Surface Reconstruction from Medical Imaging for Use in a Computer-Aided Design (CAD) Environment

Gregory M. Sturgeon

Grand Valley State University

Follow this and additional works at: http://scholarworks.gvsu.edu/theses
Part of the Engineering Commons

Recommended Citation
http://scholarworks.gvsu.edu/theses/654

This Thesis is brought to you for free and open access by the Graduate Research and Creative Practice at ScholarWorks@GVSU. It has been accepted for inclusion in Masters Theses by an authorized administrator of ScholarWorks@GVSU. For more information, please contact scholarworks@gvsu.edu.
Surface Reconstruction from Medical Imaging for Use in a Computer-Aided-Design (CAD) Environment

A thesis by
Gregory M. Sturgeon M.S.E.
Presented January 2005

In partial fulfillment of the requirements for the Masters of Science in Engineering degree

Padnos School of Engineering
Grand Valley State University

Advisor: Dr. Jeff Ray
The EGR 693 thesis by
Gregory M. Sturgeon
entitled
Surface Reconstruction from Medical Imaging for Use in a
Computer-Aided-Design (CAD) Environment
is accepted in partial fulfillment of the
requirements for the degree of
Masters of Science in Engineering

4 April 2005
Date

Dr. H. Jack, Graduate Chair
Abstract

A method was developed to create Computer-Aided-Design (CAD) models for bones of the human body utilizing medical imaging data. The human hand was chosen as the subject of the research.

Computed Tomography (CT) imaging was chosen to provide a volumetric data set. This data set was visualized through an isosurfacing technique utilizing the marching cubes algorithm. The original CT data set contained slices that were not aligned with the natural orientation or long axis of the bones. Transformation matrices and linear interpolations were used to generate a data set of slices oriented along the natural axis of the bones.

Contours were created on these slices through an edge-tracking method. B-Spline curves were then constructed utilizing the contour's vertices as knot points. A consistent starting location was found on each closed B-Spline curve relative to its centroid. Points on the closed B-Spline curves were then selected to define open non-uniform B-Spline curves in Pro/Engineer, a common CAD software package. Pro/Engineer was then utilized to trim the B-Spline curves to obtain their uniform portions. Cross-curves were developed through groupings of parallel B-Spline curves in order to define a closed boundary for a Boundary Blend surface patch. These surface patches were joined to adjacent surface patches and maintained C^2 curvature continuity.

The method presented applied common visualization techniques to a data set from CT imaging. This provided vertices from which to construct curves and surfaces in a CAD environment resulting in the ability to create detailed anatomical CAD models.
# Table of Contents

**LIST OF FIGURES** ......................................................................................................................... vii

**Chapter 1: Introduction and Background Information** ................................................................. 1
  1.1 Introduction .......................................................................................................................... 1
  1.2 Medical Imaging Technologies ............................................................................................ 1
    1.2.1 Computed Tomography .............................................................................................. 1
    1.2.2 Magnetic Resonance Imaging .................................................................................... 1
  1.2.3 Decision to Use CT Data .............................................................................................. 2
  1.3 CAD Background ................................................................................................................ 2
  1.4 B-Spline Curve Background .............................................................................................. 2
    1.4.1 B-Spline Curve Definition ......................................................................................... 3
    1.4.2 B-Spline Example ....................................................................................................... 4
    1.4.3 Local Control of Change ............................................................................................ 7
    1.4.4 Uniform B-Spline ......................................................................................................... 8
    1.4.5 Closed B-Spline ......................................................................................................... 10
  1.5 B-Spline Surfaces .............................................................................................................. 10
    1.5.1 Practical Consideration for Working With Surfaces ............................................. 12
  1.6 Overview of Visualization Techniques ............................................................................. 13
    1.6.1 Color Mapping ........................................................................................................... 13
    1.6.2 Contouring ................................................................................................................. 14
    1.6.3 Contouring Methods ................................................................................................. 14
      1.6.3.1 Edge Tracking .................................................................................................... 15
      1.6.3.2 Marching Squares" ............................................................................................ 15
    1.6.4 Extending Contouring to 3-D Isosurfacing ............................................................. 17
    1.6.5 Isosurfacing with the Marching Cubes" ................................................................ 17

**Chapter 2: Alternative Approaches** ......................................................................................... 20
  2.1 SURFdriver ....................................................................................................................... 20
    2.1.1 Using SURFdriver .................................................................................................... 21
    2.1.2 Difficulty of Sketching Contours ............................................................................. 21
    2.1.3 Resulting Surface from SURFdriver ...................................................................... 22
    2.1.4 Importing into Pro/Engineer ................................................................................... 22
    2.1.5 Tracing the Iges Curves ........................................................................................... 23
    2.1.6 Surfaces from Pro/Engineer ..................................................................................... 23
    2.1.7 Discussion of Results ............................................................................................... 24
  2.2 Isosurfacing in MATLAB" ............................................................................................... 24
    2.2.1 Importing the CT data ............................................................................................. 24
    2.2.2 Isosurface Function ................................................................................................. 26
    2.2.3 Reducing the Noise .................................................................................................. 27
    2.2.4 Discussion of Results ............................................................................................... 28
  2.3 Working with the Isosurfaces ............................................................................................ 28
    2.3.1 Method of Regrouping the Faces and Vertices ..................................................... 29
    2.3.2 Discovery of Joined Bones ....................................................................................... 29
    2.3.3 Slice Plane Through Joints .................................................................................... 29

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LIST OF FIGURES

Chapter 1: Introduction and Background Information

Figure 1.1: Basis functions for a fourth order B-Spline with K = 7. ....................... 5
Figure 1.2: Basis functions (a) for a sample fourth order B-Spline (b) with n = 6. ..... 6
Figure 1.3: The local control of change varies depending on how many segments a given control point influences. ................................................. 7
Figure 1.4: Uniform B-Spline basis functions with K = 4 shown over four intervals. ... 8
Figure 1.5: Comparison of uniform and non-uniform B-Spline curves. .................... 9
Figure 1.6: B-Spline surface with the control mesh and the numbering scheme of the control points. ................................................................. 11
Figure 1.7: The non-uniform B-Spline surface (a) interpolates the corner points of the control mesh where the uniform B-Spline surface (b) does not. ................. 12
Figure 1.8: Color mapped image of a typical slice of the CT data. ......................... 13
Figure 1.9: The color-mapped image (a) shows the region of interest of the CT data, 225x200 cells. The scalar values (b) are a 5x5 section as shown in the red on the color-mapped image. ................................................... 14
Figure 1.10: Contour lines created from the vertices seen in Figure 1.9b. ................. 15
Figure 1.11: Marching squares cases. ................................................................. 16
Figure 1.12: Solving the ambiguity by interpolating to find the above/below state of the center of the cell. ......................................................... 16
Figure 1.13: The 15 marching cubes cases. ......................................................... 17
Figure 1.14: Ambiguous cases of the marching cubes shown with their complementary cases. ........................................................ 18
Figure 1.15: The correct pairing of two adjoining marching cubes cases (a) will result in an isosurface with no holes. When the complementary case is incorrectly chosen (b) a hole will be introduced in the isosurface. ..................................... 18

Chapter 2: Alternative Approaches

Figure 2.1: Typical images of two successive slices from the CT data set. .............. 20
Figure 2.2: A sketched contour shown with an imaged slice from the CT data set at two different magnifications................................................. 21
Figure 2.3: Surface reconstruction of the M-II bone from SURFdriver. .................. 22
Figure 2.4: Boundary blend surfaces from Pro/Engineer shown with closed B-Spline curves sketched on parallel datum planes. ......................... 23
Figure 2.5: Illustration of the multidimensional array storing the slices of the CT data. 25

Figure 2.6: The region of interest, enclosed in the box, was selected from the entire dataset. .......................................................... 26

Figure 2.7: Isosurface image corresponding to the boundary between bone and soft tissue. ................................................................. 27

Figure 2.8: The isosurface from the original dataset (a) shown with the isosurface from the smoothed dataset (b). ........................................... 28

Figure 2.9: An enlarged view of the joint between two bones is shown in an edge display with its location in the shaded isosurface of the hand. ........................................... 29

Figure 2.10: A slice plane oriented through the thumb displayed as a bitmap (a). A color map was applied to the slice plane (b). ............................................ 30

Figure 2.11: Isosurfaces of four different values to investigate the separation of the bones at the joints. .......................................................... 31

Figure 2.12: Resulting separate entities, from separateBones, are shown in alternating shades. ............................................................... 32

Chapter 3: Contours on Slice Planes to B-Spline Surfaces

Figure 3.1: Two representative slices through the index finger in the original orientation (a) and in the desired “natural” orientation (b). .............................. 33

Figure 3.2: Uniformly spaced grid of points to resample the CT data. .................................................. 34

Figure 3.3: The central axis shown with the three orthogonal views of the index finger used to determine its location. .................................................. 35

Figure 3.4: Translation of point P₀ to the origin. ................................................. 36

Figure 3.5: Rotation of line P′ about the x-axis onto the x-y plane. .................. 37

Figure 3.6: Rotation of line P'’ about the z-axis in line with the x-axis. ............ 37

Figure 3.7: Points of the original dataset are illustrated as x’s and the resampled data points are illustrated as o’s. .................................................. 38

Figure 3.8: Sample contour and B-Spline fit to the contour vertices. ............ 41

Figure 3.9: Close-up of a self-intersecting loop and its location on the contour. ...... 42

Figure 3.10: Vertices of the resampled and original contours. .......................... 43

Figure 3.11: B-Spline obtained from the resampled contour, with equally spaced points, shown with the original contour. ................................. 44

Figure 3.12: Two separate non-uniform B-Spline curves, which share one end point. .. 46

Figure 3.13: Curvature plot for the two non-uniform B-Spline curves. The discontinuity is enlarged in the circle. ................................. 46
Figure 3.14: The curves of Figure 3.12 can be defined as uniform B-Spline curves through trimming longer curves. .............................................................. 47

Figure 3.15: $C^2$ curvature continuity maintained through the joint of two uniform B-Spline curves. The B-Spline curves were obtained through the trimming operations of Figure 3.14 .............................................................. 48

Figure 3.16: Non-uniform B-Spline curves (a) were trimmed to the uniform B-Spline curves (b). ......................................................................................... 49

Figure 3.17: Second direction cross curves fit through the end-points of the trimmed curves. ......................................................................................... 50

Figure 3.18: The boundary blend surface (a) was trimmed to obtain the uniform portion (b). ......................................................................................... 51

Figure 3.19: Two consecutive surfaces where the surfaces overlap prior to trimming the second surface (a) and the trimmed surfaces with $C^2$ continuity (b) ........................................ 51

Figure 3.20: Resulting surfaces with $C^2$ continuity. ........................................ 52

Figure 3.21: Image of a CT slice to be imported into Pro/Engineer. ...................... 53

Figure 3.22: The image of Figure 3.21 was imported into Pro/Engineer and is shown offset from the imported curves used to surface the finger. ......................... 53

Figure 3.23: B-Spline curves created through opposite groups of trim-points. ...... 54

Figure 3.24: End-cap curves intersecting at the centroid from the last curve projected onto the following slice. ................................................................. 55

Figure 3.25: Boundary blend surface from the end-cap curves with their intersection point and the parallel curves from the slices. ........................................ 56

Figure 3.26: The surfaces forming the end-cap of the bone shown with their boundary curves. ......................................................................................... 57

Chapter 4: Summary and Conclusions

Figure 4.1: Comparison of the CAD models created utilizing CT data and the surface of an actual bone. .............................................................................. 59

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Chapter 1: Introduction and Background Information

1.1 Introduction

The focus of this research was developing a method to create Computer-Aided-Design (CAD) models for bones of the human body utilizing medical imaging. The human hand was chosen as the subject of the research due to past interest in the area.

In this research medical imaging provided the data set from which curves and surfaces were developed. This allows the geometry of the surface models created in the CAD environment to accurately represent an individual's form and structure.

1.2 Medical Imaging Technologies

Medical imaging provides a data set from which curves and surfaces will be developed. This allows the geometry of the surface models in a CAD environment to accurately represent an individual's form and structure.

1.2.1 Computed Tomography

Computed tomography (CT) utilizes the transmission of X-rays through a material from many viewing angles. A single X-ray tube is rotated through a full rotation while recording projections at every 0.5° to 1°. This X-ray beam originates as a point source and after passing through the body is collimated to a single slice on an arc of detectors. These detectors measure the variation in transmission of the X-rays based on attenuation due to tissue opacity, which correlates to density. The projections of the X-rays are processed together to form an image of the densities on the slice. Each slice is a 2-D image with a defined thickness. The pixels, or cells, of the image can be thought of as voxels or a volumetric cube from which the density was measured. Sequential CT sections are imaged separately, by changing the position of the X-ray source and strip of detectors along a path perpendicular to the image, to construct a 3-D volume image.

1.2.2 Magnetic Resonance Imaging

Magnetic Resonance Imaging (MRI) technology measures electromagnetic energy from resonating hydrogen protons. A magnetic field is generated to align the hydrogen protons in the subject's body along the bore of the scanner.

Gradient magnets are used to generate radio frequency (RF) pulses, which excite the unbalanced protons. These protons absorb that energy and spin or precess in a different direction. Once the RF excitation is stopped, these protons precess back to their alignment in the magnetic field. In doing so, they emit electromagnetic energy in the
form of an RF signal at the characteristic Larmour frequency. The proton densities and relaxation times are reconstructed in slices similar to the CT images\(^1,2\).

1.2.3 Decision to Use CT Data

The decision to use CT data over MRI was based upon the material of interest. MRI has advantages in distinguishing between types of soft tissue. Where CT imaging provides greater sensitivity between hard and soft tissues\(^1\), which aids in the identification of bone.

Medical imaging was made available through the National Library of Medicine's (NLM) Visible Human Project\(^2\). Both CT and MRI data are available upon approval from the NLM.

1.3 CAD Background

The ultimate goal of this research is to accurately model bones of a human hand for use in CAD. It is important to understand how information is stored in a CAD environment. The industry standard for CAD software is parametric 3-D solid modeling software.

In a CAD environment, parametric equations are favored over single-valued functions, such as \( y = f(x) \). A two dimensional parametric curve is expressed as two functions of a parameter \( u \): \( x = x(u), y = y(u) \). This parametric form simplifies the modification of the curve. For example a line can be defined as \( y = mx + b \), where \( m \) is the slope of the line and \( b \) is the \( y \) intercept. The parametric equation of a line, over the interval \( 0 < u < 1 \), is \( P(u) = (1-u)P_0 + u P_1 \) where \( P_0 \) and \( P_1 \) are the start and end points\(^4\). The characteristics of the line can easily be changed in the parametric form by selecting new start or end points. Modifying the line of the explicit function would not be as easy. The advantages of parametric equations result in the ease of modifying designs.

Curves and surfaces allow for the modeling of geometric entities with complex curvature in multiple coordinate directions. B-Spline curves and surfaces have become the industry standard for this complex modeling. They enable greater control over curvature and are often used to produce “stylized” and ergonomic consumer products.

1.4 B-Spline Curve Background

B-Spline curves typically consist of multiple curve segments. Each segment is defined by a few control points, which influence the shape of the curve. B-Spline curves have several advantages over other curve types including Hermite and Bezier curves. In the Hermite and Bezier curves a change in shape of the curve affects the entire curve’s characteristics. This is known as a global propagation of local change\(^4,5\). However, in the definition of a B-Spline curve there is an on/off condition that allows a set number of
control points to influence each curve segment. This leads to a curve in which a local change affects a limited portion of the curve.

Additionally, the degree of the curve is independent of the number of control points. This means that numerous control points can define a B-Spline curve while maintaining the order of the curve.

Typically, in modeling curves and surfaces it is desired to maintain curvature continuity or $C^2$ continuity. This ensures the transition from one section to another remains smooth at the $d^2/du^2$ order. Specifically, the tangency of the curve is continuous through this transition. A B-Spline must be at least fourth order ($K = 4$) to achieve curvature continuity across curve segments.

1.4.1 B-Spline Curve Definition

A B-Spline curve is defined as:

$$p(u) = \sum_{i=0}^{n} p_i N_{i,K}(u)$$

(1.1)

There are $K$ control points that define each segment of a B-Spline curve. Where the degree of the curve is $(K-1)$. Basis functions define the influence each control point has on the curve’s shape over the parameter $u$. These basis functions are defined recursively as shown in Equation 1.2.

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

(1.2)

and

$$N_{i,k}(u) = \frac{(u - t_i) N_{i,k-1}(u)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - u) N_{i+1,k-1}(u)}{t_{i+k} - t_{i+1}}$$

for integer values of $k$, such that $2 \leq k \leq K$.

The variables $t_i$ are the knot values that make up a knot vector, which control the effective intervals of $u$ for the basis functions. For an open curve, the $t_i$ are calculated using $K$ as shown in Equations 1.3.
\( t_j = 0 \) \quad \text{if} \quad j < K
\( t_j = j - k + 1 \) \quad \text{if} \quad K \leq j \leq n
\( t_j = n - K + 2 \) \quad \text{if} \quad j > n

(1.3)

for integer values of \( k \), such that \( 0 \leq j \leq n+K \).

### 1.4.2 B-Spline Example

An example of calculating the basis functions for a fourth order curve with 7 control points (\( n = 6 \)) is shown in Appendix A. Each of the seven basis functions seen below contain a \( N_{ij}(u) \) term, which controls the effective range of the basis function.

\[
N_{0,4}(u) = (1 - u)^3 N_{3,1}(u)
\]
\[
N_{1,4}(u) = \frac{1}{4}(7u^3 - 18u^2 + 12u) N_{3,1}(u) + \frac{1}{4}(2 - u)^3 N_{4,1}(u)
\]
\[
N_{2,4}(u) = \frac{1}{12}(-11u^3 + 18u^2) N_{3,1}(u) + \frac{1}{12}(7u^3 - 36u^2 + 54u - 18) N_{4,1}(u) + \frac{1}{6}(3 - u)^3 N_{5,1}(u)
\]
\[
N_{3,4}(u) = \frac{1}{6}u^3 N_{3,1}(u) + \frac{1}{6}(-3u^3 + 12u^2 - 12u + 4) N_{4,1}(u) + \frac{1}{6}(3u^3 - 24u^2 + 60u - 44) N_{5,1}(u) + \frac{1}{6}(4 - u)^3 N_{6,1}(u)
\]
\[
N_{4,4}(u) = \frac{1}{6}(u - 1)^3 N_{4,1}(u) + \frac{1}{12}(-7u^3 + 48u^2 - 102u + 70) N_{5,1}(u) + \frac{1}{12}(11u^3 - 66u^2 + 114u - 384) N_{6,1}(u)
\]
\[
N_{5,4}(u) = \frac{1}{4}(u - 2)^3 N_{5,1}(u) + \frac{1}{4}(-7u^3 + 66u^2 - 204u + 208) N_{6,1}(u)
\]
\[
N_{6,4}(u) = (u - 3)^3 N_{6,1}(u)
\]

The \( N_{ij}(u) \), seen below, act as a switch turning on/off portions of the basis functions for a range of \( u \) values.

\[
N_{3,1}(u) = 1 \quad \text{for} \quad 0 \leq u < 1
\]
\[
= 0 \quad \text{otherwise}
\]
\[
N_{4,1}(u) = 1 \quad \text{for} \quad 1 \leq u < 2
\]
\[
= 0 \quad \text{otherwise}
\]
\[
N_{5,1}(u) = 1 \quad \text{for} \quad 2 \leq u < 3
\]
\[
= 0 \quad \text{otherwise}
\]
\[
N_{6,1}(u) = 1 \quad \text{for} \quad 3 \leq u < 4
\]
\[
= 0 \quad \text{otherwise}
\]

(1.5)

The \( N_{ij}(u) \) basis functions are shown in Figure 1.1
Notice the symmetry in the basis functions. The first and last K-1 basis functions (shown on the left and right) are non-uniform and there are \((n + 1) - 2(K - 1)\) uniform basis functions (shown center). If \(n\) had been larger there would have been more basis functions with the same shape as \(N_{3,4}\).

Recall basis functions control the influence of the control points on the B-Spline curve. The B-Spline curve is typically defined in segments over intervals of \(u\). The equations for these curve segments are found in Equations 1.6 below.
for $0 \leq u < 1$

\[ p_1(u) = (1 - u)^3 p_0 + \frac{1}{4}(7u^3 - 18u^2 + 12u)p_1 + \frac{1}{12}(-11u^3 + 18u^2)p_2 + \frac{1}{6}u^3 p_3 \]

for $1 \leq u < 2$

\[ p_2(u) = \frac{1}{4}(2 - u)^3 p_1 + \frac{1}{12}(7u^3 - 36u^2 + 54u - 18)p_2 + \frac{1}{6}(-3u^3 + 12u^2 - 12u + 4)p_3 + \frac{1}{6}(u - 1)^3 p_4 \] (1.6)

for $2 \leq u < 3$

\[ p_3(u) = \frac{1}{6}(3 - u)^3 p_2 + \frac{1}{6}(3u^3 - 24u^2 + 60u - 44)p_3 + \frac{1}{12}(-7u^3 + 48u^2 - 102u + 70)p_4 + \frac{1}{4}(u - 2)^3 p_5 \]

for $3 \leq u \leq 4$

\[ p_4(u) = \frac{1}{6}(4 - u)^3 p_3 + \frac{1}{12}(11u^3 - 114u^2 + 384u - 416)p_4 + \frac{1}{4}(-7u^3 + 66u^2 - 204u + 208)p_5 + (u - 3)^3 p_6 \]

Many relationships between the basis functions and the B-Spline curve can be observed from a plot of all the basis functions over the intervals of $u$ seen in Figure 1.2a. A sample B-Spline curve is shown in Figure 1.2b.

![Figure 1.2: Basis functions (a) for a sample fourth order B-Spline (b) with $n = 6$.](image)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Just as there are four control points that influence each segment of the B-Spline, there are 4 curves in each interval of \( u \). The \( N_{0,K} \) and the \( N_{a,K} \) basis functions reach the value of 1 at the beginning and end respectively. This means at that position the shape B-Spline curve is entirely dependent on a single control point. Simply put, the B-Spline curve interpolates its endpoints.

1.4.3 Local Control of Change

The local control of change was discussed earlier. A control point can influence at most \( K \) curve segments. The \( i^{th} \) control point only influences the B-Spline curve in the segments that contain the \( N_{i,K} \) basis function. Changing the position of the \( i^{th} \) control point only affects the segments in the range that the \( N_{i,K} \) basis function is active. The \( N_{1,4} \) basis function is active over the first two intervals as seen in Figure 1.2. If the position of control point \( p_1 \) were changed it would only influence the first two segments of the as seen in Figure 1.3a. However, the \( N_{3,4} \) basis function is active over 4 intervals. A change in \( p_3 \) would result in a change in 4 segments of the B-Spline as seen in Figure 1.3b. The joints between segments of the B-Spline curve are shown as a '+' . This point is often referred to as a knot point. The control polygon is formed by the control points defining the B-Spline.

Figure 1.3: The local control of change varies depending on how many segments a given control point influences.
1.4.4 Uniform B-Spline

Non-uniform B-Spline curves interpolate their end points where uniform B-Spline curves do not. A non-uniform B-Spline curve is defined by at least one non-uniform basis function. Often there are uniform portions, or segments, in a non-uniform B-Spline curve. This is often useful because in most CAD software it is not possible to define a curve as being either uniform or non-uniform. It is often necessary to define the curve to be longer than desired and trim it to obtain the desired portions.

A B-Spline curve can be reparameterized\(^4\) such that each curve segment is defined over the interval \(0 \leq u < 1\). For a cubic B-Spline curve with \(K = 4\), the uniform portion will be comprised of 4 basis functions that will repeat cyclically. The equations of these basis functions are shown in Equations 1.7 and are plotted over 4 intervals in Figure 1.4.

\[
\begin{align*}
N_{1,4}(u) &= \frac{1}{6}(-u^3 + 3u^2 - 3u + 1) \\
N_{2,4}(u) &= \frac{1}{6}(3u^3 - 6u^2 + 4) \\
N_{3,4}(u) &= \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1) \\
N_{4,4}(u) &= \frac{1}{6}u^3
\end{align*}
\]

Figure 1.4: Uniform B-Spline basis functions with \(K = 4\) shown over four intervals.

The equations for the uniform portions of B-Spline curves are often stored in a matrix form\(^4\). The \(i^{th}\) curve segment can be defined as seen in Equation 1.8 for a uniform B-Spline with \(K = 4\).

\[
p_i(u) = \frac{1}{6} \begin{pmatrix} u^3 & u^2 & u \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 0 & 3 & 0 \\
1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_{i-1} \\
p_i \\
p_{i+1} \\
p_{i+2} \end{pmatrix}
\]  

for \(1 \leq i \leq n-2\)  

(1.8)
The sample non-uniform B-Spline curve from Figure 1.2b can be compared to a uniform B-Spline curve with the same control points in Figure 1.5.

Recall, a uniform B-Spline curve or uniform segments of a B-Spline curve are those defined by only uniform basis functions. Notice from Figure 1.2a that there are non-uniform basis functions in the first three and last three intervals. This is true for all non-uniform cubic B-Spline curves that interpolate their end points. Since the non-uniform B-Spline curve in Figure 1.2b contains no uniform segments, it follows that none of the segments of the uniform and non-uniform B-Spline curves will match.

\[\text{Uniform B-Spline} \quad \text{Non-uniform B-Spline}\]

\textbf{Figure 1.5:} Comparison of uniform and non-uniform B-Spline curves.
1.4.5 Closed B-Spline

All of the previous discussion has involved open B-Spline curves. However, B-Splines are also particularly well suited for closed curves. Closed B-Spline curves consist of uniform B-Spline segments where the control points can be reused between the beginning and ending segments. This is often expressed in a matrix form and can be seen in Equation 1.9 for a cubic curve with $K = 4$.

$$
p_i(u) = \left( \begin{array}{c} u^3 \ u^2 \ u \ 1 \end{array} \right) \frac{1}{6} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & 6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_{(i-1) \mod (n+1)} \\ p_{(i) \mod (n+1)} \\ p_{(i+1) \mod (n+1)} \\ p_{(i+2) \mod (n+1)} \end{pmatrix} \text{ for } 1 \leq i \leq n+1 \tag{1.9}$$

Where the mod function is the remaindering operation. This function will return the remainder of a division. For example $5 \mod 4 = 1$.

A closed fourth order B-Spline with $n = 5$ (six control points) will have six segments defined as follows:

$$
p_1(u) = \frac{1}{6} \left( -u^3 + 3u^2 - 3u + 1 \right) p_0 + \frac{1}{6} \left( 3u^3 - 6u^2 + 4 \right) p_1 + \frac{1}{6} \left( -3u^3 + 3u^2 + 3u + 1 \right) p_2 + \frac{1}{6} u^3 p_3
$$

$$
p_2(u) = \frac{1}{6} \left( -u^3 + 3u^2 - 3u + 1 \right) p_1 + \frac{1}{6} \left( 3u^3 - 6u^2 + 4 \right) p_2 + \frac{1}{6} \left( -3u^3 + 3u^2 + 3u + 1 \right) p_3 + \frac{1}{6} u^3 p_4
$$

$$
p_3(u) = \frac{1}{6} \left( -u^3 + 3u^2 - 3u + 1 \right) p_2 + \frac{1}{6} \left( 3u^3 - 6u^2 + 4 \right) p_3 + \frac{1}{6} \left( -3u^3 + 3u^2 + 3u + 1 \right) p_4 + \frac{1}{6} u^3 p_5
$$

$$
p_4(u) = \frac{1}{6} \left( -u^3 + 3u^2 - 3u + 1 \right) p_3 + \frac{1}{6} \left( 3u^3 - 6u^2 + 4 \right) p_4 + \frac{1}{6} \left( -3u^3 + 3u^2 + 3u + 1 \right) p_5 + \frac{1}{6} u^3 p_0
$$

$$
p_5(u) = \frac{1}{6} \left( -u^3 + 3u^2 - 3u + 1 \right) p_4 + \frac{1}{6} \left( 3u^3 - 6u^2 + 4 \right) p_5 + \frac{1}{6} \left( -3u^3 + 3u^2 + 3u + 1 \right) p_0 + \frac{1}{6} u^3 p_1
$$

$$
p_6(u) = \frac{1}{6} \left( -u^3 + 3u^2 - 3u + 1 \right) p_5 + \frac{1}{6} \left( 3u^3 - 6u^2 + 4 \right) p_0 + \frac{1}{6} \left( -3u^3 + 3u^2 + 3u + 1 \right) p_1 + \frac{1}{6} u^3 p_2
$$

Note that each of these segments of the closed B-Spline curve could be used to define individual open uniform B-Spline curves.

1.5 B-Spline Surfaces

Where a single parameter $u$ was used to define every point on a curve, a second parameter $w$ is necessary to define a surface. The B-Spline curve can be extended to the B-Spline surface as defined in Equation 1.10.
\[ p(u, w) = \sum_{i=0}^{m} \sum_{j=0}^{n} p_{ij} N_{i,k}(u) N_{j,l}(w) \]  

(1.10)

Where the \( N_{i,k}(u) \) and \( N_{j,l}(w) \) are the same basis functions as in the B-Spline curve and defined in Equation 1.2. The control points, \( p_{ij} \), form an \((m + 1) \times (n + 1)\) array. The numbering of the control points can be seen in the B-Spline surface of Figure 1.6.

Figure 1.6: B-Spline surface with the control mesh and the numbering scheme of the control points.

Recall that a B-Spline curve is composed of multiple segments that are influenced by a limited number of control points. Similarly a B-Spline surface is composed of multiple patches. For a fourth order B-Spline surface each patch is defined by 16 control points, which will influence the geometry of that patch.

As there were uniform and non-uniform B-Spline curves there are also uniform and non-uniform B-Spline surfaces. The non-uniform B-Spline surfaces interpolate the corner points of the control mesh as seen in Figure 1.7a, where the uniform surfaces do not as seen in Figure 1.7b.
1.5.1 Practical Consideration for Working With Surfaces

When working with surfaces in a CAD environment, it is typically not possible or practical to build an entire model from a single surface. Instead, it is common practice to trim and merge surfaces to obtain the desired results. In order to merge or join two surfaces, the resulting surface must have $C^0$ point continuity. There must be one common continuous curve embedded in both surfaces in order to merge the two surfaces. This ensures that a hole is not introduced when joining two surfaces.

A surface must fully enclose a volume in order to define a solid model. A solid model contains volume. A surface model can be converted to a solid model if the surface model can "hold water". There must not be any holes on the seams for it to be watertight.

It is often desired to maintain $C^2$ curvature continuity across the patches of a surface. This provides a smooth surface where the "lines" joining the patches cannot be seen. In order to obtain $C^2$ continuity, the curves defining a surface must have $C^2$ continuity.
1.6 Overview of Visualization Techniques

Visualization allows an individual to see trends and interpret large amounts of data quickly and effectively. This section will provide an overview of common visualization techniques. The following techniques will be covered in this section: **color mapping** a technique to assign color values to scalar values arranged in a matrix, **contouring** a method that will separate regions of similar color and only display the boundary between these regions, and **isosurfacing** a 3-D extension to contouring where the boundaries created are surfaces rather than lines.

1.6.1 Color Mapping

In the previous approach images were visualized using a color mapping technique. Color mapping is a common visualization technique that maps a color to each cell or pixel. Recall that each cell of the CT dataset contains a scalar value corresponding to the density of a single voxel or volume element. These scalar values can be mapped to colors and displayed at the position of their cell. This is accomplished by utilizing the scalar value as an index into a color lookup table*. A color map of a typical slice of the CT data is shown in Figure 1.8. The higher density values corresponding to bone are shown in white and softer tissues are shown in grays.

![Figure 1.8: Color mapped image of a typical slice of the CT data.](image-url)
1.6.2 Contouring

When viewing these images, it is natural to separate areas of similar color into distinct regions. Similarly, contouring determines and displays only the boundaries between regions.

The vertices of the contour lines are created on the edges of a cell. Cells can be constructed by connecting a grid through the data points. The corners of the cells are scalar values of the data set as shown in Figure 1.9. Linear interpolation along the cell edges is then used to find a point equal to the contour value.

In order to separate the bone from soft tissue, the contour value was taken as the threshold between these two materials. The threshold between the densities of bone and soft tissue is cited in *The Visualization Toolkit* to be 120.5.

![Figure 1.9](image)

**Figure 1.9:** The color-mapped image (a) shows the region of interest of the CT data, 225x200 cells. The scalar values (b) are a 5x5 section as shown in the box on the color-mapped image. Vertices are found on the cell edges by linear interpolation finding the contour value 120.5.

1.6.3 Contouring Methods

Once the points are generated on the cell edges, it is then necessary to connect these points to form the contour. There are two methods to create the contour lines: edge tracking and a marching squares approach. The vertices from Figure 1.9b could be connected by either approach to form the contour in Figure 1.10.
1.6.3.1 Edge Tracking

The edge tracking approach tracks the edge of a contour as it enters and exits the cells. The contour is then tracked until it forms a closed contour or exits the boundary of the dataset. This is continued until all of the contours in that dataset have been tracked.

1.6.3.2 Marching Squares

The marching squares algorithm evaluates each cell individually. The premise of this approach is that contour lines can only pass through a cell in a limited number of ways. Each of the four vertices of the cell are evaluated to be either above (on) or below (off) the contour value. Then lines are constructed to separate the vertices that are on from those that are off. There are $16 (2^4)$ possible cases for the contour lines to pass through an individual cell and separate these vertices as shown in Figure 1.11. The contour lines essential separate regions that are above the contour value from those below.
Two of the cases shown in Figure 1.11 have two valid solutions for separating the vertices that are above the contour value from those below. The second solution or complement for each case is shown in dotted lines, when a second valid solution exists.

Interpolating to find the scalar value at the center of the cell can solve the ambiguity of these cases. Once the value at the center point is known there is only one possible solution for the resulting case this is illustrated in Figure 1.12.

Figure 1.11: Marching squares cases.

Figure 1.12: Solving the ambiguity by interpolating to find the above/below state of the center of the cell.
This marching squares algorithm generates individual primitives in each cell. When the primitives are displayed together they will form the contour. In this approach, each cell is evaluated individually. The vertices of the contour are determined in each cell and often a point merging operation is performed to eliminate these duplicate vertices on the cell boundaries.

1.6.4 Extending Contouring to 3-D Isosurfacing

The concepts of contouring can be extended to 3-D, where surfaces of constant value are displayed instead of contour lines. This 3-D extension to contouring is known as isosurfacing.

In 2D either the edge tracking approach or the marching squares algorithm were well suited to obtain the contour lines. However, the edge tracking method is difficult to extend to tracking a surface as it enters and exits a cube. The marching squares algorithm can best be extended to 3-D in an approach known as the marching cubes algorithm.

1.6.5 Isosurfacing with the Marching Cubes

In the marching cubes algorithm, a volumetric data set is evaluated in individual cubes. These cubes have eight vertices each with two possible, above or below, states. There are \(2^8\) combinations of separating the vertices by their states. By symmetry, these combinations reduce to the 15 marching cubes cases seen in Figure 1.13.

![Figure 1.13: The 15 marching cubes cases.](image-url)
Ambiguity also exists in some of the cases of the marching cubes. There are 6 cases with two vertices at a diagonal on at least one face, which leads to an ambiguous solution as seen for the marching squares in Figure 1.11. The marching cubes cases that exhibit complementary solutions are shown in Figure 1.14.

![Figure 1.14: Ambiguous cases of the marching cubes shown with their complementary cases.](image)

When pairing adjoining cubes, the result of improperly choosing the complementary case can lead to a hole in the surface. In Figure 1.15a two adjoining cubes are properly paired. However, in Figure 1.15b improperly choosing the complementary case will result in a hole in the surface. The decision to choose a complementary case cannot be made independently from the adjoining cubes.

![Figure 1.15: The correct pairing of two adjoining marching cubes cases (a) will result in an isosurface with no holes. When the complementary case is incorrectly chosen (b) a hole will be introduced in the isosurface.](image)
The resulting surfaces can be defined by the position of their vertices and the connectivity of the vertices. This information can then be stored in an array of the vertices' positions and an array of faces, which contains the indices to the three vertices that form the face. This will further be explained in Section 2.2.2.

The surfaces can be displayed together and will appear connected. However, it is important to note that the connectivity of the faces is not known. The faces are individual primitives stored in the order they are created. This issue is discussed in Section 2.3.
Chapter 2: Alternative Approaches

Various approaches were utilized that built upon the knowledge learned from previous attempts. The unsuccessful attempts, presented in this chapter, are included to document the discovery process and provide a framework for the proposed method of creating CAD models from medical imaging.

2.1 SURFdriver

The Visible Human Project\(^9\) listed SURFdriver\(^10\) as a software for 3-D reconstruction of CT or MRI data. The software can be obtained at http://www.surfdriver.com for a free trial period. SURFdriver allows the user to sketch linear segments on top of a background of the individual slices from the CT data.

This program enables the user to easily view slices of a CT data set in individual .fre files. The user can then visually determine the region of interest and create curves using the CT image as a reference. Curves can be edited by moving, inserting, or deleting points. This software also allowed the user to display curves from previous slices to visualize trends and identify changes in the geometry between successive slices.

![Typical images of two successive slices from the CT data set.](image)

**Figure 2.1:** Typical images of two successive slices from the CT data set.
2.1.1 Using SURFdriver

Multiple curves can be sketched on each slice with independent color designations. It was possible to cap the beginning and end curves of a specific entity to form a closed surface. Additionally, it was possible to specify that two curves joined to form one curve in the next slice, or conversely that one curve split to form two curves. This enabled the program to handle a Y shaped entity, although it was not used for bones of the hand.

2.1.2 Difficulty of Sketching Contours

The tracing of curves on the CT image was an arbitrary process. It was difficult to identify the threshold between material types. As seen in Figure 2.1, the denser regions corresponding to bone are shown in white, while less dense regions are shown in grays. When attempting to trace the boundaries of the various bones, the user had to determine where to draw the line between these regions.

It was also difficult to accurately capture the shape of the item being traced. When the image was enlarged, to aid in sketching, it became increasingly difficult to visualize the overall trends of the image. As seen in Figure 2.2b the distinction between the material types is not as sharp as in Figure 2.2a. The trade-off between size of the image (ease of point selection) and the clarity of the image necessitates switching between magnification levels repeatedly. Making the process of tracing the curves even more time consuming.

![Figure 2.2: A sketched contour shown with an imaged slice from the CT data set at two different magnifications.](image-url)
2.1.3 Resulting Surface from SURFdriver

SURFdriver uses a morphological geometric algorithm\(^1\) to construct a surface from the sketched contours. Coordinates for each contour are mapped to the subsequent contours to provide a wire-mesh model. A surface could be created from the wire-mesh model. An example of the metacarpal bone of the index finger (M-II) can be seen in Figure 2.3.

![Surface reconstruction of the M-II bone from SURFdriver.](image)

**Figure 2.3:** Surface reconstruction of the M-II bone from SURFdriver.

The surface from this technique was not smooth, which could be caused in part by the straight-line approximations between the vertices as seen in Figure 2.2.

2.1.4 Importing into Pro/Engineer

It was hypothesized that the surface quality could be improved through a more robust surface-modeling program, such as Pro/Engineer. Pro/Engineer would allow the user to have control over the definition of the surface patches where SURFdriver did not. Curves were exported from SURFdriver as Initial Graphics Exchange Specification (iges) files and imported into Pro/Engineer.
2.1.5 Tracing the Iges Curves

All of the curves were imported into Pro/Engineer as a single non-parametric feature. These curves could not be used to form surface patches directly. However, these curves would serve as a reference from which to construct B-Spline curves.

To approximate the iges curves closed B-Spline curves were drawn on datum planes. These datum planes were created every 1 mm, which was the spacing of each slice in the CT data. The curves from SURFdriver were composed of linear segments with \( C^0 \) point continuity. The closed B-Spline curves were smooth curves with \( C^2 \) curvature continuity.

2.1.6 Surfaces from Pro/Engineer

Additional curves were needed to form surface patches from the parallel curves. The start points and midpoints of these curves were used to generate secondary direction curves out of the plane. The parallel curves were used with these secondary direction curves, or cross curves, to form boundary blend surface patches. Two boundary blend surfaces can be seen in Figure 2.4 with closed B-Spline curves sketched on parallel datum planes.

![Figure 2.4: Boundary blend surfaces from Pro/Engineer shown with closed B-Spline curves sketched on parallel datum planes.](image)
Boundary blend surfaces are considered blended surfaces because the surface is blended between the curves that the surface is defined to pass through\(^7\). Boundary blend surfaces defined in two directions require the boundary curves to form a closed loop.

2.1.7 Discussion of Results

There are subtle changes between successive contours as seen in Figure 2.1. It was difficult to capture these changes without inducing other variations between contours. Vertices of the contours could only be placed at the corners of the pixels, which greatly reduced the freedom to trace these images. The subtle changes and the difficulty of consistently deciding where to place the vertices while tracing the images led to the irregularities and jagged nature of the surfaces developed from these sketches seen in Figure 2.3.

Although the surfaces shown in Figure 2.4 are considerably smoother than those in Figure 2.3, many irregularities still exist. The imprecision of manually tracing the individual images and the difficulties in tracing, due to the limited freedom in placing the vertices, ultimately led to these irregularities. The boundaries themselves were poorly defined, also leading to the inaccuracy of their tracing.

2.2 Isosurfacing in MATLAB\(^12\)

MATLAB, which stands for matrix laboratory, is an interactive software system integrating computation, graphics, and programming. MATLAB is well suited to working with matrices and arrays and is capable of handling large datasets. MATLAB also has a large library of well-documented functions for use in development in specialized applications, computation and visualization.

MATLAB was utilized to visualize the volumetric CT data set through the isosurfacing technique described in Section 1.5.4.

2.2.1 Importing the CT data

In order for the CT data to be imported into MATLAB, it first had to be converted into a supported format. The CT data slices were then converted from the .fre file format to a bitmap (.bmp) with the use of Graphics Converter Pro\(^13\) software.
Once the files of CT slices were in a supported format, MATLAB’s `imread` function was used to read from each file to an array. A simple `while` loop was used to read successively numbered files into a multidimensional array. The m-file containing this `while` loop can be found in Appendix B. The resulting array contained 200 slices with 512 rows by 512 columns as illustrated in Figure 2.5. The overall size of the array was 49.97 MB.

![Figure 2.5: Illustration of the multidimensional array storing the slices of the CT data.](image)

Slices of the volumetric data could be visualized using the `image` function. Several slices were imaged to identify a smaller region of interest. The boundary of this region is shown in Figure 2.6.

The original 512-by-512-by-200 array was reduced by over 80% to a 225-by-200-by-200 region of interest with a size of 8.58MB. Note that from this point, the reduced array was used as the dataset and may be referred to as such.
2.2.2 Isosurface Function

The volumetric data set could be visualized using the isosurfacing technique described in Section 1.5.4. Recall that the isosurfacing technique would construct surfaces of a constant scalar value. As in the previous example of contouring, a value of 120.5 was used as a threshold between bone and soft tissue.

The MATLAB *isosurface* function was used to produce an array of faces and vertices with the following code:

\[
[f \ v] = \text{isosurface}(CT, 120.5);
\]

The array of faces, \( f \), contains rows of indices pointing to the vertices that compose the face. Each face is composed of three vertices that are connected to form each of the triangular surfaces. The triangular surfaces are those created by the marching cubes approach for each cube as in the cases of Figure 1.13.

The array of vertices, \( v \), contains rows defining the position (coordinates values) of the vertices. The row numbers correspond to the indices in the array of faces.
The isosurface from the current data set resulted in 741,370 faces comprised of 393,729 vertices. All of the individual faces can be viewed together in Figure 2.7, which shows a shaded form.

![Figure 2.7: Isosurface image corresponding to the boundary between bone and soft tissue.](image)

2.2.3 Reducing the Noise

Notice there is considerable noise or debris shown in the isosurface image of Figure 2.7. The MATLAB function `smooth3` was used to smooth the dataset with the use of a convolution kernel of size [3 3 3]. An isosurfacing approach was then applied to the resulting smoothed dataset. Figure 2.8 shows the isosurface from the smoothed dataset and the original isosurface. The details of the bones are maintained in the isosurface from the smoothed data (b) without the noise from the original isosurface (a). The number of faces and vertices were reduced by over 50% to 355,332 faces and 179,162 vertices.
2.2.4 Discussion of Results

The isosurfacing approach provided an effective method for visualizing the bones of the hand from the CT dataset. Although it is important to note that this image is created from a multitude of individual triangular faces or surfaces. Due to the size of the data and the method of storing the faces and vertices, this is of little use in a CAD system.

2.3 Working with the Isosurfaces

After isosurfacing the CT data, efforts were made to rearrange the vertices and faces into groups of bones. The array of faces and vertices were constructed in a methodical marching approach as each cube was evaluated. It was desired to group the faces and vertices into smaller arrays defining individual bones.
2.3.1 Method of Regrouping the Faces and Vertices

The relationship between the vertices and faces was utilized to group the faces by their connectivity. The approach was to start with an arbitrary face and add the vertices of the face to a new array. Then search the original array of faces to find any unused faces that share any of the newly added vertices. These faces would be marked as used and added to the new array of faces and their vertices were added to the array of vertices. This was preformed on an iterative basis in the m-file *separateEntities* seen in Appendix C.

2.3.2 Discovery of Joined Bones

As a result of trying to sort the faces and vertices by entity, it was discovered that the isosurfacing approach merged separate bones into one surface. This was difficult to determine from viewing the isosurface as seen in Figure 2.9. This approach was abandoned after it was determined that the isosurface merged past the ends of the bones and joined separate bones.

![Figure 2.9: An enlarged view of the joint between two bones is shown in an edge display with its location in the shaded isosurface of the hand.](image)

2.3.3 Slice Plane Through Joints

To investigate why the isosurface joined the separate bones, a slice plane was created through the thumb as shown in Figure 2.10a. This slice plane shows a cross section through multiple joints of the thumb. Two-dimensional visualization techniques can be applied to this slice plane. A color map was applied to the slice plane, which can be seen in Figure 2.10b. The lower and upper bounds for the color map were 70 and 150 respectively. This allowed for the full color range to be localized about the threshold between bone and tissue.
Figure 2.10: A slice plane oriented through the thumb displayed as a bitmap (a). Note: the number of patches in the isosurface was reduced for clarity. A colormap was applied to the slice plane (b).

On inspection of Figure 2.10b, it is observed that the lower density contours corresponding to the isosurface value of 120.5 extended around what should be the end of one bone and encompass the next bone. The inner portions of the bones have a greater density and form separate contours. These higher density contours have separation between the bones at their joints.

2.3.4 Isosurfacing at Various Values

To investigate the separation between the bones at increased density values, the CT data was isosurfaced at various values to obtain 3-D images of this separation. Isosurfaces of the CT data at four different values can be seen in Figure 2.11.
Figure 2.11: Isosurfaces of four different values to investigate the separation of the bones at the joints.

Distinctly separate entities were created from isosurfacing the CT data at higher thresholds. The bones in Figure 2.11d are beginning to be separated into individual entities. However, the bones are no longer complete. Voids exist in the bones and fragments are separated from the bones.

2.3.5 An Attempt to Separate the Bones

The separation of individual bones, as in Figure 2.11d, was desired in order to model each bone independently. However, these surfaces do not represent the outer surfaces of the bones. They were obtained from a higher CT number than the transition from bone to soft tissue. These surfaces created from the higher CT number occur at some location inside of the bone. This causes the surfaces to be smaller and often incomplete.
An attempt was made to start from the separated entities, from a higher CT number, and 'grow' these entities to the outer surface of the bone. The m-file separateBones found in Appendix D was written for this purpose. Separate entities were found connected by a CT number of greater than 170. These entities were then each expanded by one layer of voxels on the outer boundaries. This expansion continued as long as the voxels to be added were greater than 120 and until all of the voxels were used. The results of this approach are shown in Figure 2.12.

Figure 2.12: Resulting separate entities, from separateBones, are shown in alternating shades.

2.3.6 Discussion of Results

This approach seemed to have some successes but ultimately failed to separate all of the individual bones and over separated others. The underlying difficulty was a combination of the resolution of the CT data and that each voxel could sample multiple materials. A voxel in the joint of two bones could be sampling: one bone, one bone and soft tissue, or two bones with or without soft tissue.
Chapter 3: Contours on Slice Planes to B-Spline Surfaces

3.1 Reorienting the Slices of the CT Data

After several attempts were made to utilize the isosurfaced data directly, it was then decided to work with individual slices. However, the slices of the CT data are often not arranged in an ideal or convenient orientation as seen in Figure 3.1a. The successive contours from this original orientation would tend to drift.

The CT data was resampled using transformation matrices and linear interpolation to yield a more desirable orientation. The resulting slices, as illustrated in Figure 3.1b, seem to be a more natural orientation to work with. This orientation of the slices would result in more intuitive surface patches, which would wrap around the long axis in a natural manner.

![Figure 3.1: Two representative slices through the index finger in the original orientation (a) and in the desired “natural” orientation (b).](image-url)
Different fingers and bones would have their own natural orientation that would not necessarily correspond to this orientation. For this reason, among others, it was decided that: while resampling the CT dataset to reorient the slices, the new dataset would also be reduced to include only the bones of the index finger.

3.1.1 Creating Points to Resample the CT Data

In order to provide points from which to resample the original data, a 3-D grid of uniformly spaced points was created such that it was centered about the long axis of the finger and fully enclosed the index finger. A simplified representation of this grid can be seen in Figure 3.2. These points would then be used to resample the CT data through linear interpolation to create a new dataset to work from.

Figure 3.2: Uniformly spaced grid of points to resample the CT data. Note: for clarity every fifth point is shown.
3.1.2 Finding the Long Axis

In order to create the grid of points described above, it was first necessary to construct a line approximating the central (long) axis of the bone. This line was constructed from the orthogonal views of the index finger and can be seen in Figure 3.3.

Figure 3.3: The central axis shown with the three orthogonal views of the index finger used to determine its location.

This line could then serve as a reference in determining the size of a box to enclose the index finger. For simplicity, the long axis was transformed to lie on the x-axis before the grid of uniformly spaced points was created. This grid could then easily be created about the x-axis and transformed back to the original orientation.
3.1.3 Reorienting the Long Axis

The central axis could then be transformed to the x-axis as described below.

The end points of the central axis were found to be at:

\[ \mathbf{P} = \begin{pmatrix} 40 & 90 \\ 38 & 31.5 \\ 167 & 137 \end{pmatrix} \]

Which is in the form:

\[ \mathbf{P} = \begin{pmatrix} P_0 \quad P_1 \\ P_0 \quad P_1 \\ P_0 \quad P_1 \end{pmatrix} \]

Translation of point \( P_0 \) to the origin:

\[ \mathbf{P'} = \mathbf{T} \cdot \mathbf{P} \]

\[ \mathbf{P'} = \begin{pmatrix} 1 & 0 & 0 & -P_{0x} \\ 0 & 1 & 0 & -P_{0y} \\ 0 & 0 & 1 & -P_{0z} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_{0x} & P_{1x} \\ P_{0y} & P_{1y} \\ P_{0z} & P_{1z} \end{pmatrix} \]

\[ \mathbf{P'} = \begin{pmatrix} 0 & 50 \\ 0 & -6.5 \\ 0 & -30 \end{pmatrix} \]

Shown in Figure 3.4.

**Figure 3.4:** Translation of point \( P_0 \) to the origin.
Rotation of line $P'$ about the x-axis onto the x-y plane:

$$P'' = R_x P'$$

$$P'' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{pmatrix} \begin{pmatrix} P'_{0x} \\ P'_{0y} \\ P'_{0z} \end{pmatrix}$$

$$\theta_x = \text{atan} \left( \frac{P'_{0y} - P'_{0y}}{P'_{0z} - P'_{0z}} \right)$$

$$P'' = \begin{pmatrix} 0 & 50 \\ 0 & 0 \\ 0 & -30.696 \end{pmatrix}$$

Shown in Figure 3.5.

Rotation of line $P''$ about the z-axis onto the x-axis:

$$P''' = R_x P''$$

$$P''' = \begin{pmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{pmatrix} P''$$

$$\theta_z = \text{atan} \left( \frac{P''_{0y} - P''_{0y}}{P''_{0z} - P''_{0z}} \right)$$

$$P''' = \begin{pmatrix} 0 & 58.671 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Shown in Figure 3.6.
3.1.4 Creating the Uniform Grid

The MATLAB function `meshgrid` was used to create an evenly spaced 3-D grid. This grid was centered about the x-axis and its size was determined from the three orthogonal views in Figure 3.3.

The points created from the `meshgrid` function were transformed back to the orientation of the index finger. These points are illustrated in Figure 3.7 as o's where the original points are seen as x's. This transformation was performed in the m-file `TransformGridResample` seen in Appendix E.

**Figure 3.7**: Points of the original dataset are illustrated as x’s and the resampled data points are illustrated as o’s. Note: for clarity every fifth point is shown.
3.1.5 Resampling the Uniform Grid

The points on the newly created uniform grid were then used as the points at which to resample the CT data. MATLAB's `intrep3` function performed the linear interpolation to find the CT number at the new points from the original dataset. Note that once the data was resampled this new dataset would be used to create contours, which would now follow a natural orientation.

3.2 Generating Contours and Fitting B-Spline Curves

Contours could be created at the transition from soft tissue to bone for each slice. These contours would follow the natural orientation of the finger due to the resampling of the data set in the previous section. B-Spline curves could be fit to the contours. Surfaces could then be created from these curves in a CAD environment such as Pro/Engineer.

3.2.1 Generating the Contours from MATLAB

Contours were obtained from a slice with the use of MATLAB's `contourc` function. This function utilized an edge-tracking method for contouring. The vertices were ordered such that successive vertices connected to form the contour. The command line:

```
C = contourc(CT(:,:,slice_num),[120.5 120.5]);
```

will generate the contours of the CT data for the slice ‘slice_num’ with a value of 120.5. The vertices of the contours will be stored in the array C. Before each contour is a header of the contour value and the number of vertices in the contour. This was used to separate the contours if multiple contours existed on that slice.

3.2.2 Fitting B-Spline Curves to the Contour Vertices

The contours from the `contourc` function form closed loops unless the contour exits the bounds of the CT data. Since all of the contours of interest form closed loops, the choice to use a single closed B-Spline curve over multiple open B-Spline curves seemed to follow.

Initially, a segment of the closed B-Spline curve was fit to each line segment of the contour. Each segment of the B-Spline was defined to start and end at one of the vertices of the contour. That is, the vertices of the contour were used as knot points in the B-Spline curves. Recall, knot points are the locations of the transitions between curve segments.
Each curve segment was defined by the parameter $u$ over the interval from 0 to 1. So, the transition from one curve segment to the next occurs at $u = 0$ and $u = 1$. The knot points, $P_t(0)$, can be related to the control points as seen in Equation 3.1 for a closed fourth order B-Spline curve.

\[
\frac{1}{6} \begin{bmatrix}
P_{t}(0) \\
P_{t+1}(0) \\
P_{t+2}(0) \\
\vdots \\
P_{n-1}(0) \\
P_{n}(0) \\
P_{n+1}(0)
\end{bmatrix} = \begin{bmatrix}
1 & 4 & 1 & \ldots & 0 & 0 & 0 \\
0 & 1 & 4 & \ldots & 1 & 0 & 0 \\
0 & 0 & 1 & \ldots & 4 & 1 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 1 & 4 & 1 \\
1 & 0 & 0 & \ldots & 0 & 1 & 4 \\
4 & 1 & 0 & \ldots & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
P_{t-1} \\
P_{t} \\
P_{t+1} \\
\vdots \\
P_{n-1} \\
P_{n} \\
P_{n+1}
\end{bmatrix}
\]

(3.1)

The control points can then be found to be:

\[
\begin{bmatrix}
P_{t-1} \\
P_{t} \\
P_{t+1} \\
\vdots \\
P_{n-1} \\
P_{n} \\
P_{n+1}
\end{bmatrix} = \begin{bmatrix}
1 & 4 & 1 & \ldots & 0 & 0 & 0 \\
0 & 1 & 4 & \ldots & 1 & 0 & 0 \\
0 & 0 & 1 & \ldots & 4 & 1 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 1 & 4 & 1 \\
1 & 0 & 0 & \ldots & 0 & 1 & 4 \\
4 & 1 & 0 & \ldots & 0 & 0 & 1
\end{bmatrix}^{-1} \begin{bmatrix}
P_{t}(0) \\
P_{t+1}(0) \\
P_{t+2}(0) \\
\vdots \\
P_{n-1}(0) \\
P_{n}(0) \\
P_{n+1}(0)
\end{bmatrix}
\]

(3.2)

For details on relating the knot points to the control points see Appendix F.

These control points can then be used to define the segments of the B-Spline curve. A sample B-Spline curve created from an arbitrary contour of the human hand can be seen in Figure 3.8. The function fitBspline, found in Appendix G, was written to create points on the B-Spline segments from the contours.
Figure 3.8: Sample contour and B-Spline fit to the contour vertices.

Upon inspection of this B-Spline curve a few segments self-intersect to form a loop as seen in Figure 3.9. This error seemed to occur where the contour points were spaced relatively close together.

3.2.3 Fitting B-Spline Curves to a Resampled Contour

It was hypothesized that B-Spline curves fit to contours with evenly spaced vertices would eliminate the self-intersecting loops, seen in Figure 3.9, since they occurred where the vertices were unevenly and closely spaced.
Figure 3.9: Close-up of a self-intersecting loop and its location on the contour.

It was possible to define line segments between the contour vertices and obtain equally spaced points on these line segments. The function \textit{resampleContour} in Appendix H was written to resample the contour lines and obtain equally spaced points on the contour.

In order to obtain the resampled contour, the length of the curve, from the start point to the point of interest, was evaluated for twenty points on each segment of the total curve. These lengths were then compared to equal divisions of the total perimeter of the curve. A desired number of points could be found that were equally spaced on the perimeter of the curve. See Appendix H for details.

The resampled contour can be seen with the original contour in Figure 3.10.
Figure 3.10: Vertices of the resampled and original contours.

The resampled contour in Figure 3.10 was constructed to contain the same number of contour vertices as the original contour.

The equally spaced vertices from the resampled contour were then used as the knot point of the B-Spline segments. The control points of the B-Spline segments were found as before and the resulting curve is shown in Figure 3.11.
The resulting B-Spline curve approximates the original contour and does not contain the self-intersecting loops exhibited previously.

3.2.4 Consistency in Starting Points

MATLAB's `contourc` function would not ensure that the starting points of the contours would be consistent between contours of different slices. This function would search through the rows and columns of the CT data to find the start of a new contour, which would then be traced until it formed a closed contour or exited the bounds of the data. The start point of a contour might not be located in the same position for a successive contour.

The inconsistent location of the start point could then lead to twisting in the surfaces developed from the contours. Before the B-Spline curves could be used to develop surfaces, a method to ensure the consistency of the start points was required.

The uppermost point above the centroid was used as the starting point for the B-Spline curve. This approach was also used in Solid Model Creation From CT Image Data\textsuperscript{15} where the centroid of the bitmapped image was found.
The functions \textit{centroid}_x and \textit{centroid}_y, found in Appendix I, were written to find the centroid of the closed B-Spline curve. Details of the derivation to find the centroid can be found in Appendix J.

The \( u \)-values and segment numbers of the points with the same \( x \)-value as the centroid can be found with the function \( P_{x\_cen\_x} \) found in Appendix K. The uppermost of these points can be found with the function \( top_{P_{x\_cen\_x}} \) found in Appendix L.

Once the starting point was found, the B-Spline was resampled to form a new B-Spline utilizing that starting point. Equally spaced points were constructed on the B-Spline curve, which were then used as the knot points for a new B-Spline curve. This was accomplished with the \textit{fitBspline} function seen in Appendix G.1.

3.2.5 Tracking the Contour Across Slices

Multiple contours, with the contour value of 120.5, often existed on a given slice. In addition to the contour of interest, the outer boundary of the bone, there were often contours due to noise and an inner contour within the bone.

The inner contour within the bone was not of interest and could be identified by determining if any vertices of a contour were contained within another contour. The MATLAB function \textit{inpolygon} \textsuperscript{14} was utilized in the function \textit{contoursSeperate} seen in Appendix M. This function was written to group the contours and fit the closed B-spline to the contours as described above. The results are an array of equally spaced knot points on the B-Spline curves.

The simplification of reducing the data set, to include only one finger, aided in the tracking of the contours in successive slices. Prior attempts were made to sort and group contours by their location with the goal of tracking a contour across successive slices. However, these approaches were only applicable when the number of contours on each slice was constant. The number of contours did not stay constant when dealing with the dataset from the whole hand. This was due to many factors including small contours induced by noise, inner contours within the bone were unpredictable, and contours would even merge into what should be a contour of a separate bone.

3.3 Utilizing Pro/Engineer to Create Surfaces from the B-Spline Curves

Pro/Engineer contains a robust surface-modeling module that was utilized to create surfaces from the parallel B-Spline curves developed in the previous section.
3.3.1 Importing to Pro/Engineer from an Ibl File Format

It is possible to import multiple curves into Pro/Engineer using an .ibl file. However, it is only possible to import open non-uniform curves with the use of these .ibl files. The points in the ibl file will be used as the knot points that join segments of the B-Spline curve. A sample ibl file can be found in Appendix N.

The closed B-Splines used to approximate the contours are comprised of uniform B-Spline curve segments. See Section 1.4.5 for details. The closed B-Spline can easily be separated into multiple open uniform B-Spline curves. However, using B-Splines with non-uniform segments, as would be created from the ibl file, would result in curves with $C^1$ tangent continuity at best.

3.3.1.1 Non-uniform B-Spline from an Ibl File

It was desired to connect a series of curve segments to yield a closed curve with $C^2$ continuity. The two curves shown in Figure 3.12 could be defined from an .ibl file. Pro/Engineer would construct two non-uniform B-Spline curves through the knot points specified in the .ibl file.

**Figure 3.12:** Two separate non-uniform B-Spline curves, which share one end point.

**Figure 3.13:** Curvature plot for the two non-uniform B-Spline curves. The discontinuity is enlarged in the circle.
The two separate non-uniform curves in Figure 3.12 could be joined with \( C^1 \) continuity. However, there is a discontinuity in the curvature at the joining point of the two curves as seen in Figure 3.13, which prevents the two curves from having \( C^2 \), curvature, continuity.

### 3.3.1.2 Trimming Curves to Maintain \( C^2 \) Continuity

If the two curves from the previous example were uniform B-Splines then it would be possible to obtain \( C^2 \) or curvature continuity. However, it is not possible in Pro/Engineer to simply define the curves to be uniform. Instead, it is possible to obtain the desired uniform portions from a longer non-uniform curve through trimming operations.

In Section 1.4.2 it was shown that the shape of a fourth order B-Spline curve was defined by non-uniform basis functions for the first three and last three segments of the curve. A trimming operation can be performed on a curve in Pro/Engineer that will maintain the geometry of the original curve. So, it is possible to trim off the non-uniform portions of a B-Spline and obtain the uniform portion of a curve.

A curve can be defined longer than desired, utilizing three points from the preceding curve and three points from the following curve. The three extra segments on each end can be trimmed to yield the desired uniform portion of the curve as shown in Figure 3.14.

![Figure 3.14: The curves of Figure 3.12 can be defined as uniform B-Spline curves through trimming longer curves.](image)
The trimmed curves from Figure 3.14 interpolate the same knot points as the curves in Figure 3.12. However, the trimmed curves are defined by uniform basis functions. The joint between the resulting trimmed curves maintains $C^2$ continuity as seen in Figure 3.15.

![Figure 3.15: $C^2$ curvature continuity maintained through the joint of two uniform B-Spline curves. The B-Spline curves were obtained through the trimming operations of Figure 3.14.](image)

### 3.3.2 Exporting the B-Splines from MATLAB

The closed B-Spline curves were divided into eight curve segments. Each curve segment would contain eight knot points. A knot point is shared at each of the joints between the current curve segments and the preceding and following curve segments. Due to the sharing of a knot point with the preceding and following curve, each of the curves would
contain seven unique knot points. So, there would only be 56 unique knot points for the group of curves. The function \textit{contoursSeperate} seen in Appendix M.1 was used to generate 56 points on the closed B-Spline, which would be used as the knot points in the Pro/E curves.

Recall, in order to obtain uniform B-Spline curves in Pro/E, the curves must be defined longer than desired and trimmed to obtain the desired uniform portion. Each of the resulting B-Spline curves would be defined by 14 knot points to obtain the desired uniform portion defined by eight knot points.

The knot points for each of the 8 groups of open B-Splines were exported to individual Microsoft Excel files with the m-file \textit{ExportToExcel} seen in Appendix P. The numbering scheme in Appendix Q shows the knot points that were used in each ibl file. In addition to the ibl files of B-Spline curves, points which would serve as the trim points were also exported. These points could then be imported as a pts file to be imported into Pro/E.

### 3.3.3 Creating Surfaces in Pro/E

The curves imported in the ibl files were trimmed using the points from the pts files. The non-uniform B-Spline curves in Figure 3.16a were trimmed to the uniform B-Spline curves in Figure 3.16b.

![Figure 3.16](image-url)

*Figure 3.16:* Non-uniform B-Spline curves (a) were trimmed to the uniform B-Spline curves (b).
Second direction cross curves were fit through the end-points of the trimmed curves as seen in Figure 3.17. Prior to trimming the first direction curves were defined by 14 knot points. For consistency, the cross curves were also defined by 14 knot points. After the trimming operations the uniform portions of these curves were each defined by eight knot points.

![Figure 3.17: Second direction cross curves fit through the end-points of the trimmed curves.](image)

A boundary blend surface was created from the 14 trimmed curves and the two cross curves and can be seen in Figure 3.18a. Note that boundary blend surfaces defined in two directions require the boundary curves to form a closed loop. This surface was trimmed to obtain the uniform portion of the surface shown in Figure 3.18b.
Figure 3.18: The boundary blend surface (a) was trimmed to obtain the uniform portion (b).

The next surface would utilize four of the same curves as the previous surface resulting in the overlap in the surfaces seen in Figure 3.19a. This surface would be trimmed to obtain the uniform portion and the resulting surfaces would have $C^2$ continuity at the joint as seen in Figure 3.19b.

Figure 3.19: Two consecutive surfaces where the surfaces overlap prior to trimming the second surface (a) and the trimmed surfaces with $C^2$ continuity (b).
3.3.4 Resulting Surfaces

The surfaces from this approach maintain $C^2$ curvature continuity through the joints between patches. These surfaces can be seen in Figure 3.20, which is in the same orientation as the previous section.

![Image](image.png)

**Figure 3.20:** Resulting surfaces with $C^2$ continuity.

3.3.5 Capping the Bones

These surfaces were created without end-caps to close the surface where the bone should end. These end-caps can be created separately by one of a few different methods.

3.3.5.1 Style Curves to Cap the Bones

An image can be inserted on to any of the default datum planes in Pro/E. This image can then serve as a reference in creating a style curve. This is often utilized to capture the style of a product from a free-hand sketch.

Style curves were to be created from images oriented through the long axis of the bone. The style curves would be sketched to determine where the bone should end. These style curves could then be reoriented to their location on the bone surface. They could then be utilized to generate surfaces for the end-caps of the bones.
However, the quality of the CT image was degraded when imported into Pro/E as seen by comparing Figures 3.21 and 3.22. The attempt to utilize this image to sketch style curves from was abandoned due to its poor quality.

Figure 3.21: Image of a CT slice to be imported into Pro/Engineer.

Figure 3.22: The image of Figure 3.21 was imported into Pro/Engineer and is shown offset from the imported curves used to surface the finger.
3.3.5.2 Capping the Finger

The surfaces to form the end-cap of the finger could not be defined by the same methods to construct the majority of the finger. B-Spline curves were created through a grouping of trim-points and the grouping of trim-points opposite them. The B-Spline curves that were created did not intersect each other as seen in Figure 3.23. Surfaces created from these curves would have no continuity across the boundaries. Therefore these surfaces could not be merged together.

Figure 3.23: B-Spline curves created through opposite groups of trim-points.
In order to provide a common point for all of the curves, the centroid of the last closed curve was projected to the following slice. This point was then used as an intermediate point when defining the curves from opposite groups of trim-points. These curves forming the end-caps were then forced to intersect at this point as seen in Figure 3.24.

Figure 3.24: End-cap curves intersecting at the centroid from the last curve projected onto the following slice.

Boundary blend surfaces were then created using these end-cap curves with the parallel curves from the slices and this intersection point as shown in Figure 3.25.
Figure 3.25: Boundary blend surface from the end-cap curves with their intersection point and the parallel curves from the slices.

This boundary blend surface differs from those forming the body of the finger. This surface forms a tri-patch, a surface patch with three sides, where a curve was blended to a single point. This surface is still a robust surface that can maintain $C^2$ continuity with its neighboring surfaces.

As before, these surfaces were defined to be longer than desired and were trimmed to obtain $C^2$ continuity with the adjoining surfaces. The surfaces forming the end-cap of the bone are shown in Figure 3.26.

The joint between bones are often a ball and cup pair. This process of capping the end of the bone was demonstrated for the convex ball-shaped end of a bone. The process of creating an end cap for a concave or cup-shaped end of a bone could be similar although multiple contours could exist on a slice. The concave surface of the bone could intersect the slice plane and appear as a second contour inside of the outer perimeter of the bone. However, as discussed in Section 2.3.2 the joints of the bones were not captured in the resolution of the CT scan.
Figure 3.26: The surfaces forming the end-cap of the bone shown with their boundary curves.
Chapter 4: Summary and Conclusions

Visualization techniques including isosurfacing and contouring were applied to a volumetric data set from CT imaging. These techniques were then extended to construct curves and surfaces in a CAD environment to create anatomical models.

Transformation matrices and linear interpolations were used to sample the volumetric data set to obtain slices aligned with the natural orientation of the bones. The boundaries between bones and soft tissues were identified by contours at a threshold related to their densities.

Closed B-Spline curves were fit to the contours as equally spaced points on the contours were used to define the knot points of the B-Spline curves. To ensure a consistent starting point and eliminate twisting between curves from consecutive slices, the uppermost point above the centroid of the closed B-Spline curve was used as the start point.

Due to the difficulty of importing closed B-Spline curves in Pro/Engineer, the closed curves were broken into open uniform curves. The original closed curves were sampled at equal lengths along the perimeter to provide knot points, to Pro/Engineer, defining open non-uniform B-Spline curves.

Pro/Engineer did not support uniform curves directly but non-uniform curves could be trimmed to yield their uniform portions. So, a curve could be defined that, once trimmed, the desired uniform curve would be obtained. This practice of trimming curves to obtain the uniform portions enabled the closed B-Spline curves to be modeled with open uniform B-Splines, while maintaining $C^2$ continuity.

Cross curves were developed through the endpoints of parallel B-Spline curves in order to define a closed boundary. These cross curves were utilized with the uniform curves used to define them to create boundary blend surface patches. These surface patches were joined to adjacent surface patches and maintained $C^2$ curvature continuity.

An area of difficulty that was explored but not fully overcome was the merging of separate bones into one entity. The resolution of the CT data and the sampling of a voxel containing multiple materials were factors limiting the separation of successive bones.

A method for capping the bone was developed for where the bone was known to end. This method utilized the centroid of the last slice project to the plane of the next slice. B-Spline curves were developed between two opposite trim-point groupings through this point. The tri-patch surfaces from these curves were robust surfaces also maintained $C^2$ continuity.

The process of modeling the surfaces of the bones involved many transformations and approximations to fit B-Spline curves and surfaces to the CT data set. In doing so, there was the potential for any errors to magnify over successive operations.
To obtain confidence in the methods presented, it was necessary to verify that the undulations present in the CAD model were representative of the surface of an actual bone. Bones of a human hand were obtained, on loan, from Grand Valley State University’s Anatomy Department. The index middle phalanx (MP-II) is shown for the CAD model created from CT data in Figure 4.1a and an actual bone in Figure 4.1b where both are shown at roughly the same orientation and magnification.

![Figure 4.1: Comparison of the CAD models created utilizing CT data (a) and the surface of an actual bone (b).](image)

The CAD model contains undulations in a similar magnitude as the surface of the bone. The surface of the bone contains indentations or pits not present in the CAD model. This is due to limitations in the resolution of the CT scan. The CAD model appears smoother than the surface of the bone. This alleviates the concern that the successive operation would magnify the irregularities of the bone’s surface. The CAD model also appears not to be over simplified, but it maintains the details of the bone.
Cited References:


Appendix A: Non-uniform B-Spline Example

The B-Spline curve is defined

\[ p(u) = \sum_{i=0}^{n} p_i N_{i,K}(u) \quad (A.1) \]

There are \( K \) control points that define each segment of a B-Spline curve. Where the degree of the curve is \( (K-1) \). Basis functions define the influence of each control point on the curve over the parameter \( u \). These basis functions are defined recursively as found in Equation A.2.

\[ N_{i,1}(u) = 1 \quad \text{if} \quad t_i < u \leq t_{i+1} \]
\[ N_{i,1}(u) = 0 \quad \text{otherwise} \]  \quad (A.2)

and

\[ N_{i,k}(u) = \frac{(u - t_i)N_{i,k-1}(u)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - u)N_{i+1,k-1}(u)}{t_{i+k} - t_{i+1}} \]

for integer values of \( k \), such that \( 2 \leq k \leq K \).

The variable \( t_j \) are the knot values that make up a knot vector. For an open curve the \( t_j \) are calculated using \( K \).

\[ t_j = 0 \quad \text{if} \quad j < K \]
\[ t_j = j - k + 1 \quad \text{if} \quad K \leq j \leq n \]  \quad (A.3)
\[ t_j = n - K + 2 \quad \text{if} \quad j > n \]

for integer values of \( k \), such that \( 0 \leq j \leq n + K \).

Example

For a cubic B-Spline with \( K = 4 \) and seven control points \( n = 6 \) the basis functions can be found as follows.

First the knot vector is computed using Equation A.3.
\[ t_0 = 0 \]
\[ t_1 = 0 \]
\[ t_2 = 0 \]
\[ t_3 = 0 \]
\[ t_4 = 1 \]
\[ t_5 = 2 \]
\[ t_6 = 3 \]
\[ t_7 = 4 \]
\[ t_8 = 4 \]
\[ t_9 = 4 \]
\[ t_{10} = 4 \]

Then the \( N_{i,1} \) basis functions, which act as switches can be determined from Equation A.2.

\[ N_{3,i}(u) = 1 \quad \text{for} \quad 0 < u < 1 \]
\[ = 0 \quad \text{otherwise} \]

\[ N_{4,i}(u) = 1 \quad \text{for} \quad 1 < u < 2 \]
\[ = 0 \quad \text{otherwise} \]

\[ N_{5,i}(u) = 1 \quad \text{for} \quad 2 < u < 3 \]
\[ = 0 \quad \text{otherwise} \]

\[ N_{6,i}(u) = 1 \quad \text{for} \quad 3 < u < 4 \]
\[ = 0 \quad \text{otherwise} \]

The \( N_{i,2} \) basis functions are then found from Equation A.2.

For \( N_{0,2}(u) \)

\[ N_{0,2}(u) = \frac{(u - t_0)N_{0,1}(u)}{t_1 - t_0} + \frac{(t_2 - u)N_{1,1}(u)}{t_2 - t_1} \]
\[ N_{0,2}(u) = \frac{(u - 0)N_{0,1}(u)}{0 - 0} + \frac{(0 - u)N_{1,1}(u)}{0 - 0} \]
\[ N_{0,2}(u) = 0 \]

For \( N_{1,2}(u) \)

\[ N_{1,2}(u) = \frac{(u - t_1)N_{1,1}(u)}{t_2 - t_1} + \frac{(t_3 - u)N_{2,1}(u)}{t_3 - t_2} \]
\[
N_{1,2}(u) = \frac{(u - 0)N_{1,1}(u)}{0 - 0} + \frac{(0 - u)N_{2,1}(u)}{0 - 0}
\]

\[
N_{1,2}(u) = 0
\]

For \(N_{2,2}(u)\)
\[
N_{2,2}(u) = \frac{(u - t_2)N_{2,1}(u)}{t_3 - t_2} + \frac{(t_4 - u)N_{3,1}(u)}{t_4 - t_3}
\]
\[
N_{2,2}(u) = \frac{(u - 0)N_{2,1}(u)}{0 - 0} + \frac{(1 - u)N_{3,1}(u)}{1 - 0}
\]

\[
N_{2,2}(u) = (1 - u)N_{3,1}(u)
\]

For \(N_{3,2}(u)\)
\[
N_{3,2}(u) = \frac{(u - t_3)N_{3,1}(u)}{t_4 - t_3} + \frac{(t_5 - u)N_{4,1}(u)}{t_5 - t_4}
\]
\[
N_{3,2}(u) = \frac{(u - 0)N_{3,1}(u)}{0 - 0} + \frac{(2 - u)N_{4,1}(u)}{2 - 1}
\]

\[
N_{3,2}(u) = (u)N_{3,1}(u) + (2 - u)N_{4,1}(u)
\]

For \(N_{4,2}(u)\)
\[
N_{4,2}(u) = \frac{(u - t_4)N_{4,1}(u)}{t_5 - t_4} + \frac{(t_6 - u)N_{5,1}(u)}{t_6 - t_5}
\]
\[
N_{4,2}(u) = \frac{(u - 1)N_{4,1}(u)}{2 - 1} + \frac{(3 - u)N_{5,1}(u)}{3 - 2}
\]

\[
N_{4,2}(u) = (u - 1)N_{4,1}(u) + (3 - u)N_{5,1}(u)
\]

For \(N_{5,2}(u)\)
\[
N_{5,2}(u) = \frac{(u - t_5)N_{5,1}(u)}{t_6 - t_5} + \frac{(t_7 - u)N_{6,1}(u)}{t_7 - t_6}
\]
\[
N_{5,2}(u) = \frac{(u - 2)N_{5,1}(u)}{3 - 2} + \frac{(4 - u)N_{6,1}(u)}{4 - 3}
\]

\[
N_{5,2}(u) = (u - 2)N_{5,1}(u) + (4 - u)N_{6,1}(u)
\]

For \(N_{6,2}(u)\)
\[
N_{6,2}(u) = \frac{(u - t_6)N_{6,1}(u)}{t_7 - t_6} + \frac{(t_8 - u)N_{7,1}(u)}{t_8 - t_7}
\]
Then the \( N_{i,3} \) basis functions are found and the \( N_{i,2} \) basis functions are substituted appropriately.

For \( N_{0,3}(u) \)
\[
N_{0,3}(u) = \frac{(u - t_0)N_{0,2}(u)}{t_2 - t_0} + \frac{(t_3 - u)N_{1,2}(u)}{t_3 - t_1}
\]
\[
N_{0,3}(u) = \frac{(u - 0)N_{0,2}(u)}{0 - 0} + \frac{(0 - u)N_{1,2}(u)}{0 - 0}
\]
\[N_{0,3}(u) = 0\]

For \( N_{1,3}(u) \)
\[
N_{1,3}(u) = \frac{(u - t_3)N_{1,2}(u)}{t_3 - t_1} + \frac{(t_4 - u)N_{2,2}(u)}{t_4 - t_2}
\]
\[
N_{1,3}(u) = \frac{(u - 0)N_{1,2}(u)}{0 - 0} + \frac{(1 - u)N_{2,2}(u)}{1 - 0}
\]
\[N_{1,3}(u) = (1 - u)N_{2,2}(u)\]
\[N_{1,3}(u) = (1 - u)[(1 - u)N_{1,2}(u)]\]
\[N_{1,3}(u) = (1 - u)^2N_{3,1}(u)\]

For \( N_{2,3}(u) \)
\[
N_{2,3}(u) = \frac{(u - t_2)N_{2,2}(u)}{t_4 - t_2} + \frac{(t_5 - u)N_{3,2}(u)}{t_5 - t_3}
\]
\[
N_{2,3}(u) = \frac{(u - 0)N_{2,2}(u)}{1 - 0} + \frac{(2 - u)N_{3,2}(u)}{2 - 0}
\]
\[N_{2,3}(u) = (u)N_{2,2}u + \frac{1}{2}(2 - u)N_{3,2}u\]
\[N_{2,3}(u) = (u)[(1 - u)N_{3,1}(u)] + \frac{1}{2}(2 - u)[(u)N_{3,1}(u) + (2 - u)N_{4,1}(u)]\]
\[N_{2,3}(u) = \frac{1}{2}(-3u^2 + 4u)N_{3,1}(u) + \frac{1}{2}(2 - u)^2N_{4,1}(u)\]
For $N_{3,3}(u)$

$$N_{3,3}(u) = \frac{(u - t_3)N_{3,2}(u)}{t_5 - t_3} + \frac{(t_6 - u)N_{4,2}(u)}{t_6 - t_4}$$

$$N_{3,3}(u) = \frac{(u - 0)N_{3,2}(u)}{2 - 0} + \frac{(3 - u)N_{4,2}(u)}{3 - 1}$$

$$N_{3,3}(u) = \frac{1}{2}(u)N_{3,2}(u) + \frac{1}{2}(3 - u)N_{4,2}(u)$$

$$N_{3,3}(u) = \frac{1}{2}(u)[(u)N_{3,1}(u) + (2 - u)N_{4,1}(u)] + \frac{1}{2}(3 - u)[(u - 1)N_{4,1}(u) + (3 - u)N_{5,1}(u)]$$

$$N_{3,3}(u) = \frac{1}{2}u^2 N_{3,1}(u) + \frac{1}{2}(-2u^2 + 6u - 3) N_{4,1}(u) + \frac{1}{2}(3 - u)^2 N_{5,1}(u)$$

For $N_{4,3}(u)$

$$N_{4,3}(u) = \frac{(u - t_4)N_{4,2}(u)}{t_6 - t_4} + \frac{(t_7 - u)N_{5,2}(u)}{t_7 - t_5}$$

$$N_{4,3}(u) = \frac{(u - 1)N_{4,2}(u)}{3 - 1} + \frac{(4 - u)N_{5,2}(u)}{4 - 2}$$

$$N_{4,3}(u) = \frac{1}{2}(u - 1)(u)N_{4,1}(u) + \frac{1}{2}(4 - u)N_{5,1}(u)$$

$$N_{4,3}(u) = \frac{1}{2}(u - 1)[(u - 1)N_{4,1}(u) + (3 - u)N_{5,1}(u)] + \frac{1}{2}(4 - u)[(u - 2)N_{5,1}(u) + (4 - u)N_{6,1}(u)]$$

$$N_{4,3}(u) = \frac{1}{2}(u - 1)^2 N_{4,1}(u) + \frac{1}{2}(-2u^2 + 10u - 11) N_{5,1}(u) + \frac{1}{2}(4 - u)^2 N_{6,1}(u)$$

For $N_{5,3}(u)$

$$N_{5,3}(u) = \frac{(u - t_5)N_{5,2}(u)}{t_7 - t_5} + \frac{(t_6 - u)N_{6,2}(u)}{t_6 - t_4}$$

$$N_{5,3}(u) = \frac{(u - 2)N_{5,2}(u)}{4 - 2} + \frac{(4 - u)N_{6,2}(u)}{4 - 3}$$

$$N_{5,3}(u) = \frac{1}{2}(u - 2)(u)N_{5,1}(u) + (4 - u)N_{6,1}(u)$$

$$N_{5,3}(u) = \frac{1}{2}(u - 2)[(u - 2)N_{5,1}(u) + (4 - u)N_{6,1}(u)] + (4 - u)[(u - 3)N_{6,1}(u)]$$

$$N_{5,3}(u) = \frac{1}{2}(u - 2)^2 N_{5,1}(u) + \frac{1}{2}(-3u^2 + 20u - 32) N_{6,1}(u)$$
For \( N_{6,3}(u) \)
\[
N_{6,3}(u) = \frac{(u - t_0)N_6(u)}{t_0 - t_0} + \frac{(t_0 - u)N_7(u)}{t_0 - t_7}
\]
\[
N_{6,3}(u) = \frac{(u - 3)N_6(u)}{4 - 3} + \frac{(4 - u)N_7(u)}{4 - 4}
\]
\[
N_{6,3}(u) = (u - 3)N_6(u)
\]
\[
N_{6,3}(u) = (u - 3)^2 N_6(u)
\]

Now the \( N_{6,4} \) basis functions can be found and the \( N_{6,3} \) basis functions are substituted appropriately.

For \( N_{0,4}(u) \)
\[
N_{0,4}(u) = \frac{(u - t_0)N_0(u)}{t_0 - t_0} + \frac{(t_0 - u)N_1(u)}{t_0 - t_1}
\]
\[
N_{0,4}(u) = \frac{(u - 0)N_0(u)}{0 - 0} + \frac{(1 - u)N_1(u)}{1 - 0}
\]
\[
N_{0,4}(u) = (1 - u)N_1(u)
\]
\[
N_{0,4}(u) = (1 - u)^3 N_3,1(u)
\]

For \( N_{1,4}(u) \)
\[
N_{1,4}(u) = \frac{(u - t_1)N_1(u)}{t_1 - t_1} + \frac{(t_1 - u)N_2(u)}{t_1 - t_2}
\]
\[
N_{1,4}(u) = \frac{(u - 1)N_1(u)}{1 - 0} + \frac{(2 - u)N_2(u)}{2 - 0}
\]
\[
N_{1,4}(u) = (u)N_1(u) + \frac{1}{2}(2 - u)N_2(u)
\]
\[
N_{1,4}(u) = (u)\left[ (1 - u)^2 N_3,1(u) \right] + \frac{1}{2}(2 - u)\left[ \frac{1}{2}(-3u^2 + 4u)N_3,1(u) + \frac{1}{2}(2 - u)^2 N_{1,4}(u) \right]
\]
\[
N_{1,4}(u) = \frac{1}{4}\left( 7u^3 - 18u^3 + 12u \right)N_3,1(u) + \frac{1}{4}(2 - u)^3 N_{4,1}(u)
\]
For $N_{2,4}(u)$

\[
N_{2,4}(u) = \frac{(u - t_2)N_{2,3}(u)}{t_5 - t_2} + \frac{(t_5 - u)N_{3,3}(u)}{t_5 - t_3}
\]

\[
N_{2,4}(u) = \frac{(u - t_2)N_{2,3}(u)}{2 - 0} + \frac{(3 - u)N_{3,3}(u)}{3 - 0}
\]

\[
N_{2,4}(u) = \frac{1}{2}uN_{2,3}(u) + \frac{1}{3}(3 - u)N_{3,3}(u)
\]

\[
N_{2,4}(u) = \frac{1}{2}u\left[\frac{1}{2}\left(-3u^2 + 4u\right)N_{3,1}(u) + \frac{1}{2}(2 - u)^2N_{4,1}(u)\right] + \frac{1}{3}(3 - u)\left[\frac{1}{2}u^2N_{3,1}(u) + \frac{1}{2}\left(-2u^2 + 6u - 3\right)N_{4,1}(u) + \frac{1}{2}(3 - u)^2N_{5,1}(u)\right]
\]

\[
N_{2,4}(u) = \frac{1}{12}\left(-11u^3 + 18u^2\right)N_{3,1}(u) + \frac{1}{12}\left(7u^3 - 36u^2 + 54u - 18\right)N_{4,1}(u) + \frac{1}{6}(3 - u)^3N_{5,1}(u)
\]

For $N_{3,4}(u)$

\[
N_{3,4}(u) = \frac{(u - t_2)N_{3,3}(u)}{t_5 - t_3} + \frac{(t_7 - u)N_{4,3}(u)}{t_7 - t_4}
\]

\[
N_{3,4}(u) = \frac{(u - t_2)N_{3,3}(u)}{3 - 0} + \frac{(4 - u)N_{4,3}(u)}{4 - 1}
\]

\[
N_{3,4}(u) = \frac{1}{3}uN_{3,3}(u) + \frac{1}{3}(4 - u)N_{4,3}(u)
\]

\[
N_{3,4}(u) = \frac{1}{3}\left(u\right)\left[\frac{1}{2}u^2N_{3,1}(u) + \frac{1}{2}\left(-2u^2 + 6u - 3\right)N_{4,1}(u) + \frac{1}{2}(3 - u)^2N_{5,1}(u)\right] + \frac{1}{3}(4 - u)\left[\frac{1}{2}(u - 1)^2N_{4,1}(u) + \frac{1}{2}\left(-2u^2 + 10u - 11\right)N_{5,1}(u) + \frac{1}{2}(4 - u)^2N_{6,1}(u)\right]
\]

\[
N_{3,4}(u) = \frac{1}{6}u^3N_{3,1}(u) + \frac{1}{6}\left(-3u^3 + 12u^2 - 12u + 4\right)N_{4,1}(u) + \frac{1}{6}\left(3u^3 - 24u^2 + 40u - 44\right)N_{5,1}(u) + \frac{1}{6}(4 - u)^3N_{6,1}(u)
\]

For $N_{4,4}(u)$

\[
N_{4,4}(u) = \frac{(u - t_4)N_{4,3}(u)}{t_7 - t_4} + \frac{(t_7 - u)N_{5,3}(u)}{t_7 - t_5}
\]

\[
N_{4,4}(u) = \frac{(u - 1)N_{4,3}(u)}{4 - 1} + \frac{(4 - u)N_{5,3}(u)}{4 - 2}
\]
\begin{align*}
N_{4,4}(u) &= \frac{1}{3}(u - 1) N_{4,3}(u) + \frac{1}{2}(4 - u) N_{5,3}(u) \\
N_{4,4}(u) &= \frac{1}{3}(u - 1) \left[ \frac{1}{2}(u - 1)^2 N_{4,4}(u) + \frac{1}{2}(-2u^2 + 10u - 11) N_{5,1}(u) + \frac{1}{2}(4 - u)^2 N_{6,1}(u) \right] + \\
&\quad \frac{1}{2}(4 - u) \left[ \frac{1}{2}(u - 2)^2 N_{5,1}(u) + \frac{1}{2}(-3u^2 + 20u - 32) N_{6,1}(u) \right] \\
N_{4,4}(u) &= \frac{1}{6}(u - 1)^3 N_{4,1}(u) + \frac{1}{12}(-7u^3 + 48u^2 - 102u + 76) N_{5,1}(u) + \frac{1}{12}(11u^3 - 114u^2 + 384u - 416) N_{6,1}(u) \\
\end{align*}

For \( N_{5,4}(u) \)
\begin{align*}
N_{5,4}(u) &= \frac{(u - t_5) N_{3,3}(u)}{t_6 - t_5} + \frac{(t_9 - u) N_{6,3}(u)}{t_9 - t_5} \\
N_{5,4}(u) &= \frac{(u - 2) N_{5,3}(u)}{4 - 2} + \frac{(4 - u) N_{5,3}(u)}{4 - 3} \\
N_{5,4}(u) &= \frac{1}{2}(u - 2) N_{5,3}(u) + (4 - u) N_{6,3}(u) \\
N_{5,4}(u) &= \frac{1}{2}(u - 2) \left[ \frac{1}{2}(u - 2)^2 N_{5,1}(u) + \frac{1}{2}(-3u^2 + 20u - 32) N_{6,1}(u) \right] + (4 - u) \left[ (u - 3)^2 N_{6,1}(u) \right] \\
N_{5,4}(u) &= \frac{1}{4}(u - 2)^3 N_{5,1}(u) + \frac{1}{4}(-7u^3 + 66u^2 - 204u + 208) N_{6,1}(u) \\
\end{align*}

For \( N_{6,4}(u) \)
\begin{align*}
N_{6,4}(u) &= \frac{(u - t_6) N_{6,3}(u)}{t_9 - t_6} + \frac{(t_{10} - u) N_{7,3}(u)}{t_{10} - t_7} \\
N_{6,4}(u) &= \frac{(u - 3) N_{6,3}(u)}{4 - 3} + \frac{(4 - u) N_{7,3}(u)}{4 - 4} \\
N_{6,4}(u) &= (u - 3) N_{6,3}(u) \\
N_{6,4}(u) &= (u - 3) \left[ (u - 3)^2 N_{6,1}(u) \right] \\
N_{6,4}(u) &= (u - 3)^3 N_{6,1}(u) \\
\end{align*}
Appendix B: import_ct_data.m

% import_ct_data.m was used to import the CT data from bmp files into a
% multidimensional array CT. The files used were slice number 1634 to 1834.

prefix = input ('Enter the prefix before the slice number in the file name: ');
first_slice = input ('Enter the slice number of the first slice: ');
last_slice = input ('Enter the slice number of the last slice: ');
format = input ('Enter the format of the image: ');
slice_num = first_slice;
n=1;
while slice_num<=last_slice
    slice_str = num2str (slice_num);
    CT(:,:,n) = imread((strcat(prefix,slice_str,format)));
    slice_num = slice_num + 1;
    n=n+1;
end

CT(:,500:512,:,:)=[];
CT(:,1:299,:,:)=[];
CT(300:512,:,:)=[];
CT(1:74,:,:)=[];
CT(:,2:3,:)=[];
CT=squeeze(CT);
Appendix C: separateEntities.m

% The approach was to start with an arbitrary face and add the vertices of
% the face to a new array. Then search the original array of faces to find
% any unused faces that share any of the newly added vertices. These faces
% would be marked as used and added to a new array of faces. This was
% performed on an iterative basis.

f_temp = f;
v_temp = v;

f_new(1,:) = f_temp(1,:);
f_temp(1,:)= [nan nan nan];

i_new = 1;
i = 1;
i_v = 1;
index_i_v = 0;

while i_new <= length(f_new(:,1))
    j_new = 1:
    while j_new <= 3
        % Search through f_temp to find a face that utilizes the
        % same vertex as in f_new.
        while i <= length(f_temp(:,1))
            j = 1;
            while j <=3;
                if (f_temp(i,j)==f_new(i_new,j_new))
                    % Store these vertices in f_new.
                    f_new(length(f_new(:,1))+1,:) = f_temp(i,:);
                    % Delete that face from f_temp.
                    f_temp(i,:)= [nan nan nan];
                end
                j = j+1;
            end
            i = i+1;
        end
        i = 1;
        i_replace_v = 1;
        vertexToReplace = f_new(i_new,j_new);
        if (f_new(i_replace_v,j)>0)
            v_new(i_v,:) = v_temp(f_new(i_new,j_new),:);
        end
        % Store and renumber the vertices.
        while i_replace_v <= length(f_new(:,1))
            j = 1;
            while j <=3;
                if (f_new(i_replace_v,j) == vertexToReplace)
                    if (f_new(i_replace_v,j)>0)
                        f_new(i_replace_v,j) = -i_v; % Mark as used.
                        index_i_v = 1;
                    end
                end
                j = j+1;
            end
            i_replace_v = i+1;
        end
    end
end

% Search through f_temp to find a face that utilizes the
% same vertex as in f_new.
while i <= length(f_temp(:,1))
    j = 1;
    while j <=3;
        if (f_temp(i,j)==f_new(i_new,j_new))
            % Store these vertices in f_new.
            f_new(length(f_new(:,1))+1,:) = f_temp(i,:);
            % Delete that face from f_temp.
            f_temp(i,:)= [nan nan nan];
        end
        j = j+1;
    end
    i = i+1;
end
end
end
j=j+1;
end
i_replace_v = i_replace_v + 1;
end
if (lndexj_v )
  l_v = length(v_new(:,1))+1;
end
index_i_v = 0;
j_new = j_new + 1;
end
i_new = i_new + 1;
end

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Appendix D: separateBones.m

% Find the separate entities that are connected by a CT number of > 170
% These entities are then each expanded by one layer of voxels.
% This expansion continues as long as the voxels are greater than 120 or
% until all of the voxels are used.

CT(1:2,:,:,:) = 0;
CT(length(CT(:,1,1))-1:length(CT(:,1,1)),:,:)=0;
CT(:,1:2,:) = 0;
CT(:,length(CT(:,1,1))-1:length(CT(:,1,1)),:,:)=0;
CT(:,1:2,:) = 0;
CT(:,length(CT(:,1,1))-1:length(CT(:,1,1)),:,:)=0;
CT(:,1:2,:) = 0;
CT(:,length(CT(1,:,1))-1:length(CT(1,:,1)),:,:)=0;
CTbone = CT;
% All of the outer values are set to zero to ensure searching in a valid range

kCT = 1;
D = sparse(length(CT(:,1,1)),length(CT(:,1,1)));
while kCT <= length(CT(1,1,:))
  iCT = 1;
  while iCT <= length(CT(:,1,1))
    jCT = 1;
    while jCT <= length(CT(1,:,1))
      if CT(iCT,jCT,kCT) >= 170
        D(iCT,jCT) = CT(iCT,jCT,kCT);
      end
      jCT = jCT + 1;
    end
    iCT = iCT + 1;
  end
  DBone{kCT} = D;
  D = sparse(length(CT(:,1,1)),length(CT(:,1,1)));
  kCT = kCT + 1;
end
% Find the first non-zero value and expand this to yield a new entity
for kSlices = 1:length(CT(1,1,:))
  Entity{kSlices} = sparse(length(CT(:,1,1)),length(CT(:,1,1)));
end
allEntities = 0;
iEntities = 1;
while allEntities == 0
  kDBone = 1;
lengthi = 0;
i = [];
j = [];

while lengthi == 0
    [i j] = find(Dbone(kDbone));
    lengthi = length(i);
    k = kDbone;
    kDbone = kDbone + 1;
    if kDbone >= length(CT(1,1,:))
        allEntities = 1;
        break
    end
end
if lengthi ~= 0
    i = i(1);
    j = j(1);
    bonelndices(1,:) = [i j k];
    Dbone{k}(i,j) = 0;
    ilindices = 1;
while ilindices <= length(bonelndices(:,1))
    i = bonelndices(ilindices,1);
    j = bonelndices(ilindices,2);
    k = bonelndices(ilindices,3);
    front = Dbone{k-1}(i,j);
    back = Dbone{k+1}(i,j);
    left = Dbone{k}(i-1,j);
    right = Dbone{k}(i+1,j);
    top = Dbone{k}(i,j+1);
    bottom = Dbone{k}(i,j-1);
    if (front ~= 0)
        Entity{k-1}(i,j) = front;
        Dbone{k-1}(i,j) = 0;
        bonelndices(length(bonelndices(:,1))+1,:) = [i j k-1];
    end
    if (back ~= 0)
        Entity{k+1}(i,j) = back;
        Dbone{k+1}(i,j) = 0;
        bonelndices(length(bonelndices(:,1))+1,:) = [i j k+1];
    end
    if (left ~= 0)
        Entity{k}(i-1,j) = left;
        Dbone{k}(i-1,j) = 0;
        bonelndices(length(bonelndices(:,1))+1,:) = [i-1 j k];
    end
    if (right ~= 0)
        Entity{k}(i+1,j) = right;
        Dbone{k}(i+1,j) = 0;
        bonelndices(length(bonelndices(:,1))+1,:) = [i+1 j k];
    end
    if (top ~= 0)
        Entity{k}(i,j+1) = top;
        Dbone{k}(i,j+1) = 0;
        bonelndices(length(bonelndices(:,1))+1,:) = [i j+1 k];
    end
    if (bottom ~= 0)
        Entity{k}(i,j-1) = bottom;
        Dbone{k}(i,j-1) = 0;
        bonelndices(length(bonelndices(:,1))+1,:) = [i j-1 k];
    end
end
if (bottom ~= 0)
    Entity{k}(l,j-1) = bottom;
    Dbone{k}(l,j-1) = 0;
    bonelndices(length(bonelndices(:,1))+1,:) = [i j-1 k];
end

iindices = iindices + 1;
end

if length(bonelndices(:,1))>=10
    EntityBoneIndices{iEntitles} = boneindices;
    Entitles{iEntitles} = Entity;
    iEntitles = iEntitles + 1;
end

clear boneindices;
% Clear Entity to contain kSlices sparse matrices
for kSlices = 1:length(CT(:,1,:))
    Entity{kSlices} = sparse(length(CT(:,1,1)),length(CT(:,1,1)));
end
end
end

for I  = 1:length(Entitles)
    boneindlcesLength(Entity,1 ) = [1 length(EntitlesBoneIndices{i}(1,:))];
end

if length(bonelndices(:,1))>=10
    EntityBoneIndices{iEntitles} = boneindices;
    Entitles{iEntitles} = Entity;
    iEntitles = iEntitles + 1;
end

clear boneindices;
% Clear Entity to contain kSlices sparse matrices
for kSlices = 1:length(CT(:,1,:))
    Entity{kSlices} = sparse(length(CT(:,1,1)),length(CT(:,1,1)));
end
end

for i = 1:length(Entitles)
    if(bonelndicesLength(Entity,1) ~= boneindlcesLength(Entity,2));
        more = 1;
    end
end

if (more)
    while iEntity <= length(Entitles)
        iIndices = boneindlcesLength(iEntity,1);
        more = 1;
        while more == 1
            iEntity = 1;
            while iEntity <= (length(Entitles))
                CTbone = 1;
                while iCTbone <= length(EntitlesBoneIndices(iEntity(:,1))
                    CTbone(EntitlesBoneIndices(iEntity)(iCTbone,:)) = 0;
                iCTbone = iCTbone + 1;
            end
            iEntity = iEntity+1;
        end
    end

boneValue = 100;

iEntity = 1;
if(bonelndicesLength(Entity,1) ~= boneindlcesLength(Entity,2));
    more = 1;
end

if (more)
    while iEntity <= length(Entitles)
indices = 1;

while indices <= boneIndicesLength(iEntity,2);

    i = EntityBoneIndices{iEntity}(indices,1);
    j = EntityBoneIndices{iEntity}(indices,2);
    k = EntityBoneIndices{iEntity}(indices,3);

    if (i>=2)&(j>=2)&(k>=2)
        front = CTbone(i,j,k-1);
        back = CTbone(i,j,k+1);
        left = CTbone(i-1,j,k);
        right = CTbone(i+1,j,k);
        top = CTbone(i,j+1,k);
        bottom = CTbone(i,j-1,k);

        if (Entities{iEntity}{k}(i,j) >= 120)
            if (front > 0)
                Entities{iEntity}{k-1}(i,j) = front;
                CTbone(i,j,k-1) = 0;
                EntityBoneIndices{iEntity}(length(EntityBoneIndices{iEntity}(,1))+1,:) = [i j k-1];
            end
            if (back > 0)
                Entities{iEntity}(k+1)(i,j) = back;
                CTbone(i,j,k+1) = 0;
                EntityBoneIndices{iEntity}(length(EntityBoneIndices{iEntity}(,1))+1,:) = [i j k+1];
            end
            if (left > 0)
                Entities{iEntity}{k}(i-1,j) = left;
                CTbone(i-1,j,k) = 0;
                EntityBoneIndices{iEntity}(length(EntityBoneIndices{iEntity}(,1))+1,:) = [i-1 j k];
            end
            if (right > 0)
                Entities{iEntity}{k}(i+1,j) = right;
                CTbone(i+1,j,k) = 0;
                EntityBoneIndices{iEntity}(length(EntityBoneIndices{iEntity}(,1))+1,:) = [i+1 j k];
            end
            if (top > 0)
                Entities{iEntity}(k)(i,j+1) = top;
                CTbone(i,j+1,k) = 0;
                EntityBoneIndices{iEntity}(length(EntityBoneIndices{iEntity}(,1))+1,:) = [i j+1 k];
            end
            if (bottom > 0)
                Entities{iEntity}(k)(i,j-1) = bottom;
                CTbone(i,j-1,k) = 0;
                EntityBoneIndices{iEntity}(length(EntityBoneIndices{iEntity}(,1))+1,:) = [i j-1 k];
            end
            else
                if (front > 0)
                    Entities{iEntity}{k-1}(i,j) = front;
                    CTbone(i,j,k-1) = 0;
                    EntityBoneIndices{iEntity}(length(EntityBoneIndices{iEntity}(,1))+1,:) = [-i -j -(k-1)];
                end
                if (back > 0)
                    Entities{iEntity}(k+1)(i,j) = back;
                    CTbone(i,j,k+1) = 0;
                    EntityBoneIndices{iEntity}(length(EntityBoneIndices{iEntity}(,1))+1,:) = [-i -j -(k+1)];
                end
        end
    end
end
if (left > 0)
   Entities(iEntity)(i-1,j) = left;
   CTbone(i-1,j,k) = 0;
   EntityBoneIndices(iEntity)(length(EntityBoneIndices(iEntity)(1,:))+1,:) = [-i-1 -j -k];
end
if (right > 0)
   Entities(iEntity)(i+1,j) = right;
   CTbone(i+1,j,k) = 0;
   EntityBoneIndices(iEntity)(length(EntityBoneIndices(iEntity)(1,:))+1,:) = [i+1 -j -k];
end
if (top > 0)
   Entities(iEntity)(i,j+1) = top;
   CTbone(i,j+1,k) = 0;
   EntityBoneIndices(iEntity)(length(EntityBoneIndices(iEntity)(1,:))+1,:) = [-i -j+1 -k];
end
if (bottom > 0)
   Entities(iEntity)(i,j-1) = bottom;
   CTbone(i,j-1,k) = 0;
   EntityBoneIndices(iEntity)(length(EntityBoneIndices(iEntity)(1,:))+1,:) = [-i -(j-1) -k];
end
indices = indices + 1;
end
boneIndicesLength(iEntity,:) = [boneIndicesLength(iEntity,2) length(EntityBoneIndices(iEntity)(1,:))]
% boneIndicesLength [2ndLastLength lastLength] with rows corresponding to the entity number
iEntity = iEntity + 1;
Appendix E: TransformGridResample

% TransformGridResample

% Resample the CT data with new points orientated along the index finger.

% Rotate about the z-axis
Rzback = [.85221 -.52319 0; .52319 .85221 0; 0 0 1];

% Rotate about the x-axis
Rxback = [1 0 0; 0 -.21175 .97732; 0 -.97732 -.21175];

% Translate back to starting position
T2 = [1 0 0 40; 0 1 0 38; 0 0 1 167; 0 0 0 1];

[X Y Z] = meshgrid(1:1:60, -20:1:20, -30:1:30);

% The size of the arrays from meshgrid were (41,60,61)
% The expected size was (60,41,61). The following will reorder the columns.
for i = 1:60
    X1(i,:) = X(i,:);
    Y1(i,:) = Y(i,:);
    Z1(i,:) = Z(i,:);
end

% Perform the transformations.

j=1;

while k<=length(X1(1,1,:))
    j = 1;
    while j<=length(X1(1,:,1))
        B = Rzback*[X1(:,j,k) Y1(:,j,k) Z1(:,j,k)];
        B = Rxback*[B];
        B(4,:) = 1;
        B = T2 * B;
        B(4,:)=[];
        X2(:,j,k)=B(:,1);
        Y2(:,j,k)=B(:,2);
        Z2(:,j,k)=B(:,3);
        j=j+1;
    end
    k=k+1;
end

% Perform the linear interpolation
% VI is then a multidimensional array with the values from interpolation.
[VI] = interp3(CT,X2,Y2,Z2);

% This dataset was then used to create the contours.
Appendix F: From Knot Points to Control Points

For a closed B-Spline curve with $K = 4$ the $i^{th}$ curve segment can be defined as:

$$p_i(u) = \frac{1}{6} \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_i \text{mod}(n+1) \\ p_{(i-1)} \text{mod}(n+1) \\ p_{(i+1)} \text{mod}(n+1) \\ p_{(i+2)} \text{mod}(n+1) \end{pmatrix}$$

The transition from the previous segment to this segment occurs at $u=0$. This transition or knot point is defined:

$$p_i(0) = \frac{1}{6} \begin{pmatrix} 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_{(i-1)} \text{mod}(n+1) \\ p_i \text{mod}(n+1) \\ p_{(i+1)} \text{mod}(n+1) \\ p_{(i+2)} \text{mod}(n+1) \end{pmatrix}$$

$$p_{i+1}(0) = \frac{1}{6} \begin{pmatrix} 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_i \text{mod}(n+1) \\ p_{(i+1)} \text{mod}(n+1) \\ p_{(i+2)} \text{mod}(n+1) \\ p_{(i+3)} \text{mod}(n+1) \end{pmatrix}$$

$$p_{i+2}(0) = \frac{1}{6} \begin{pmatrix} 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_{(i+1)} \text{mod}(n+1) \\ p_{(i+2)} \text{mod}(n+1) \\ p_{(i+3)} \text{mod}(n+1) \\ p_{(i+4)} \text{mod}(n+1) \end{pmatrix}$$

The $(n-1)^{th}$ curve segment utilizes the $P_0$ control point. Note that $(n+1)\text{mod}(n+1) = 0$.

$$p_{n-1}(0) = \frac{1}{6} \begin{pmatrix} 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_{(n-2)} \text{mod}(n+1) \\ p_{(n-1)} \text{mod}(n+1) \\ p_n \text{mod}(n+1) \\ p_{(n+1)} \text{mod}(n+1) \end{pmatrix}$$

79
The $n^{th}$ curve segment utilizes control point $P_0$ and $P_1$.

$$P_n(0) = \frac{1}{6} \begin{pmatrix} 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} P_{(n-1) \mod (n+1)} \\ P_{(n) \mod (n+1)} \\ P_{(n+1) \mod (n+1)} \\ P_{(n+2) \mod (n+1)} \end{pmatrix}$$

And the $(n+1)^{th}$ curve segment utilizes control point $P_0$, $P_1$, and $P_2$.

$$P_{n+1}(0) = \frac{1}{6} \begin{pmatrix} 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} P_{(n) \mod (n+1)} \\ P_{(n+1) \mod (n+1)} \\ P_{(n+2) \mod (n+1)} \\ P_{(n+3) \mod (n+1)} \end{pmatrix}$$

Which can be written in matrix form as:

$$\begin{bmatrix} P_0(0) \\ P_{i+1}(0) \\ P_{i+2}(0) \\ \vdots \\ P_{n-1}(0) \\ P_n(0) \\ P_{n+1}(0) \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 4 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 4 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 4 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 1 & 4 \\ 4 & 1 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ \vdots \\ P_{n-1} \\ P_n \\ P_{n+1} \end{bmatrix}$$

The control points can then be found as:

$$\begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ \vdots \\ P_{n-1} \\ P_n \\ P_{n+1} \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 4 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 4 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 4 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 1 & 4 \\ 4 & 1 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} P_0(0) \\ P_{i+1}(0) \\ P_{i+2}(0) \\ \vdots \\ P_{n-1}(0) \\ P_n(0) \\ P_{n+1}(0) \end{bmatrix}$$
Appendix G: Fit a B-Spline Curve to Contour

G.1: fitBspline

function [P_x, P_y] = fitBspline(P,num_u)

% Use the vertices of P as knot points of the B-spline curve
% and determine the control points from these knot points.
% Then find num_u points on each segment of the B-Spline.

% P_x & P_y are Matrices of the x and y values of the contour
% where the each column is a new segment
% and the rows are the x or y values evaluated at each u-value
% P_x where P_x(u,i) is the x value on the ith segment at u=u

if nargin == 1
    num_u = 21; % default value for the resolution of u
end

num_seg = length(P(:,1));
CP = knot_to_cntrl(num_seg,P); % Appendix G.2
x = x_mod(CP,num_seg); % Appendix G.3
y = y_mod(CP,num_seg); % Appendix G.4
N = N_array(num_u); % Appendix G.5
P_x = P_x_u_segments(N,num_seg,num_u,x); % Appendix G.6
P_y = P_y_u_segments(N,num_seg,num_u,y); % Appendix G.7
G.2: knot_to_cntrl

function P = knot_to_cntrl(num_seg,KP)

% This function will convert from knot points to the control points.
% num_seg is the number of segments in the B-Spline
% The input KP are the knot points
% The output P are the control points

U_M=zeros(num_seg,num_seg)
i=0
while i < num_seg
    U_M(i+1,(mod(i,num_seg)+1))=1/6;
    U_M(i+1,(mod(i+1,num_seg)+1))=4/6;
    U_M(i+1,(mod(i+2,num_seg)+1))=1/6;
    U_M(i+1,(mod(i+3,num_seg)+1))=0;
    i=i+1;
end

P = inv(U_M)*KP;
G.3: x_mod

function x = x_mod(P,num_seg)

% Create an array x which defines the control points
% used to define each segment of the B-Spline

% x = (2, 1, 1(u value), segment number)
% P is an array of the contour vertices
% P = (num_vertices, 2)

c_x = P(:,1);
x = zeros(4,1,1,num_seg);
k = 1;
while k <= num_seg
    for i = 1:4
        x(i,:,:,k) = c_x(mod(i+k-2,num_seg)+1);
    end
    k = k + 1;
end

G.4: y_mod

function y = y_mod(P,num_seg)

% Create an array y which defines the control points
% used to define each segment of the B-Spline

% y = (2, 1, 1(u value), segment number)
% P is an array of the contour vertices
% P = (num_vertices, 2)

c_y = P(:,2);
y = zeros(4,1,1,num_seg);
k = 1;
while k <= num_seg
    for i = 1:4
        y(i,:,:,k) = c_y(mod(i+k-2,num_seg)+1);
    end
    k = k + 1;
end
G.5: N_array

function N = N_array(num_u)

% Generate the N array for the B-Spline. This array is multiplied by
% the x array and y array to generate points on the B-Spline at the u values
% N = (1, 2, u value)
% num_u is a scalar defining the resolution of the u values
% which define the B-Spline. du = 1/(num_u-1)

u = 0;
k = 1;
N = zeros(1,4,num_u);
while k <= num_u
    N_1_4 = 1/6*(-u.^3+3*u.^2-3*u+1);
    N_2_4 = 1/6*(3*u/3-6*u/2+4);
    N_3_4 = 1/6*(-3*u.^3+3*u/2+3*u+1);
    N_4_4 = 1/6*(u.'^3);
    M = [N_1_4 N_2_4 N_3_4 N_4_4];
    N(:,:,k) = M;
    u = u+(1/(num_u-1));
k = k+1;
end
**G.6: P\_x\_u\_segments**

function \( P_x = P_x_u_segments(N, \text{num\_seg}, \text{num\_u}, x) \);

% \( P_x \) is a Matrix of the x values of the B-spline
% where the each column is a new segment
% and the rows are the x-values evaluated at each u-value

% \( N(1, 4, u \text{ value}) \) & \( x(4, 1, 1, \text{segments}) \)
% \( P_x \) where \( P_x(u,i) \) is the x value on the ith segment at \( u=u \)

\[ k = 1; \]
\[ \text{while } k <= \text{num\_u} \]
\[ \quad \text{for } i = 1: \text{num\_seg} \]
\[ \quad \quad P_x(:,:,k,i) = N(:,:,k)*x(:,:,1,i); \]
\[ \quad \text{end} \]
\[ \quad k = k+1; \]
\[ \text{end} \]

\[ P_x = \text{squeeze}(P_x); \]

**G.7: P\_y\_u\_segments**

function \( P_y = P_y_u_segments(N, \text{num\_seg}, \text{num\_u}, y) \);

% \( P_y \) is a Matrix of the y-values of the B-spline
% where the each column is a new segment
% and the rows are the y-values evaluated at each u-value

% \( N(1, 4, u \text{ value}) \) & \( y(4, 1, 1, \text{segments}) \)
% \( P_y \) where \( P_y(u,i) \) is the y value on the ith segment at \( u=u \)

\[ k = 1; \]
\[ \text{while } k <= \text{num\_u} \]
\[ \quad \text{for } i = 1: \text{num\_seg} \]
\[ \quad \quad P_y(:,:,k,i) = N(:,:,k)*y(:,:,1,i); \]
\[ \quad \text{end} \]
\[ \quad k = k+1; \]
\[ \text{end} \]

\[ P_y = \text{squeeze}(P_y); \]
Appendix H: Resample the Contour at Equal Spaced Points

H.1: resampleContour

function P_s = resampleContour(Ctour,num_u)

% Resample a contour to generate equally spaced points

% Ctour is an array of the contour to be resampled
% Ctour = (num_vertices, 2)

% num_u is a scalar defining the resolution of the u values
% which define the contour lines. du = 1/(num_u-1)

if nargin == 1
    num_u = 21;
end

num_seg = length(Ctour(:,1))-1;

X = contour_x(Ctour,num_seg); % Appendix H.2
y = contour_y(Ctour,num_seg); % Appendix H.3

N = contour_N_array(num_u); % Appendix H.4
P_x = contour_P_x_u_segments(N,num_seg,num_u,x); % Appendix H.5
P_y = contour_P_y_u_segments(N,num_seg,num_u,y); % Appendix H.6

u_seg_start = [1 1]; % set the start point to be equal to the start
% of the original contour

len_s = contour_per_spaced(x,y,u_seg_start, num_u, num_seg); % Appendix H.7
P_s = contour_per_spaced_pnts(x,y,num_seg,len_s,P_x,P_y,u_seg_start); % Appendix H.8
H.2: contour_x

function x = contour_x(Ctour,num_seg)

% Create an array x which defines the x values of the
% points used to define each segment of the contour

% x = (2, 1, 1(u value), segment number)
% Ctour is an array of the contour vertices
% Ctour = (num_vertices, 2)

c_x = Ctour(:,1);
x = zeros(2,1,1,num_seg);

k = 1;
while k <= num_seg
x(1,:,:,:) = c_x(k);
x(2,:,:,:) = c_x(k+1);
k = k+1;
end

H.3: contour_y

function y = contour_y(Ctour,num_seg)

% Create an array y which defines the y values of the
% points used to define each segment of the contour

% y = (2, 1, 1(u value), segment number)
% Ctour is an array of the contour vertices
% Ctour = (num_vertices, 2)

c_y = Ctour(:,2);
y = zeros(2,1,1,num_seg);

k = 1;
while k <= num_seg
y(1,:,:,:) = c_y(k);
y(2,:,:,:) = c_y(k+1);
k = k+1;
end
H.4: contour_N_array

function N = contour_N_array(num_u)

% Generate the N array for the contour. This array is multiplied by
% the x array and y array to generate points on the contour at the u values

% N = (1, 2, u value)

% num_u is a scalar defining the resolution of the u values
% which define the contour lines. du = 1/(num_u-1)

u = 0;
k = 1;
N = zeros(1,2,num_u);

while k <= num_u
    N(:,:,k) = [1-u u];
    u = u + (1/(num_u-1)); % Remember that u=0, 1/(num_u-1).. 1
    k = k+1;
end
H.5: contour_P_x_u_segments

function P_x = contour_P_x_u_segments(N, num_seg, num_u, x);

% P_x is a Matrix of the x values of the contour
% where the each column is a new segment
% and the rows are the x-values evaluated at each u-value
% P_x where P_x(u,i) is the x value on the ith segment at u=u
% N(1, 4, u value) & x(4, 1, 1, segments)

k = 1;
while k <= num_u
    for i = 1:num_seg
        P_x(:,:,k,l) = N(:,:,k)*x(:,:,1,l);
    end
    k = k+1;
end

P_x = squeeze(P_x);

H.6: contour_P_y_u_segments

function P_y = contour_P_y_u_segments(N, num_seg, num_u, y);

% P_y is a Matrix of the y values of the contour
% where the each column is a new segment
% and the rows are the y-values evaluated at each u-value
% P_y where P_y(u,i) is the y value on the ith segment at u=u
% N(1, 4, u value) & y(4, 1, 1, segments)

k = 1;
while k <= num_u
    for i = 1:num_seg
        P_y(:,:,k,i) = N(:,:,k)*y(:,:,1,i);
    end
    k = k+1;
end

P_y = squeeze(P_y);
H.7: contour_per_spaced

function len_s = contour_per_spaced(x,y,u_seg_start, num_u, num_seg)

% Create an Array of the current length of the contour starting
% at u_start

% This current length is taken from the start point to a point on the curve
% for each u value on every segment.

% Remember that for an Array of u values u(1)=0 and u(num_u)=1
% The actual u value for u(n) = (n-1)/(num_u-1)

% len_s is an array of the current length at a given u value for a given segment.
% This is developed starting at the upper most point above the centroid.

% len_s = (current length, u value, segment number)

% num_u is a scalar defining the resolution of the u values
% which define the contour lines. du = 1/(num_u-1)

u_start = (u_seg_start(1,1)-1)/(num_u-1);
seg_start = u_seg_start(1,2);
u_1 = u_start;
i_1 = seg_start;
inc = 1/(num_u-1);
u_2 = u_1+inc;
i_2 = i_1;
n = 1;

while i_2 <= num_seg
    while u_2 <= (1+inc)
        len_s(n,1) = contour_perimeter(x,y,num_seg,u_1,u_2,i_1,i_2);
        len_s(n,2) = (u_2*(num_u-1));
        len_s(n,3) = i_2;
        n = n+1;
        u_2 = u_2+inc;
    end
    if u_2 > 1
        u_2 = inc;
        i_2 = i_2+1;
    end
end
%reverse the order of finding the perimeter
i_2 = 1;
i_1 = 1;
u_1 = 0;
u_2 = inc;

while u_2 <= u_start & i_2 <= seg_start
    while u_2 <= (1+inc)
        len_s(n,1) = contour_perimeter(x,y,num_seg,u_start,u_2,i_1,i_2) + ...
                       contour_perimeter(x,y,num_seg,u_1,u_2,i_1,i_2);
        len_s(n,2) = ((u_2*(num_u-1)));
        len_s(n,3) = i_2;
        u_2 = u_2 + inc;
end

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\begin{verbatim}
n = n+1;
end
if u_2 > 1 & i_2+1 ~= seg_start
    u_2 = inc;
i_2 = i_2+1;
else
    i_2 = i_2+1;
end
end

i_1 = 1;
u_1 = 0;
u_2 = inc;
while u_2 <= u_start
    len_s(n,1) = contour_perimeter(x,y,num_seg,u_start,1,seg_start,num_seg) + ...
    contour_perimeter(x,y,num_seg,0,1,1,(seg_start-1)) + ...
    contour_perimeter(x,y,num_seg,0,u_2,seg_start,seg_start); \% Appendix H.7.1
    len_s(n,2) = ((u_2*(num_u-1)));
    len_s(n,3) = i_2;
u_2 = u_2+inc;
n = n+1;
end
\end{verbatim}
H.7.1: contour_perimeter

function per = contour_perimeter(x,y,num_seg,u_1,u_2,i_1,i_2)

% Return the perimeter of the contour between u_1 on the
% i_1th segment and u_2 on the i_2th segment.
% If i_1 and i_2 are not specified it assumes i_1 =1 and i_2 =num_seg

if nargin ==5
    i_1=1;
    i_2=num_seg;
end

clear per
per=zeros(1,1,(i_2-i_1 +1 ));
i=1;

if i_1 == i_2
    i=i_1;
    per(1,1,i) = sqrt(((x(2,1,1,i) - x(1,1,1,i))*(u_2 - u_1))^2 + ((y(2,1,1,i) -...y(1,1,1,i))*(u_2-0))^2);
else
    i=i_1;
    while i <=i_2
        if i == i_1
            per(1,1,i) = sqrt(((x(2,1,1,i) - x(1,1,1,i))*(1))^2 + ((y(2,1,1,i) -...y(1,1,1,i))*(1))^2);
            i=i+1;
        end
        if i < i_2
            per(1,1,i) = sqrt(((x(2,1,1,i) - x(1,1,1,i))*(1))^2 + ((y(2,1,1,i) -...y(1,1,1,i))*(1))^2);
            i=i+1;
        end
        if i == i_2
            per(1,1,i) = sqrt(((x(2,1,1,i) - x(1,1,1,i))*(u_2 - 0))^2 + ((y(2,1,1,i) -...y(1,1,1,i))*(u_2-0))^2);
            i=i+1;
        end
    end
end

per = sum(per(:,:));
H.8: contour_per_spaced_pnts

function Pnts_spaced = contour_per_spaced_pnts(x,y,num_seg,len_s,P_x,P_y,u_seg_start)

% Find the u values and segments for the points that are equally spaced along the perimeter of the contour

% Pnts_spaced = per_spaced_pnts(x,y,num_seg,len_s,P_x,P_y,u_seg_start)
% Pnts_spaced is an array of num_pnts points that are equally spaced on the contour.

% len_s is an array of the current length at a given u value for a given segment.
% len_s = (current length, u value, segment number)

% P_x & P_y are arrays of x and y values of points on the contour.
% u_seg_start = (u value, segment) for the start position

per_full = contour_perimeter(x,y,num_seg,0,1); %The full length of the perimeter
num_pnts = 42; %number of equally spaced points to be fit

n = 1;
size_len_s = length(len_s(:,1));

% Appended to the bottom of the array len_s are equal portions of the total length of the perimeter (per_full/num_pnts)*n.

while n < num_pnts
    len_s(size_len_s+n,1) = (per_full/num_pnts)*n;
    len_s(size_len_s+n,2) = nan;
    len_s(size_len_s+n,3) = nan;
    n = n+1;
end

% The equally spaced lengths are sorted into len_s (the current length at a given u values and segments)

len_s = sortrows(len_s);

size_len_s = length(len_s(:,1));

Pnts_spaced(1,1) = P_x(u_seg_start(1,1), u_seg_start(1,2));
Pnts_spaced(1,2) = P_y(u_seg_start(1,1), u_seg_start(1,2));

% Identify where the equal spaced lengths were sorted into the len_s array and find the P_x points at u value and segment number

k = 2;
n = 2;
while n <= size_len_s
    if isnan(len_s(n,2))
        Pnts_spaced(k,1) = P_x(int8(len_s((n-1),2)),int8(len_s((n-1),3)));
        Pnts_spaced(k,2) = P_y(int8(len_s((n-1),2)),int8(len_s((n-1),3)));
        k = k+1;
    end
    n = n+1;
end
Appendix I: Finding the Centroid of the Closed B-Spline Curve

I.1: centroid_x

function cen_x = centroid_x(Area, num_seg,y,x);

% Computes the x centroid for the closed B-Spline curve

% Area is the area of the B-Spline curve
% num_seg is the number of segments in the B-Spline

% y is an array of the Control Points in the y
% x is an array of the Control Points in the x

E=1/6*[-1 3 -3 1; 3 -6 3 0; -3 0 3 0; 1 4 1 0];
F=1/6*[-3 9 -9 3; 6 -12 6 0; -3 0 3 0];
for i=1:num_seg
A(:,:,i)=E(:,:)*y(:,:,1,i);
B(:,:,i)=F(:,:)*x(:,:,1,i);
C(:,:,i)=E(:,:)*x(:,:,1,i);
end
clear E
clear F
for i=1:num_seg
  cen_x(1,1,i)=(1/9*(C(:,:,i)*A(:,:,i)+B(:,:,i)))+...
  (1/8*(C(:,:,i)*A(:,:,1)*B(:,:,i)+C(:,:,1)*A(:,:,i)*B(:,:,1)+C(:,:,1)*A(:,:,i)*B(:,:,1)))+...
  (1/7*(C(:,:,i)*A(:,:,1)*B(:,:,1)+C(:,:,1)*A(:,:,i)*B(:,:,1)+C(:,:,1)*A(:,:,i)*B(:,:,1)))+...
  B(:,:,1)+C(:,:,1)+A(:,:,1)+C(:,:,1)*A(:,:,1)+B(:,:,1)))+...
  (1/6*(C(:,:,i)*A(:,:,2)+B(:,:,3)+C(:,:,1)+A(:,:,1)*B(:,:,2)+C(:,:,1)*A(:,:,1)*B(:,:,2)+C(:,:,1)*A(:,:,1)*B(:,:,1)))+...
  B(:,:,3)+C(:,:,2)+A(:,:,2)+C(:,:,2)+A(:,:,1)*B(:,:,2)+C(:,:,1)+A(:,:,1)*B(:,:,1)))+...
  (1/5*(C(:,:,1)*A(:,:,3)+B(:,:,3)+C(:,:,1)+A(:,:,1)*B(:,:,3)+C(:,:,1)*A(:,:,1)*B(:,:,3)+C(:,:,1)*A(:,:,1)*B(:,:,1)))+...
  B(:,:,1)+C(:,:,1)+A(:,:,1)+C(:,:,1)+A(:,:,1)+B(:,:,1)))+...
  (1/4*(C(:,:,1)*A(:,:,4)+B(:,:,3)+C(:,:,1)+A(:,:,1)*B(:,:,3)+C(:,:,1)*A(:,:,1)*B(:,:,3)+C(:,:,1)*A(:,:,1)*B(:,:,1)))+...
  B(:,:,3)+C(:,:,1)+A(:,:,1)+B(:,:,1)+C(:,:,1)+A(:,:,1)+B(:,:,1)))+...
  (1/3*(C(:,:,2)*A(:,:,4)+B(:,:,3)+C(:,:,1)+A(:,:,1)*B(:,:,3)+C(:,:,1)*A(:,:,1)*B(:,:,3)+C(:,:,1)*A(:,:,1)*B(:,:,1)))+...
  B(:,:,3)+C(:,:,1)*A(:,:,1)*B(:,:,1)+C(:,:,1)*A(:,:,1)*B(:,:,1)+C(:,:,1)*A(:,:,1)*B(:,:,1)))+...
  (1/2*(C(:,:,1)*A(:,:,4)+C(:,:,1)+B(:,:,3)+C(:,:,1)*A(:,:,1)*B(:,:,3)+C(:,:,1)*A(:,:,1)*B(:,:,3)+C(:,:,1)*A(:,:,1)*B(:,:,1)))+...
  C(:,:,1)+B(:,:,1)+C(:,:,1)+A(:,:,1)+B(:,:,1)));

end
cen_x=((sum(cen_x(:,:,i))/Area);
I.2: centroid_y

function cen_y = centroid_y(Area, num_seg,y,x);

% Computes the y centroid for the closed B-Spline curve
% Area is the area of the B-Spline curve
% num_seg is the number of segments in the B-Spline
% y is an array of the Control Points in the y
% x is an array of the Control Points in the x

E=1/6*[1 3 -3 1; 3 -6 3 0; -3 0 3 0; 1 4 1 0];
F=1/6*[3 -9 3; 6 -12 6 0; -3 0 3 0];
for i=1:num_seg
A(:,:,i)=E(:,:,i)\*x(:,:,1,i);
B(:,:,i)=F(:,:,i)\*y(:,:,1,i);
C(:,:,i)=-1*E(:,:,i)\*y(:,:,1,i);
end
clear E
clear F
for i=1:num_seg
cen_y(1,1,i)=(1/9*(C(1,1,i)*A(1,1,i)*B(1,1,i)))+... 
(1/8*(C(1,1,i)*A(1,1,i)*B(2,1,i)+C(1,1,i)*A(2,1,i)*B(1,1,i)+C(2,1,i)*A(1,1,i)*B(1,1,i)))+... 
(1/7*(C(1,1,i)*A(1,1,i)*B(3,1,i)+C(1,1,i)*A(2,1,i)*B(2,1,i)+C(1,1,i)*A(3,1,i)*B(1,1,i)+C(2,1,i)*A(1,1,i)*... 
B(2,1,i)+C(2,1,i)*A(2,1,i)*B(1,1,i)+C(3,1,i)*A(1,1,i)*B(1,1,i)))+... 
(1/6*(C(1,1,i)*A(2,1,i)*B(3,1,i)+C(1,1,i)*A(3,1,i)*B(2,1,i)+C(1,1,i)*A(4,1,i)*B(1,1,i)+C(2,1,i)*A(1,1,i)*... 
B(3,1,i)+C(2,1,i)*A(2,1,i)*B(1,1,i)+C(2,1,i)*A(3,1,i)*B(1,1,i)+C(3,1,i)*A(1,1,i)*B(1,1,i)+C(3,1,i)*A(2,1... 
B(1,1,i)+C(4,1,i)*A(1,1,i)*B(1,1,i)))+... 
(1/5*(C(1,1,i)*A(3,1,i)*B(3,1,i)+C(1,1,i)*A(4,1,i)*B(2,1,i)+C(2,1,i)*A(2,1,i)*B(3,1,i)+C(2,1,i)*A(3,1,i)*... 
B(2,1,i)+C(2,1,i)*A(4,1,i)*B(1,1,i)+C(3,1,i)*A(1,1,i)*B(3,1,i)+C(3,1,i)*A(2,1,i)*B(2,1,i)+C(3,1,i)*A(3,1... 
B(1,1,i)+C(4,1,i)*A(1,1,i)*B(2,1,i)+C(4,1,i)*A(2,1,i)*B(1,1,i)))+... 
(1/4*(C(1,1,i)*A(4,1,i)*B(3,1,i)+C(2,1,i)*A(3,1,i)*B(2,1,i)+C(2,1,i)*A(4,1,i)*B(2,1,i)+C(3,1,i)*A(2,1,i)*... 
B(3,1,i)+C(3,1,i)*A(1,1,i)*B(3,1,i)+C(4,1,i)*A(1,1,i)*B(2,1,i)+C(4,1,i)*A(1,1,i)*B(3,1,i)+C(4,1,i)*A(2,1... 
B(2,1,i)+C(4,1,i)*A(3,1,i)*B(1,1,i)))+... 
(1/3*(C(2,1,i)*A(4,1,i)*B(3,1,i)+C(3,1,i)*A(3,1,i)*B(2,1,i)+C(3,1,i)*A(4,1,i)*B(2,1,i)+C(4,1,i)*A(3,1, ... 
B(3,1,i)+C(4,1,i)*A(3,1,i)*B(2,1,i)+C(4,1,i)*A(3,1,i)*B(1,1,i)))+... 
(1/2*(C(3,1,i)*A(4,1,i)*B(3,1,i)+C(4,1,i)*A(3,1,i)*B(3,1,i)+C(4,1,i)*A(4,1,i)*B(2,1,i))+...
C(4,1,i)*A(4,1,i)*B(3,1,i);

End
cen_y=((sum(cen_y(:,:,i)))/Area);
I.3: findArea

function Area= find_area(num_seg,x,y);

% Computes the area of the closed B-Spline curve using Green's Theorem
% Area is the area of the closed B-Spline curve
% num_seg is the number of segments in the B-Spline curve
% x is an array of the control points in the x-direction for each segment
% y is an array of the control points in the y-direction for each segment

C=1/6([-1 3 -3 1; 3 -6 3 0; -3 0 3 0; 1 4 1 0]);
D=1/6([-3 9 -9 3; 6 -12 6 0; -3 0 3 0]);
for i=1:1:1num_seg
A(:,:,i)=G(:,:,i)*y(:,:,1,i);
B(:,:,i)=D(:,:,i)*x(:,:,1,i);
end
Area=Area(:,:)+Area(1,1,i)+Area(2,1,i)+Area(3,1,i)+Area(4,1,i);
clear A

clear B

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Appendix J: Derivation of the Area, Centroid, and Perimeter for a Closed B-Spline

The area of the closed B-Spline curve can be found as:

\[ A = \int y \, dx \]  \hspace{1cm} (J.1)

where \( x \) and \( y \) are parametric functions in terms of \( u \) and

\[ \frac{dx}{du} = x'(u) \]

\[ y = y(u) \]  \hspace{1cm} (J.2)

Each of the \( n+1 \) segments are defined over the interval \( 0 < u < 1 \). So, the area can be written as:

\[ A = \sum_{i=1}^{n+1} \int_{0}^{1} y(u)_i \cdot x'(u)_i \, du \]  \hspace{1cm} for a closed non-self-intersecting curve  \hspace{1cm} (J.3)

where the notation \( y(u)_i \) denotes the \( y \) position at \( u \) on the \( i^{th} \) segment.

The derivative of a B-Spline curve is written in the form of a B-Spline curve with the degree reduced by 1.

\[ \frac{d}{du} P(u) = \sum_{i = l-k+2}^{l} P_i \cdot N_{i,k-1}(u) \]  \hspace{1cm} (J.4)

where

\[ P_i^{l} = (k-1) \cdot \frac{P_i - P_{i-1}}{t_{i+k-1} - t_i} \]  \hspace{1cm} (J.5)

For a cubic B-Spline curve with \( K = 4 \) and \( l = 3 \) the derivative can be found as follows:

\[ \frac{d}{du} P(u)_l = (4 - 1) \cdot \frac{P_i - P_{i-1}}{3} \cdot N_{1,3}(u) + (4 - 1) \cdot \frac{P_{i+1} - P_1}{3} \cdot N_{2,3}(u) + (4 - 1) \cdot \frac{P_{i+2} - P_{i+1}}{3} \cdot N_{3,3}(u) \]  \hspace{1cm} (J.6)

Since the B-Spline curve is uniform with \( K = 4 \):

\[ t_{i+k-1} - t_i = 3 \]
The $N_{i,3}(u)$ terms were found to be:

\[ N_{1,3}(u) = \frac{1}{2}(u^2 - 2u + 1) \]

\[ N_{2,3}(u) = \frac{1}{2}(-2u^2 + 2u + 1) \]  \hspace{1cm} (J.7)

\[ N_{3,3}(u) = \frac{1}{2}u^2 \]

for uniform B-Spline curves.

Which can be substituted into Equation J.6

\[ \frac{d}{du} P(u) = (P_i - P_{i-1}) \left[ \frac{1}{2}(u^2 - 2u + 1) \right] + (P_{i+1} - P_i) \left[ \frac{1}{2}(-2u^2 + 2u + 1) \right] + (P_{i+2} - P_{i+1}) \left[ \frac{1}{2}u^2 \right] \]  \hspace{1cm} (J.8)

Collecting the terms

\[ \frac{d}{du} P(u) = P_{i-1} \left[ \frac{1}{2}(-u^2 + 2u - 1) \right] + P_i \left[ \frac{1}{2}(3u^2 - 4u) \right] + P_{i+1} \left[ \frac{1}{2}(-3u^2 + 2u + 1) \right] + P_{i+2} \left[ \frac{1}{2}u^2 \right] \]  \hspace{1cm} (J.9)

The derivative in the x can be defined:

\[ \frac{d}{du} x(u) = x_{i-1} \left[ \frac{1}{2}(-u^2 + 2u - 1) \right] + x_i \left[ \frac{1}{2}(3u^2 - 4u) \right] + x_{i+1} \left[ \frac{1}{2}(-3u^2 + 2u + 1) \right] + x_{i+2} \left[ \frac{1}{2}u^2 \right] \]  \hspace{1cm} (J.10)

Which can be written in Matrix form as:

\[ x'(u) = \frac{1}{2} \begin{bmatrix} u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -4 & 2 & 0 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{i-1 \text{ mod}(n+1)} \\ x_{i \text{ mod}(n+1)} \\ x_{i+1 \text{ mod}(n+1)} \\ x_{i+2 \text{ mod}(n+1)} \end{bmatrix} \]  \hspace{1cm} for 1 \leq i \leq n+1 \hspace{1cm} (J.11)

The area can then be written as:
The centroid can then be found by extending the equation for the area of the B-Spline Curve.

\[
A = \sum_{i=1}^{n+1} \left[ \int_{0}^{1} \frac{1}{6} \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} y_{i-1 \mod (n+1)} \\ y_{i \mod (n+1)} \\ y_{i+1 \mod (n+1)} \\ y_{i+2 \mod (n+1)} \end{pmatrix} \frac{1}{2} \begin{pmatrix} u^2 & u & 1 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_{i-1 \mod (n+1)} \\ x_{i \mod (n+1)} \\ x_{i+1 \mod (n+1)} \\ x_{i+2 \mod (n+1)} \end{pmatrix} \right] du
\]

(J.12)

For the \(n+1\) segments of the closed non-self-intersecting B-Spline curve, the x centroid can be found as:

\[
x_c = \frac{\sum_{i=1}^{n+1} \int_{0}^{1} x(u) y(u) x'(u) du}{\sum_{i=1}^{n+1} \int_{0}^{1} y(u) x'(u) du}
\]

(J.14)

Which was extended from Equation J.12 to yield:

\[
x_c = \frac{\sum_{i=1}^{n+1} \int_{0}^{1} \frac{1}{6} \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} y_{i-1 \mod (n+1)} \\ y_{i \mod (n+1)} \\ y_{i+1 \mod (n+1)} \\ y_{i+2 \mod (n+1)} \end{pmatrix} \frac{1}{2} \begin{pmatrix} u^2 & u & 1 \end{pmatrix} \begin{pmatrix} x_{i-1 \mod (n+1)} \\ x_{i \mod (n+1)} \\ x_{i+1 \mod (n+1)} \\ x_{i+2 \mod (n+1)} \end{pmatrix} du}{\sum_{i=1}^{n+1} \int_{0}^{1} \frac{1}{6} \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} y_{i-1 \mod (n+1)} \\ y_{i \mod (n+1)} \\ y_{i+1 \mod (n+1)} \\ y_{i+2 \mod (n+1)} \end{pmatrix} \frac{1}{2} \begin{pmatrix} u^2 & u & 1 \end{pmatrix} \begin{pmatrix} x_{i-1 \mod (n+1)} \\ x_{i \mod (n+1)} \\ x_{i+1 \mod (n+1)} \\ x_{i+2 \mod (n+1)} \end{pmatrix} du}
\]

(J.15)

Where the y centroid can be expressed as:
The arc length of the curve is used to find the perimeter. The arc length for a parametric curve can be found as:

\[ s = \int_a^b \sqrt{(f'(u))^2 + (g'(u))^2} \, du \]  

(J.17)

where \( x = f(u) \) and \( y = g(u) \), \( a \leq u \leq b \) defines a smooth curve that does not intersect itself for \( a < u < b \).

\[
\frac{dx}{du} = (u^2 \ u \ 1) \cdot \frac{1}{6} \begin{pmatrix}
-3x_0 + 9x_1 - 9x_2 + 3x_3 \\
6x_0 - 12x_1 + 6x_2 \\
-3x_0 + 3x_2
\end{pmatrix}
\]  

(J.19)

\[
\frac{dy}{du} = (u^2 \ u \ 1) \cdot \frac{1}{6} \begin{pmatrix}
-3y_0 + 9y_1 - 9y_2 + 3y_3 \\
6y_0 - 12y_1 + 6y_2 \\
-3y_0 + 3y_2
\end{pmatrix}
\]  

(J.20)

In order to simplify the expansion
\[ \frac{1}{6} \begin{pmatrix} -3x_0 + 9x_1 - 9x_2 + 3x_3 \\ 6x_0 - 12x_1 + 6x_2 \\ -3x_0 + 3x_2 \end{pmatrix} \text{ can be expressed as: } \begin{pmatrix} a \\ b \\ c \end{pmatrix} \quad (J.21) \]

and

\[ \frac{1}{6} \begin{pmatrix} -3y_0 + 9y_1 - 9y_2 + 3y_3 \\ 6y_0 - 12y_1 + 6y_2 \\ -3y_0 + 3y_2 \end{pmatrix} \text{ can be expressed as: } \begin{pmatrix} f \\ g \\ h \end{pmatrix} \quad (J.22) \]

\[ \left( \frac{dx}{du} \right)^2 \text{ and } \left( \frac{dy}{du} \right)^2 \text{ can be expressed as:} \]

\[ \left( \frac{dx}{dt} \right)^2 = \begin{pmatrix} a^2 \\ 2a \cdot b \\ 2a \cdot c \\ b^2 \\ 2b \cdot c \\ c^2 \end{pmatrix} \quad (J.23) \]

\[ \left( \frac{dy}{dt} \right)^2 = \begin{pmatrix} f^2 \\ 2f \cdot g \\ 2f \cdot h \\ g^2 \\ 2g \cdot h \\ h^2 \end{pmatrix} \quad (J.24) \]

Substituting into Equation J.18
\[ s = \int_a^b \sqrt{u^4 + u^3 u^2 + u^2 + 1} \, du \]  
(J.25)

Which can be rewritten as:

\[ s = \int_a^b \sqrt{a^2 + f^2 - 2a - b + 2f + g \cdot 2a - c + 2g + h \cdot 2b - c + 2g + h} \, du \]  
(J.26)

The total arc length of each segment can be found by integrating from \( u = 0 \) to \( u = 1 \).

\[ s = \int_a^b \sqrt{u^2 + 0 + 0 + 0 + 0 + \frac{3}{2}} \, du \]  
(J.27)
After performing the matrix multiplications the length over one segment was expressed as:

\[ s = \sqrt{a^2 + f^2 + 2ab + 2fg + 2ac + 2fh + b^2 + g^2 + 2bc + 2gh + c^2 + h^2} \]

which could have been expanded. However this form was useful in separating the \( x \) and \( y \) terms and was utilized over multiple segments in Appendix M.2.1.
Appendix K: P_x_cen_x

function u_seg = P_x_cen_x(num_u, num_seg, cen_x, P_x);

% Find all the points on the curve where the x value is equal
% to the x centroid
% Returns u_seg(:,:,n) where u_seg(1,1,n) is the u value of
% the nth point where x = xcentroid.
% u_seg(1,2,n) is the segment number that the nth point lies on.

% num_u is a scalar defining the resolution of the u values
% which defines the B-Spline. du = 1/(num_u-1)

% num_seg is the number of curve segments comprising the B-Spline Curve

% cen_x is the x-value of the centroid

% P_x is an array of the x-values evaluated at num_u points on each segment

k = 1;
i = 1;
n = 1;
u_seg = ones(1,2,1);
while (k <= num_u) & (i <= num_seg)
    if P_x(k,i) < cen_x
        T_x_cen = cen_x - 0.001;
        while T_x_cen < cen_x
            if i <= num_seg
                T_x_cen = P_x(k,i);
                k = k+1;
                if k > num_u
                    k = 1;
                    i = i+1;
                end
            else
                break
            end
        end
        if i <= num_seg
            if k==1
                k = num_u;
                i = i-1;
            else
                k = k-1;
            end
        end
    end
    if i <= num_seg
        if P_x(k,i) > cen_x
            T_x_cen = cen_x + 0.001;
            while T_x_cen > cen_x
            end
        end
    end
end

u_seg(:,:,n) = [k; i];
n = n+1;
end
if i <= num_seg
    T_x_cen = P_x(k,i);
k = k+1;
    if k > num_u
        k = 1;
i = i+1;
    end
else
    break
end
end
if i <= num_seg
    if k==1
        k = num_u;
i = i-1;
    else
        k = k-1;
    end
end
if i <= num_seg
    u_seg(:,:,n) = [k; i];
n = n+1;
end
end
if i <= num_seg
    if P_x(k,i)==cen_x
        u_seg(:,:,n) = [k; i];
k = k+1;
n = n+1;
    if k > num_u
        k = 1;
i = i+1;
    end
end
end
end
Appendix L: top_P_x_cen_x

function u_seg_start = top_P_x_cen_x(P_y, u_seg)

  % P_y is an array of the y-values evaluated at num_u points on each segment
  % u_seg_start is the u-value and the segment number of the start point.
  % u_seg(:,n) where u_seg(1,1,n) is the u value of
  % the nth point where x = xcentroid.
  % u_seg(1,2,n) is the segment number that the nth point lies on.
  n = length(u_seg(1,1,:));
  for i=1:n
    P_y_temp(1,i) = P_y(u_seg(1,1,i),u_seg(1,2,i));
    P_y_temp(2,i) = i;
    if P_y_temp(1,i) > P_y_temp(1,1);
      P_y_temp(1,1) = P_y_temp(1,i);
      P_y_temp(2,1) = P_y_temp(2,i);
    end
  end
  u_seg_start = u_seg(:,:,P_y_temp(2,1));

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Appendix M: Separating the Contours

M.1 contoursSeperate

function contoursGrouped = contoursSeperate(CT, slice_num_start)

% Group contours of the CT data.
% The open contours and the contours within another contour are eliminated.
% Knot points of the B-Splines are refit to the contours. These knot points
% will then be used to define open B-spline curves in Pro-Engineer.

if nargin == 1
    slice_num_start = 4; % default value for the starting slice
end

slice_num = slice_num_start;
k = 1;
centroidLast = [];
while slice_num <= length(CT(1,1,:))
    C = contourc(CT(:,:,slice_num),[120.5 120.5]);
    C = C';
    Ctours = [];
    i = 1;
    j = 1;
    while i<length(C(:,1))
        numpoints = C(i,2);
        Ctours(j) = C(i+1:i+numpoints,:);
        j=j+1;
        i=i+numpoints +1;
    end
    % Delete the open contours
    i=1;
    while i<=length(Ctours)
        thisContour = Ctours(i);
        if (thisContour(1,1) == thisContour(length(thisContour(:,1)),1)) &
            (thisContour(1,2) == thisContour(length(thisContour(:,1)),2))
            i=i+1;
        else
            Ctours(i) = [];
        end
    end
    % Delete any contours found inside of another contour
    i = 1;
    while i<=length(Ctours)
        inside = 0;
        iContour = Ctours(i);
        j=1;
        while j<=length(Ctours)
            jContour = Ctours(j);
            inside = inpolygon(iContour(:,1),iContour(:,2),jContour(:,1),jContour(:,2));
            if inside
                insideDep = 1;
                if insideDep
                    inside = 0;
                    iContour = Ctours(i);
                    j=1; % reset j back to 1
                end
            end
        end
    end

end
end
j=j+1;
end
if ilnsidej
Clours(i) = [];
else
i=i+1;
end
end

% From the contours closed b-spline curves will be fit through the contour's points

j=1;
while j <= length(Contours)
    ContourResampled = resampleContour(Contours(i)); % Appendix H.1
    % CP = knot_to_cntr(length(ContourResampled(:,1)),ContourResampled);
    % num_seg = length(CP(:,1));
    % num_u = 21;
    % x = x_mod(CP,num_seg);
    % y = y_mod(CP,num_seg);
    % N = N_array(num_u);
    % P_x = P_x_u_segments(N,num_seg,num_u,x);
    % P_y = P_y_u_segments(N,num_seg,num_u,y);

    [P_x, P_y] = fltBspline(ContourResampled); % Appendix G.1
    Area = flnd_area(num_seg,x,y); % Appendix I.3
    cen_x = centroid_x(Area, num_seg,x,y); % Appendix I.1
    cen_y = centroid_y(Area, num_seg,y,x); % Appendix I.2
    u_seg = P_x_cen_x(num_u, num_seg, cen_x, P_x); % Appendix K
    u_seg_start = top_P_x_cen_x(P_y, u_seg); % Appendix L
    len_s = per_spaced(x,y,u_seg_start, num_u, num_seg); % Appendix M.2
    P_s = per_spaced_pnts(x,y,num_seg,len_s,P_x,P_y,u_seg_start); % Appendix M.3
    KP_s = P_s;
    centroidCurrent(i,:) = [cen_x cen_y]; %Array of all the contours in that slice
    size_current = length(Contours);
    size_last = size(centroidLast);
    size_last = size_last(1,1);
    if size_current == size_last
        j=1;
        while j <= length(Contours)
            distance = sqrt((centroidCurrent(i,1) - centroidLast(j,1))^2 + (centroidCurrent(i,2) - centroidLast(j,2))^2);
            if j == 1
                closest = [distance 1];
            elseif distance < closest(1,1)
                closest = [distance j];
            end
            j=j+1;
        end
        contoursGrouped(:,j,1,:) = KP_s;
    elseif i == 1
        if slice_num == slice_num_start
            contoursGrouped(:,slice_num,closest(1,2)) = KP_s;
        end
    end

108

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
equalNumContoursGrouped(k) = contoursGrouped;
k = k + 1;
clear contoursGrouped;
end

closest = [0 1];
contoursGrouped(:,:,slice_num,closest(1,2)) = KP_s;
else
     closest = [0 1];
     contoursGrouped(:,:,slice_num,closest(1,2)) = KP_s;
end
i=i+1;
clear distance;
clear closest;
end

try
    centroidLast = centroidCurrent;
catch
end
slice_num = slice_num + 1;
end

figure
hold on
for i = 1:length(contoursGrouped(1,1,:))
    for j = 1:length(contoursGrouped(:,1,:))
        plane(j,1) = i;
    end
    plot3(contoursGrouped(:,1,i),contoursGrouped(:,2,i),plane(:,1))
end
M.2: per_spaced

function len_s = per_spaced(x,y,u_seg_start, num_u, num_seg);

% Create an Array of the current length of the B-spline curve (s) starting
% at u_start (the uppermost point on the curve directly above the centroid)

% This current length is taken from the start point to a point on the curve
% for each u value on every segment.

% Remember that for an Array of u values u(1)=0 and u(num_u)=1
% The actual u value for u(n) = (n-1)/(num_u-1)

% len_s is an array of the current length at a given u value for a given segment.
% This is developed starting at the upper most point above the centroid.

% len_s = (current length, u value, segment number)

u_start = (u_seg_start(1,1) - 1)/(num_u-1);
seg_start = u_seg_start(1,2);
u_1 = u_start;
i_1 = seg_start;
inc = 1/(num_u-1);
u_2 = u_1 + inc;
i_2 = i_1;

n = 1;

while i_2 <= num_seg
    while u_2 <= (1+inc)
        len_s(n,1) = perimeter(x,y,num_seg,u_1,u_2,i_1,i_2); % Appendix M.2.1
        len_s(n,2) = (u_2*(num_u-1));
        len_s(n,3) = i_2;
        n = n+1;
        u_2 = u_2+inc;
        if u_2 > 1
            u_2 = inc;
            i_2 = i_2+1;
        end
    end
    if u_2 > 1 & i_2 ~= seg_start
        u_2 = inc;
        i_2 = i_2+1;
    end
% reverse the order of finding the perimeter
i_2 = 1;
i_1 = 1;
u_1 = 0;
u_2 = inc;

while u_2 <= u_start & i_2 ~= seg_start
    while u_2 <= (1+inc)
        len_s(n,1) = perimeter(x,y,num_seg,u_start,1,seg_start,num_seg) +...
                    perimeter(x,y,num_seg,u_1,u_2,i_1,i_2);
        len_s(n,2) = (u_2*(num_u-1));
        len_s(n,3) = i_2;
        u_2 = u_2 + inc;
        n = n+1;
    end
    if u_2 > 1 & i_2+1 ~= seg_start
        u_2 = inc;
        i_2 = i_2+1;
    end
end
else
    i_2 = i_2 + 1;
end

i_1 = 1;
u_1 = 0;
u_2 = inc;
while u_2 <= u_start
    len_s(n,1) = perimeter(x,y,num_seg,u_start,1,seg_start,num_seg)+...
                 perimeter(x,y,num_seg,0,1,1,(seg_start-1)) + perimeter(x,y,num_seg,0,u_2,seg_start,seg_start);
    len_s(n,2) = (u_2*(num_u-1));
    len_s(n,3) = i_2;
    u_2 = u_2 + inc;
n = n + 1;
end
M.2.1: perimeter

```matlab
function per = perimeter(x,y,num_seg,u_1,u_2,l_1,l_2)

% x and y are arrays of the control points defining the B-Spline curve.
% num_seg is the number of segments in the B-Spline curve.
% u_1 is the u-value on segment l_1 to start taking the length from.
% u_2 is the ending u-value on segment l_2.

% Return the perimeter of the B-spline between u_1 on the l_1th segment and u_2 on the l_2th segment.
% If l_1 and l_2 are not specified it assumes l_1 = 1 and l_2 = num_seg.

if nargin == 5
    l_1 = 1;
    l_2 = num_seg;
end
D=1/6*[-3 9 -9 3; 6 -12 6 0; -3 0 3 0];
for i=1:num_seg
    A(:,:,i)=D(:,:,i);
    B(:,:,i)=D(:,:i)*y(:,i);
end
clear D
U=1;
clear per
per=zeros(1,1);(l_2-l_1+1);
i=1;
if i_1 == i_2
    i_m=1;
    per(1,1)=(1/30*sqrt((100*(A(1,1),i)*A(1,1)+B(1,1,1)+B(1,1,1))*(u_2-6)+88*A(1,1,i)*A(2,1,i)+B(1,1,1)*B(2,1,1))*u_2-5)+...
    ((450*(A(1,1),i)*A(3,1,i)+B(1,1,1)+B(3,1,1)+B(3,1,1))*(u_2-4)+...
    800*(A(2,1,1)+A(3,1,i)+B(3,1,1))*(u_2-3)+800*(A(3,1,i)+B(3,1,1))*(u_2-2)));
else
    i_m=1;
    while i <= l_2
        if i < l_1
            per(1,1)=(1/30*sqrt((100*(A(1,1,i)*A(1,1,i)+B(1,1,1)+B(1,1,1))*(1)+88*A(1,1,i)*A(2,1,i)+B(1,1,1)*B(2,1,1))*1)+...
            ((450*(A(1,1,i)+A(3,1,i)+B(3,1,1)+225*(A(2,1,i)+A(3,1,i)+B(2,1,1))*(1)+800*(A(2,1,i)+B(3,1,i))*(1)+900*(A(1,1,i)+B(1,1,1))*(1))+...
            (4500*(A(1,1,i)+A(3,1,i)+B(3,1,1))+(u_1+5)+800*(A(1,1,i)+B(1,1,1))*(u_1+4))+...
            (800*(A(2,1,i)+B(1,1,1))*(u_1+3)+900*(A(1,1,i)+B(3,1,i))*(u_1+2)))+
            i=i+1;
        end
        if i < l_2
            per(1,1)=(1/30*sqrt((100*(A(1,1,i)+B(1,1,1)+B(1,1,1))*(1)+88*A(1,1,i)*A(2,1,i)+B(1,1,1)*B(2,1,1))*1)+...
            ((450*(A(1,1,i)+A(3,1,i)+B(3,1,1)+225*(A(2,1,i)+A(3,1,i)+B(2,1,1))*(1)+800*(A(2,1,i)+B(3,1,i))*(1)+900*(A(1,1,i)+B(1,1,1))*(1))+...
            (4500*(A(1,1,i)+A(3,1,i)+B(3,1,1))+(u_1+5)+800*(A(1,1,i)+B(1,1,1))*(u_1+4))+...
            (800*(A(2,1,i)+B(1,1,1))*(u_1+3)+900*(A(1,1,i)+B(3,1,i))*(u_1+2)))+
            i=i+1;
        end
    end
    per=per(1,1);
end
per=sum(per(:,:));
```

112
function Pnts_spaced = per_spaced_pnts(x,y,num_seg,len_s,P_x,P_y,u_seg_start)

% Find the u values and segments for the points that are equally
% spaced along the perimeter of the B-Spline

% Pnts_spaced is an array of num_pnts points that are equally spaced on the B-spline.

% len_s is an array of the current length at a given u value for a given segment.
% This is developed starting at the upper most point above the centroid.
% len_s = (current length, u value, segment number)

% P_x & P_y are arrays of x and y values of points on the B-Spline curve.

% u_seg_start = (u value, segment) for the start position (upper most point above centroid)

per_full = perimeter(x,y,num_seg,0,1); %The full length of the perimeter
num_pnts = 56; %number of equally spaced points to be fit
n = 1;
size_len_s = length(len_s(:,1));

% Appended to the bottom of the array len_s are equal portions of the
% total length of the perimeter (per_full/num_pnts)*n.

while n < num_pnts
    len_s(size_len_s+n,1) = (per_full/num_pnts)*n;
    len_s(size_len_s+n,2) = nan;
    len_s(size_len_s+n,3) = nan;
    n = n+1;
end

% The equally spaced lengths are sorted into len_s (the current length at a given u values and
% segments)

len_s = sortrows(len_s);
size_len_s = length (len_s(:,1));

Pnts_spaced(1,1) = P_x(u_seg_start(1,1), u_seg_start(1,2));
Pnts_spaced(1,2) = P_y(u_seg_start(1,1), u_seg_start(1,2));

% Identify where the equal spaced lengths were sorted Into the len_s array
% and find the P_x points at u value and segment number

k = 2;
n = 2;
while n <= size_len_s
    if isnan(len_s(n,2))
        Pnts_spaced(k,1) = P_x(int8(len_s((n-1),2)),int8(len_s((n-1),3)));
        Pnts_spaced(k,2) = P_y(int8(len_s((n-1),2)),int8(len_s((n-1),3)));
        k = k+1;
    end
    n = n+1;
end
### Appendix N: Sample Ibl File

<table>
<thead>
<tr>
<th>Closed</th>
<th>Index</th>
<th>Arclength</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin section 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Begin Curve! 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>29.45420706</td>
<td>13.08039189</td>
</tr>
<tr>
<td>2</td>
<td>29.8517874</td>
<td>13.06615672</td>
</tr>
<tr>
<td>3</td>
<td>30.10621279</td>
<td>13.05011635</td>
</tr>
<tr>
<td>4</td>
<td>30.42037558</td>
<td>13.00378901</td>
</tr>
<tr>
<td>5</td>
<td>30.75845766</td>
<td>12.8895332</td>
</tr>
<tr>
<td>6</td>
<td>30.97337179</td>
<td>12.76901419</td>
</tr>
<tr>
<td>7</td>
<td>31.22522048</td>
<td>12.60377545</td>
</tr>
<tr>
<td>8</td>
<td>31.39078208</td>
<td>12.48961519</td>
</tr>
<tr>
<td>9</td>
<td>31.71053156</td>
<td>12.26521981</td>
</tr>
<tr>
<td>10</td>
<td>31.91959567</td>
<td>12.12131217</td>
</tr>
<tr>
<td>11</td>
<td>32.08056089</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>32.25568226</td>
<td>11.78795073</td>
</tr>
<tr>
<td>13</td>
<td>32.40325884</td>
<td>11.61257425</td>
</tr>
<tr>
<td>14</td>
<td>32.567211067</td>
<td>11.41572749</td>
</tr>
<tr>
<td>Begin Curve! 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>29.43430003</td>
<td>13.68219913</td>
</tr>
<tr>
<td>2</td>
<td>29.8340049</td>
<td>13.62341665</td>
</tr>
<tr>
<td>3</td>
<td>30.11745989</td>
<td>13.51777202</td>
</tr>
<tr>
<td>4</td>
<td>30.35973376</td>
<td>13.40403029</td>
</tr>
<tr>
<td>5</td>
<td>30.79774044</td>
<td>13.16764401</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>13.04862981</td>
</tr>
<tr>
<td>7</td>
<td>31.19308418</td>
<td>12.91588733</td>
</tr>
<tr>
<td>8</td>
<td>31.43329306</td>
<td>12.76041719</td>
</tr>
<tr>
<td>9</td>
<td>31.70746713</td>
<td>12.5841129</td>
</tr>
<tr>
<td>10</td>
<td>32</td>
<td>12.38439314</td>
</tr>
<tr>
<td>11</td>
<td>32.27931047</td>
<td>12.16183992</td>
</tr>
<tr>
<td>12</td>
<td>32.52163492</td>
<td>11.89956177</td>
</tr>
<tr>
<td>13</td>
<td>32.75883107</td>
<td>11.66502507</td>
</tr>
<tr>
<td>14</td>
<td>33</td>
<td>11.45273567</td>
</tr>
<tr>
<td>Begin Curve! 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>29.55931395</td>
<td>13.98453616</td>
</tr>
<tr>
<td>2</td>
<td>29.8949468</td>
<td>13.89101297</td>
</tr>
<tr>
<td>3</td>
<td>30.26755723</td>
<td>13.74175808</td>
</tr>
<tr>
<td>4</td>
<td>30.60073733</td>
<td>13.58617478</td>
</tr>
<tr>
<td>5</td>
<td>30.97628213</td>
<td>13.38477429</td>
</tr>
<tr>
<td>6</td>
<td>31.2190643</td>
<td>13.23477074</td>
</tr>
<tr>
<td>7</td>
<td>31.48167186</td>
<td>13.04835752</td>
</tr>
<tr>
<td>8</td>
<td>31.8205935</td>
<td>12.77790298</td>
</tr>
<tr>
<td>9</td>
<td>32.08384791</td>
<td>12.55100239</td>
</tr>
<tr>
<td>10</td>
<td>32.32877553</td>
<td>12.35587677</td>
</tr>
<tr>
<td>11</td>
<td>32.6170202</td>
<td>12.11690807</td>
</tr>
<tr>
<td>12</td>
<td>32.93076067</td>
<td>11.84581772</td>
</tr>
<tr>
<td>13</td>
<td>33.14555349</td>
<td>11.6475129</td>
</tr>
<tr>
<td>14</td>
<td>33.39307603</td>
<td>11.42085913</td>
</tr>
<tr>
<td>Begin Curve! 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>32.03449574</td>
<td>12.53197354</td>
</tr>
<tr>
<td>2</td>
<td>32.04251181</td>
<td>12.71789468</td>
</tr>
<tr>
<td>3</td>
<td>31.71943773</td>
<td>12.94679001</td>
</tr>
<tr>
<td>4</td>
<td>31.4689722</td>
<td>13.16523021</td>
</tr>
<tr>
<td>5</td>
<td>31.08975013</td>
<td>13.44126271</td>
</tr>
<tr>
<td>6</td>
<td>30.81062757</td>
<td>13.594391</td>
</tr>
<tr>
<td>7</td>
<td>30.56908056</td>
<td>13.70385082</td>
</tr>
<tr>
<td>8</td>
<td>30.15106071</td>
<td>13.85086318</td>
</tr>
<tr>
<td>9</td>
<td>29.75562144</td>
<td>13.96875362</td>
</tr>
<tr>
<td>10</td>
<td>29.3658133</td>
<td>14.0185546</td>
</tr>
<tr>
<td>11</td>
<td>29</td>
<td>13.9497616</td>
</tr>
<tr>
<td>12</td>
<td>28.65852259</td>
<td>13.78445579</td>
</tr>
<tr>
<td>13</td>
<td>28.38205103</td>
<td>13.60581074</td>
</tr>
<tr>
<td>14</td>
<td>28.06556153</td>
<td>13.38658924</td>
</tr>
</tbody>
</table>
Appendix O: Numbering Scheme of the Ibl Files

![Diagram of numbering scheme]

- ✅ Point to be kept
- ⚠️ Point to be trimmed

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Appendix P: Exporting to Excel

P.1: ExportToExcel.m

% Export points to excel
% Use the find and replace function in Excel to modify this file
% to obtain the correct format for an ibl file.

% Replace 999999 with 'Begin Curve'
% Replace 888888 with ''

% The file can then be saved as a txt file.
% Then in a program such as notepad the file can be saved
% with an .ibl extension.

excell = [999999 1 888888 888888];
firstPoint = 50;
umPoints = 14; %number of points in each curve
lastName = firstPoint + numPoints;
k=1;
i=firstPoint;

while k<= length(contoursGrouped_knot_points(1,1,:))
    i=firstPoint;
    while i< lastName
        lengthexcell = length(excell(:,1))+1;
        excell(lengthexcell, 2:3) = contoursGrouped_knot_points(i,:,k);
        excell(lengthexcell, 4) = k;
        excell(lengthexcell,1) = i;
        i=i+1;
    end
    k=k+1;
    excell(lengthexcell+1, :) = [999999 k 888888 888888];
end

firstcellrow = 5;
cellcolumn = 'B';
curvenumber = 1;
filename = 'indexFingerCurves8.xls'

xlsCell([cellcolumn, num2str(firstcellrow)], excell, filename); %Appendix P.2

TrimPoints = [888888 888888 888888];
k=1;
i=4;
while i < length(contoursGrouped_knot_points(:,1,1)) - 3
    k=1;
    while k<= length(contoursGrouped_knot_points(1,1,:))
        j = length(TrimPoints(:,1));
        TrimPoints(j+1,1:2) = contoursGrouped_knot_points(i,:,k);
    end
end

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
TrimPoints(i+1,3) = k;
k = k + 1;
end

dlmwrite(strcat('trimpoints',num2str(i),'.txt'),TrimPoints)
TrimPoints = [888888 888888 888888];
i = i + 7;
end
P.2: xlsceil

function xlsceil(xcell,value, varargin)

%XLSCELL Assignes value(s) to specified cell(s) in Microsoft Excel.
%
% xlsceil(xcell,value)
% xlsceil(xcell,value,filename)
% xlsceil(xcell,value,filename,sheetname)
%
% xlsceil : Assignes a value to a specified cell in excel sheet.
%
% xcell: cell name (range) (ex. C3 or A2:F19)
% value: value(s) to assign to specified cell
% filename: Name of excel file.
% sheetname: sheet name.
%
% Example:
% filename = 'file.xls';
% value = {'N1','N2','N3','N4'};
% xlsceil('C4',value);
% xlsceil('C4',value,filename);
% xlsceil('C5',rand(10,4),filename);
% xlsceil('C5',rand(10,4),filename,'Sheet1');

% Copyright 2004 Fahad Al Mahmood
% Version: 1.0 $Date: 27-Feb-2004
% 1.5 $Date: 28-Feb-2004 (multiple cell, value, &
% sheetnames added)
% 2.0 $Date: 24-May-2004 (major changes to write numerical matrices
% or cells by only specifying upper-left
% cell name or whole range)
% 2.1 $Date: 26-May-2004 (more debugging documentation)

if nargin==3
    filename=varargin{1};
    sheetname = 1;
elseif nargin==4
    filename=varargin{1};
    sheetname=varargin{2};
end

if nargin==3 | nargin==4
    [fpath,fname,fext] = fileparts(filename);
    if isempty(fpath)
        out_path = pwd;
    elseif fpath(1)=='.'
        out_path = [pwd filesep fpath];
    else
        out_path = fpath;
    end
end
% Making sure the dimensions of (value) match the dimensions of (xcell)
colon_loc = findstr(':',xcell);
if ~isempty(colon_loc)
    [nrows, ncols] = dimcells(xcell);
    if nrows ~= size(value,1) | ncols ~= size(value,2)
        error('Error! dimensions of range and values assigned does NOT match!');
    end
end

% Opening Excel
Excel = actxserver('Excel.Application');

if nargin>=3
    if ~exist(filename,'file')
        % The following case if file specified and does not exist (Creating New Workbook)
        Workbook = invoke(Excel.Workbooks,'Add');
        new = 1;
    else
        % The following case if file specified and does exist (Opening Workbook)
        Workbook = invoke(Excel.Workbooks, 'open', [out_path filesep fname fext]);
        new = 0;
    end
else
    % The following case if file is not specified (Creating New Workbook)
    Workbook = invoke(Excel.Workbooks,'Add');
    new = 1;
    set(Excel, 'Visible', 1);
end

% Writing to cell(s)
if (size(value,1)>1 | size(value,2)>1) & ~ismember(':',xcell)
    [row, col] = locate_cell(xcell);
    xcell = [xcell ':' xlcolumn(size(value,2)+xlcolumn(col)-1) num2str(size(value,1)+row-1)];
end

% Activating sheet
Sheets = Excel.Worksheets;
try
    sheet = get(Sheets, 'Item', sheetname);
catch
    invoke(Excel, 'Quit');
    delete(Excel);
    error(['Excel sheet (' sheetname ') does not exist!'])
end
invoke(sheet, 'Activate');

% testing if xcell range include more than one cell
x = findstr(xcell,':');
ExAct = Excel.Activesheet;
if isempty(x)
    ExActRange = get(ExAct,'Range',xcell,xcell);
else
    xcell_1 = xcell(1:x-1);
    xcell_2 = xcell(x+1:end);
ExActRange = get(ExAct,'Range',xcell_1,xcell_2);
end

set(ExActRange,'Value',value);

% Saving
try
  if nargin>=3
    if new
      invoke(Workbook, 'SaveAs', [out_path filesep fname fext]);
    else
      invoke(Workbook, 'Save');
    end
    invoke(Excel, 'Quit');
  end
catch
    invoke(Excel, 'Quit');
    delete(Excel);
    error('Cannot save file to specified location!');
end

delete(Excel);

function loc = xicoiumn(column)
if isnumeric(column)
  if column>256
    error('Excei is limited to 256 columns! Enter an integer number <256');
  end
  letters = {'A','B','C','D','E','F','G','H','i','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};
  count = 0;
  if column-26<=0
    loc = char(letters(column));
  else
    ocolumn = column;
    while column-26>0
      count = count + 1;
      column = column - 26;
    end
    loc = [char(letters(count)) char(letters(column))];
  end
else
  if size(column,2)==1
    loc = findstr(column,letters);
  elseif size(column,2)==2
    loc1 = findstr(column(1),letters);
    loc2 = findstr(column(2),letters);
    loc = (26 + 26*loc1)-(26-loc2);
  end
end

function [row,col] = locate_cell(cell)
col="/
row=[];
for n=1:length(cell)
if isempty(str2num(cell(n)))
    col = [col cell(n)];
else
    row = [row cell(n)];
end
end
row=str2num(row);

function [nrows,ncols]=dimcells(cell)
    letters = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'];
    colon_loc = findstr(':',cell);
    if isempty(colon_loc)
        nrows=1;
        ncols=1;
    else
        cell_1 = cell(1:colon_loc-1);
        cell_2 = cell(colon_loc+1:end);
        [cell_1_r,cell_1_c] = locate_cell(cell_1);
        [cell_2_r,cell_2_c] = locate_cell(cell_2);
        nrows = cell_2_r - cell_1_r + 1;
        ncols = cell_2_c - cell_1_c + 1;
    end