersects one of the surfaces in `SpecListOfFaces`.

51

```
for (int j=0; j<pnts360.size(); j++)
{
        //if there is a data gap around theta bin j, check for
        //intersections with design space surfaces
for (int k=1; k<=SpecListOfFaces.Size(); k++)
{
        //create the projection of AveCSPnt on each face unless it fails
        CATISpecObject_var PntToProj = CreateGSMPoint(AveCSPnt,false);
        CATIGSMProject_var projection = spGSMFactory->
            CreateProject(PntToProj,SpecListOfFaces[k],spSearchDir,FALSE);
        CATISpecObject_var spProj = projection;
        HRESULT hr;
        CATTry
        {
                hr = spProj->Update();
        }
        CATCatch(CATMfErrUpdate,error)
        {
                printf("No intersection with design space surfaces\n");
                continue;
        }
        CATEndTry;
```

If the projection does not update correctly, there must not be an intersection with the design

space surface and the algorithm moves on to the next surface. If it was successful, then the

CATIGeometricalElement interface is used to access the CATBody result of the projection.

```
        CATBody_var spBody = NULL_var;
        CATIGeometricalElement_var spGeoEle = NULL_var;
        spGeoEle = spProj;
        spBody = spGeoEle->GetBodyResult();
```

Then, the vertices are retrieved from the CATBody, a point is created (GSMPoint) and finally

added to pnts360.

```
        CATLISTP(CATCell) listVertexCells;
        spBody->GetAllCells(listVertexCells,0);

        for ( int i = 1; i <= listVertexCells.Size(); i++)
        {
        CATVertex *aVertex = (CATVertex*) (listVertexCells[i]);
        CATPoint *aCatPoint = aVertex->GetPoint();
        CATMathPoint* tmp_pt = new CATMathPoint(0,0,0);
        aCatPoint->GetMathPoint(*tmp_pt);
        if(CreatePnts360) CreateGSMPoint(*tmp_pt);
        pnts360.erase(pnts360.begin()+j);
        pnts360.insert(pnts360.begin()+j,tmp_pt);
        }
} //(end of for loop k)
} //(end of for loop j)
```

In our example, nine intersection points were added. With this addition, the averaged points from our example can be seen in Figure **4-9** along with its corresponding polar map.
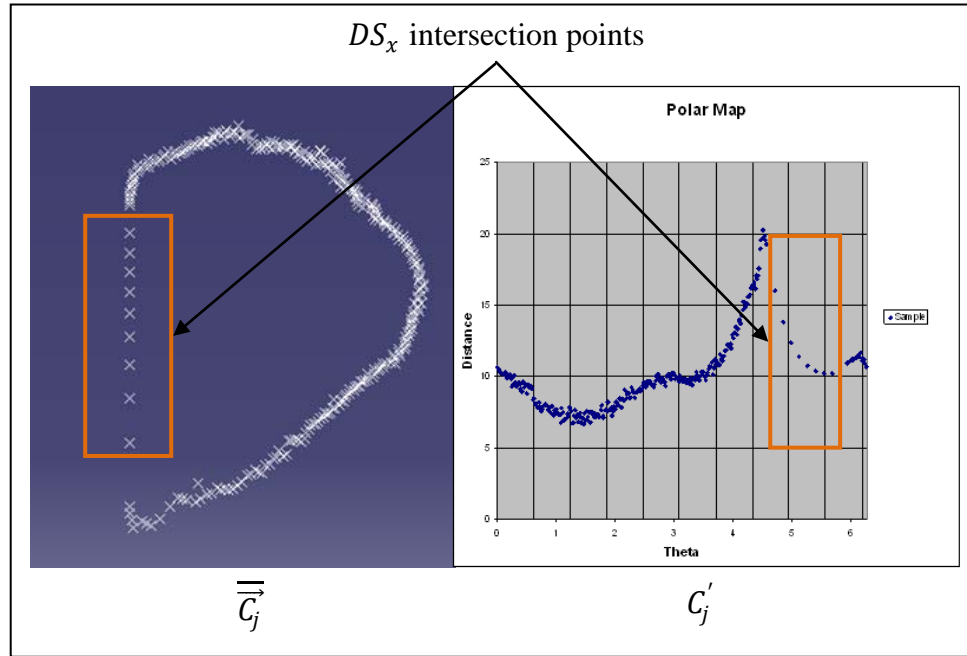

Figure 4-9: Cartesian points with corresponding polar map

4.7 **Finding the Peaks**

The following code shows the implementation of the routine described in section 3.2.6 used to identify "peak" points in the polar map C_j'. In this code `checkPeakNum` $= pk_{tar}$, `numPeaks` $= pk_{act}$, and `range` $= rng$.

```
while (checkPeakNum > numPeaks && range < 150 )
{
       checkPeakNum = 0;
       range = 0;
       peaks.clear();
       peaks.resize(rads360.size(),NULL);
       range++;

       for (int i=0; i<rads360.size(); i++)
       {
               if (rads360[i] == 0) continue;
               bool isMax = TRUE;
```

53

```
            //get all rads within a certain range of the current point
            std::vector<double> nearPnts;
            for (int j=-range; j<=range; j++)
            {
                    nearPnts.push_back(rads360[(i+j+rads360.size())%
                                                    rads360.size()]);
            }

            //if it is the max R within range, mark it as a peak.
            double maxR=rads360[i];
            for (int k=0; k<=range*2; k++)
            {
                    if (k==range) continue;
                    if (maxR <= nearPnts[k]) isMax = FALSE;
            }
            if(isMax)
            {
                    peaks.erase(peaks.begin()+i);
                    peaks.insert(peaks.begin()+i,rads360[i]);
                    checkPeakNum++;
            }
        }
    }
```

It is important to note the data structure used in this portion of the code. When a peak is identified, the r value is stored in a vector called `peaks` which has a set size equal to that of `rads360` or `points360`. The r value is stored in the same position within `peaks` as the r value obtained from `rads360`. This makes identifying peak points in $C_j^{'}$ in later steps a trivial matter. If `peaks[i]` is not empty, then `rads360[i]` and `i` represent the (r, θ) coordinates respectively of a "peak" point in polar space.

4.8   Parametric Shape Template Generation

To generate r values from the shape templates for each θ value in $C_j^{'}$, a number of sub routines were defined for each type of shape template. Each one is passed the reference to the vectors `rads360` and `peaks`, and returns a double representing the fitness of the shape template

54

to the actual point data. The Quadrilateral Shape Template is shown here. All other shape templates follow a similar method but have slight differences that are intuitive.

First, the variables to be used are declared and initialized.

```
int th1, th2, th3, th4;
double pRad1,pRad2,pRad3,pRad4;
double reportFit = 0;
bool firstTime = true;
```

Then, two new vectors are filled representing all of the "peak" r and θ values.

```
for (unsigned int i=0; i< peaks.size(); i++)
{
        if (peaks[i] != 0)
        {
                peakRads.push_back(rads360[i]);
                theta.push_back(i);
        }
}
```

Now, the `pRad` and `th` variables declared above are assigned actual values. Since these values must be assigned in consecutive order in terms of θ, an exhaustive search of the possible combinations of these values can be done with merely a nested for loop as shown below

```
for (int m=0; m<peakRads.size()-3; m++)
{
        th1 = theta[m];
        pRad1 = peakRads[m];
        for (int j=m+1; j<peakRads.size()-2; j++)
        {
                th2 = theta[j];
                pRad2 = peakRads[j];
                for (int k=j+1; k<peakRads.size()-1; k++)
                {
                        th3 = theta[k];
                        pRad3 = peakRads[k];
                        for (int l=k+1; l<peakRads.size(); l++)
                        {
                                th4 = theta[l];
                                pRad4 = peakRads[l];
```

The fitness of each of these different combinations of peaks within the quadrilateral template are calculated by subtracting $G(\theta)$ (Equation 3-7) from the actual r value corresponding to θ from C_j'. This implementation is limited in scope to straight line shape templates, so a generic routine

55

(GenerateLineTemplate) was created which interpolates a point on a straight line when given the two endpoints and returns the distance of that point from the polar origin. To determine which of the piecewise functions from equation 3-7 to use, the current θ value must be compared to the four θ values of the peaks being used. Then, the appropriate limits can be given to GenerateLineTemplate to generate the correct $G(\theta)$.

```
for (int i=0; i< rads360.size(); i++)
{
        if (rads360[i] == 0) continue;
        if (i > th1 && i <= th2)
        {
        double tmp = GenerateLineTemplate
                        (i,th1*pi/180,th2*pi/180,pRad1,pRad2));
        tmp = fabs(rads360[i]- tmp);
        reportFit += tmp;
        }
```

This is then repeated for the 4 other possible locations of the current θ value within the shape template and the inputs to GenerateLineTemplate are changed accordingly.

```
        else if (i > th2 && i <= th3)…
        else if (i > th3 && i <= th4)…
        else if (i > th4 || i <= th1)…
        else printf("theta does not fall in a
                        real quadrant");
}
```

The set of peaks that produce the lowest reportFit are kept and the rest are ignored. This process is repeated for each defined shape template so that the templates can be compared to one another. The polar maps of the analytical shape templates super-imposed on $C_j^{'}$ is shown in Figure 4-10.

Notice that the analytical shape templates only produce comparison points for points that exist in $C_j^{'}$. This can be seen within the orange box of Figure 4-10. The spacing of the points generated by the shape templates perfectly match the spacing of the points in the sample data. If this were not the case, the reportFit variable would be biased and the comparison of shape templates in the next step could be rendered useless.
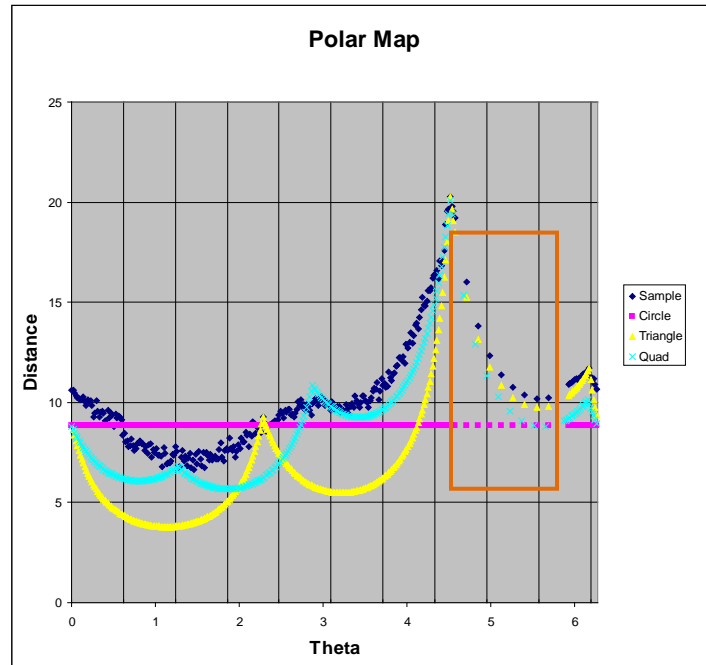
56

Figure 4-10: Analytical shape templates compared to $C_j^{'}$

4.9   Template Selection

The implementation of the Template Selection process is trivial. Each of the reportFit variables for each cross section must be summed. Then the template with the lowest summed reportFit is chosen unless the user specified through the GUI that a specific shape template should be used.

In our example problem the fitness values for each cross section and shape template are:

Table 4-1: 2D and 3D Approximation Errors for Defined Templates

		Circle	**Triangle**	**Quadrilateral**
ε_{2D}	CS 1	1200	4000	600
	CS 2	1450	3500	1250
ε_{3D}		**2650**	**7500**	**1850**

57

Comparing the approximation error of each template to the number of parameters required to represent each template illuminates the tradeoff between model complexity and model fitness.

Table 4-2: Cross Section Fitness vs Complexity

Template	ε_{3D}	# Parameters
Circle	2650	2
Triangle	7500	18
Quadrilateral	1850	24

A pair-wise comparison of each template fitness as well as a pair-wise comparison of the number of parameters required to represent the entire feature with a given template are presented in Table 4-3 and

Table 4-4 respectively.

Table 4-3: Pair-wise Comparison of Template Fitnesses

	Fitness Comparison		
	Circle	Triangle	Quadrilateral
Circle	0%	-183%	30%
Triangle	65%	0%	75%
Quadrilateral	-43%	-305%	0%

Table 4-4: Pair-wise Comparison of Template Complexity

	Complexity Comparison		
	Circle	Triangle	Quadrilateral
Circle	0%	-200%	-300%
Triangle	67%	0%	-33%
Quadrilateral	75%	25%	0%

58

4.10 **Geometry Creation**

A `CATIGSMPolyline` was used to create the cross sections (except for the circle which utilized a `CATIGSMCircleCenterAxis`). The `CATIGSMPolyline` represents an n-sided polygon with C^2 continuity. It is formed by adding a fillet to each corner of the polygon which maintains tangency to the two adjoining lines. It requires, as input, the location of each point as well as the radius of the fillet corresponding to each point. These cross sections are then used as input into the `CATIGSMLoft` object and the solid feature is either added or subtracted from the design space. A graphical representation of this whole process for the single feature treated in this implementation, including the created geometry, is shown in Figure 4-11.

CATIGSMPolylines   CATIGSMLoft
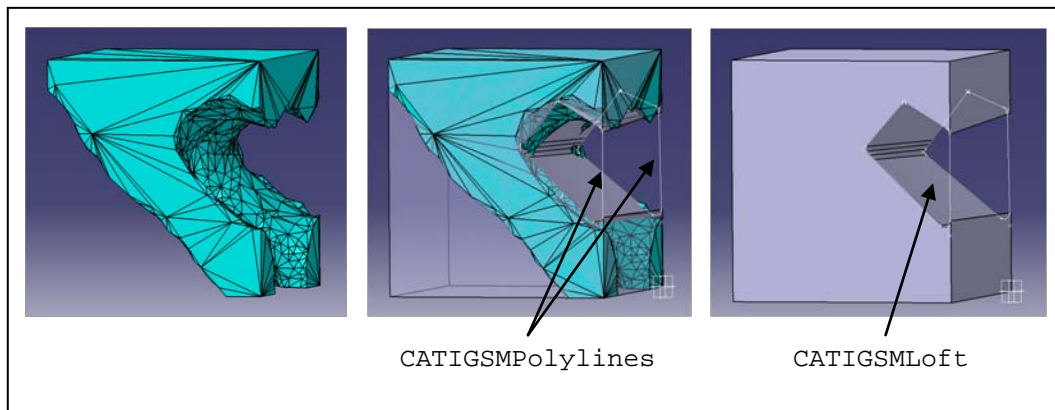
Figure 4-11: Graphical representation of the TO results refinement process

4.11 **Results**

The results of the implementation of the methods in four different TO case studies are presented in Table 4-5. The first column shows the original design space and loading conditions. The second column shows the tessellation object obtained from exporting an iso-surface of the TO results as discussed in section 2.1.2. The third column shows the final parametric model with

59

the outlines of the *features* that were created by the algorithm highlighted in orange. The fourth column shows the number of features used to approximate the optimal model. Finally, the fifth column shows the percent volume of the optimal model that the parametric model occupies. In each case, the objective of the TO was to minimize total mass subject to a total deflection constraint at a certain point. The point at which this deflection constraint was applied is represented in Table 4-5 with an X.

Table 4-5: Results of Implementation on 4 Models
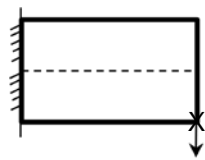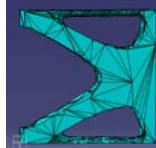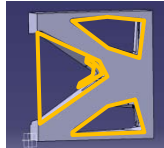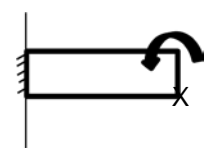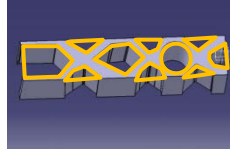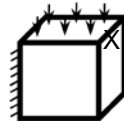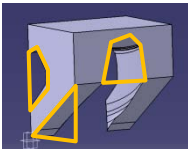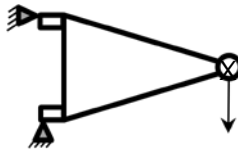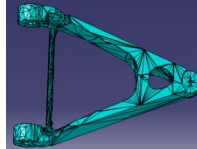
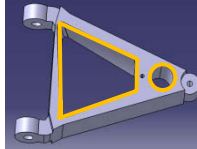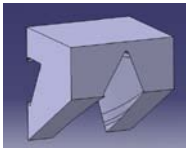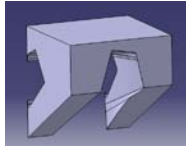| Loading Conditions	TO Results (Optimal Part)	Parametric Model	# Features	% of Optimal Volume
I			4	133%
II			10	158%
III			3	101%
IV			2	126%

Table 4-6 demonstrates the difference which results from using different shape templates on the same part. Notice that as the number of parameters increases, the volume approaches the optimal part volume.

Table 4-6: Tradeoff Between Complexity and Approximation Error

Template	Result	Number of Parameters (using 2 cross sections)	% of Optimal Volume
Triangle		30	130%
Quadrilateral		36	114%
Pentagon		42	101%

The reader may recall that the results from this implementation are intended to be used as a starting point for parametric optimization in the next step of the design process (see section 1.3). This brings up the question of how to measure the performance of the shape recognition algorithm. Industry standard practice would suggest that the models obtained from the algorithm be meshed and loaded in the same manner that the original part was and their *performances* in terms of stress, deflection, and mass be compared to what the Topology Optimization results said it should be. The parametric models shown in Table 4-5, however, are not intended to be the

61

finalized designs. Instead, the intent was to establish an appropriate *geometric* fit to the optimal model such that the subsequent parametric optimization will yield the best possible *performance*. For this reason, the volume of the resultant model was compared to the volume of the optimal model to provide feedback as to the effectiveness of the methods employed.

5   CONCLUSIONS AND FUTURE WORK

The main objective of this research was to create an automated process to evaluate a given set of points and surfaces from TO results and determine the best fit parametric CAD feature to represent the set of surfaces and points. This was to be done using standard CAD features that allow for simple parameterization and parametric optimization in later steps of the design process. At the same time, this process was intended to allow the designer/engineer to maintain control of the tradeoff between goodness of fit and geometric complexity.

Chapter 3 presented a generalized algorithm for determining the best shape to approximate a given set of TO results surfaces. This best fit shape was chosen from among a set of predefined parametric Shape Templates of varying complexities. Chapter 4 presented a simple implementation of the methods from Chapter 3 within the CATIA V5 R18 environment. This implementation was used to demonstrate the effectiveness of the methods on four different case studies. The case studies showed that a close geometric approximation to the optimal part can be obtained through using the method (Table 4-5). The implementation also showed the magnitude of the tradeoff between model complexity and model fitness and that this tradeoff can be effectively managed through using the methods of Chapter 3 (Table 4-6).

This thesis demonstrates that a shape recognition algorithm can be constructed to automatically recognize the topological entity that most closely approximates surfaces from TO results. It also shows that this can be done using simple geometric shapes that form standard features in most commercial CAD packages. This allows the designer/engineer to create

63

topologically optimal parts from within the CAD-centric environment. The simplicity of features and design also allow the model to be represented in a standard CAD format that other designers who come in contact with the model later will be able to work with in an intuitive, standard way.

5.1   **Future Work**

Although the results in chapter 4 demonstrate the effectiveness of the methods presented in this thesis, there are many opportunities for improvement. There are possible advancements that will increase the scope and/or eliminate some of the limitations of the methods. This section will present a few promising possibilities for future research.

5.1.1   Shape Distributions

One alternative method to Shape Templates that was considered in the initial phases of this research has to do with Shape Distributions [13] (see section 2.2.6). Shape Distributions could be used in the 3D realm in the same way that Shape Templates were used in this thesis in a 2D realm. Figure 5-1 shows shape distributions for open ended 3D rectangular extrusions of varying dimensions. These distributions were created by taking 100,000 random measurements between two points on the surface of the extrusion and creating a histogram of those measurements.

From these shape distributions it is possible to pick out the defining dimensional values of the original rectangular extrusion. For example, looking more closely at Figure 5-1 (d), it is easy to see that peaks occur in the shape distribution at the dimensions 25, and 50.
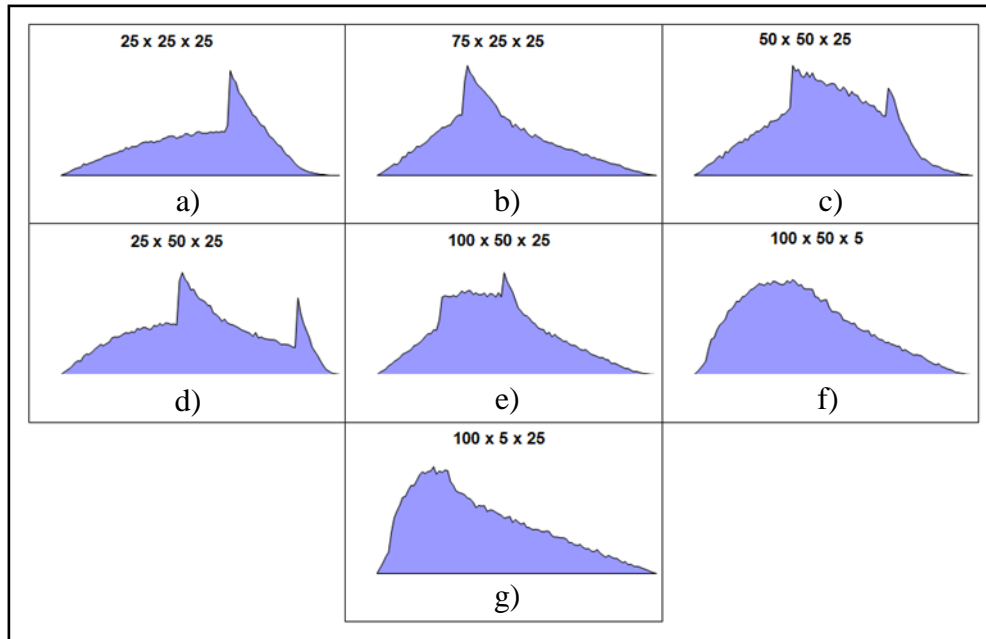
64

Figure 5-1: Shape distributions of for variations of an open ended rectangle
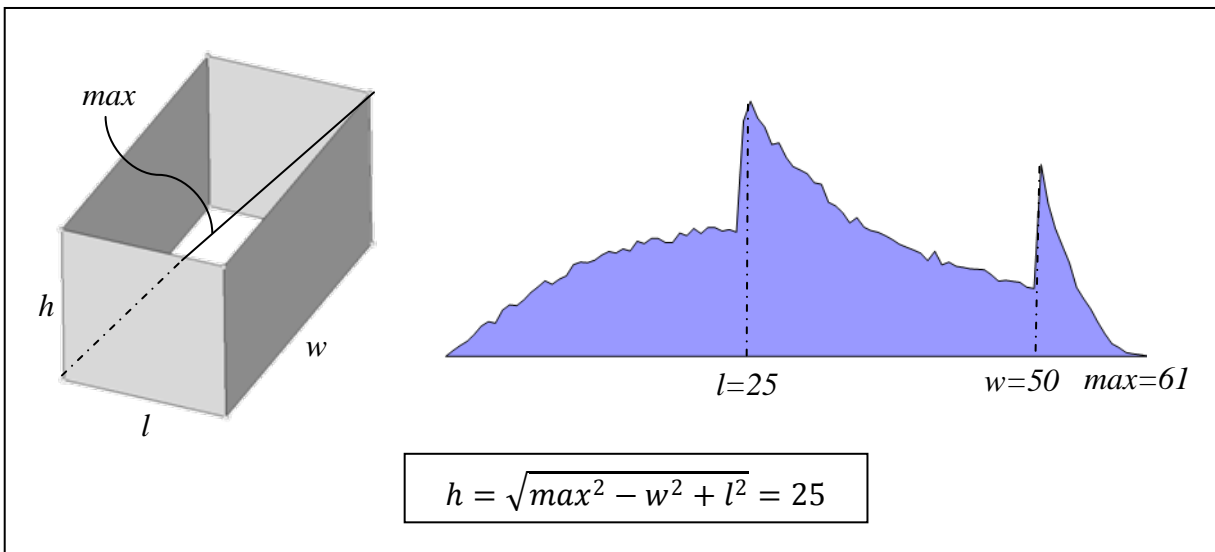

$$h = \sqrt{max^2 - w^2 + l^2} = 25$$

Figure 5-2: Parameters taken from a shape distribution

If a method were discovered to create shape distributions on the fly for different types of

3D shapes according to an analytical model instead of by taking 100,000 random measurements,

then shape distributions could be used in 3D space in the same way that Shape Templates were

used in 2D space. This would allow geometry to be created without sweeping through multiple cross sections, which would reduce some of the approximation error of the methods presented in this thesis.

5.1.2 Non-convex shape templates

One of the major limitations of the methods presented in this thesis is the requirement that all shape templates be convex shapes. One possible solution to this problem is to create cross sections using multiple reference points. Figure 5-3 shows the difference between the current method (left) and the theoretical proposed method (right). In the image on the right, if the relationship between points A and B are known, then the whole non-convex shape could be represented with multiple convex shape templates. If this is possible, the flexibility of the methods would be greatly increased.
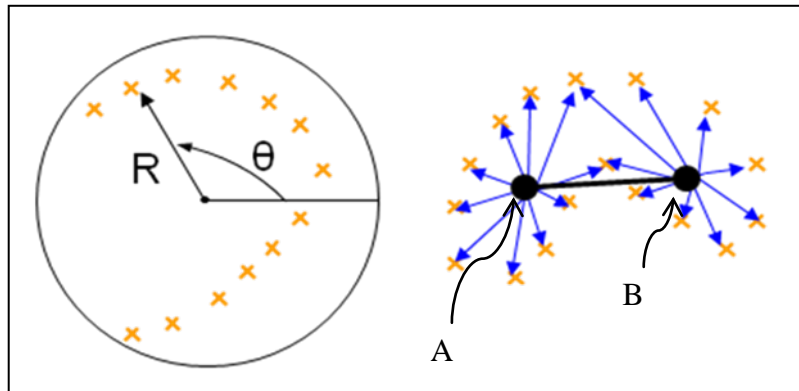

Figure 5-3: Convex template (left) and proposed non-convex template (right)

5.1.3 Other

One of the major limitations of the shape recognition algorithm is the ability to segment the input surfaces in a useful way. Currently, feature segmentation is done by a human with selection tools that are built into the CAD package. It is very difficult to select the desired surfaces with these rudimentary tools. Feature segmentation is also a source of variation. When one person looks at a model, they may segment the model into different features in a different way from the next person. This is not necessarily bad, but there should be some way to know if the feature surface segmentation step could be improved to yield improved overall results

More research is also needed in the area of spine rules. Spines control the shape of the feature between cross sections (see section 3.2.1). The *Feature Orientation Geometry,* which is a user input, is currently fulfilling this role. In the implementation of chapter 4, this *Feature Orientation Geometry* was simplified significantly in order to prove out the concept. An automated process to infer the *Feature Orientation Geometry* would add a great deal of value to the algorithm as a whole.

REFERENCES

[1]   Baumgartner, A.; Harzheim, L.; Mattheck, C.; "SKO (Sot Kill Option): the Biological way to find an optimum structure topology", *International Journal of Fatigue*, vol. 14, 387-393, 1992.

[2]   Bendsoe, M.; Kikuchi, N.; "Generating Optimal Topologies in Structural Design Using a Homogenization Method", *Computer Methods in Applied Mechanics and Engineering*, 71:197-224, 1988.

[3]   Bernardini, F.; Bajaj C. L.; Cheny J.; Schikore D. R.: "Automatic Reconstruction of 3D CAD Models", Department of Computer Sciences, Purdue University, West Lafayette, IN, 1999 http://ww.cs.purdue.edu/research/shastra.

[4]   Blattman W. R.: "Generating CAD Parametric Features Based on Topology Optimization Results", MS Thesis, Brigham Young University, Provo, UT, 2008 .

[5]   Campelo F.; Ota S.; Watanabe K.; Igarashi H.: "Generating Parametric Design Models Using Information From Topology Optimization", *IEEE Transactions on Magnetics*, 44-6:986-989, 2008.

[6]   Eschenauer, H. A.; Olhoff N.: "Topology optimization of continuum structures:A review", *Appl Mech* Rev vol 54, no 4, July 2001.

[7]   Hoppe, H.; DeRose, T.; Duchamp, T.; McDonald, J.; Stuetzle, W.; "Surface Reconstruction from Unorganized Points", *Computer Graphics*, 26, 2, 1992

[8]   Hsu, M.-H.; Hsu, Y.-L.: "Interpreting three-dimensional structural topology optimization results", *Computers and Structures*, 83:327-337, 2005.

[9]   Jain A. K.; Zhong Y.; Lakshmanan S.: "Object Matching Using Deformable Templates", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 18-3:267-278, 1996.

[10] King, M. L.; Fisher, M.J.; Jensen, C.G.; "A CAD-centric Approach to CFD Analysis With Discrete Features", *Computer-Aided Design & Applications*, Vol. 3, Nos. 1-4, 279-288, 2006.

[11] Lin, C.-Y.; Chao L.-S.; "Automated image interpretation for integrated topology and shape optimization", *Struct Multidisc Optim*, 20:125-137, 2000.

69

[12] Lin, C.-Y.; Lin S.-H.; "Artificial neural network based hole image interpretation techniques for integrated topology and shape optimization", *Computer Methods in Applied Mechanics and Engineering*, Vol. 194, Nos. 36-38, 3817-3837, 2005

[13] Osada, R.; Funkhouser, T.; Chazelle B.; Dobkin, D.: "Shape Distributions", *ACM Transactions on Graphics*, 21-4 807-832, 2002.

[14] Rozvany, G.I.N.; "Aims, Scope, Methods, History and Unified Terminology of Computer-Aided Topology Optimization in Structural Mechanics", *Struct Multidisc Optim*, 21, 90-108, 2001

[15] Tang, P.-S.; Chang, K.-H.: "Integration of topology and shape optimization for design of structural components", *Struct Multidisc Optim*, 22:65–82, 2001.

[16] Yin L.; Ananthasuresh G. K.; "A novel topology design scheme for the multi-physics problems of electro-thermally actuated compliant micromechanisms", *Finite Elements in Analysis and Design* 40, 1317-1331, 2004